Зростання функцій. Оцінки складності алгоритмів

Дискретна математика

Лекція 13

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Характеристики зростання функцій використовуються в теоретичній інформатиці для оцінювання часової складності алгоритмів. Для спрощення викладу під *алгоритмом* ми будемо розуміти сукупність правил для розв'язування певного класу задач, які відрізняються початковими даними. Початкові дані — це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру початкових даних, то його можна визначити як функцію від обсягу вхідної інформації. Це добре ілюструють задачі сортування. У цих задачах вхідні дані — це список елементів, які підлягають сортуванню, а результат — ті самі елементи, відсортовані в порядку зростання, чи спадання. Наприклад, вхідний список

$$4, 14, 8, 1, 6, 1, 13, 11, 4$$

у випадку сортування за зростанням, буде перетворено у вихідний список

$$1, 1, 4, 4, 6, 8, 11, 13, 14.$$

Як міру обсягу вхідної інформації для алгоритму сортування природно вибирати кількість елементів, які підлягають сортуванню, або, іншими словами, довжину вхідного списку. Загалом довжина вхідних даних — придатна міра їхнього об'єму.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція*** $f(n)$ ***зі збільшенням числа*** n***?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають ***"$f(x)$ є о-велике від $g(x)$"***, і називають ***O-оцінкою***.

Як синонім терміну “вхідні дані” часто використовують термін “входи”.

Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція*** $f(n)$ ***зі збільшенням числа*** n***?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають ***“$f(x)$ є о-велике від $g(x)$”***, і називають ***O-оцінкою***.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція*** $f(n)$ ***зі збільшенням числа*** n***?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "$f(x)$ ***є о-велике від*** $g(x)$", і називають O***-оцінкою***.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція*** $f(n)$ ***зі збільшенням числа*** n***?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "$f(x)$ ***є о-велике від*** $g(x)$", і називають O***-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція $f(n)$ зі збільшенням числа n?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "***$f(x)$ є о-велике від $g(x)$***", і називають ***O-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція*** $f(n)$ ***зі збільшенням числа*** n***?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "$f(x)$ ***є о-велике від*** $g(x)$", і називають O***-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n.

Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція*** $f(n)$ ***зі збільшенням числа*** n***?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "$f(x)$ ***є о-велике від*** $g(x)$", і називають O***-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція*** $f(n)$ ***зі збільшенням числа*** n***?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "$f(x)$ ***є о-велике від*** $g(x)$", і називають O***-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція*** $f(n)$ ***зі збільшенням числа*** n***?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "$f(x)$ ***є о-велике від*** $g(x)$", і називають O***-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція*** $f(n)$ ***зі збільшенням числа*** n***?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "$f(x)$ ***є о-велике від*** $g(x)$", і називають O***-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція $f(n)$ зі збільшенням числа n?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "***$f(x)$ є о-велике від $g(x)$***", і називають ***O-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція $f(n)$ зі збільшенням числа n?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "***$f(x)$ є о-велике від $g(x)$***", і називають ***O-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція $f(n)$ зі збільшенням числа n?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "***$f(x)$ є о-велике від $g(x)$***", і називають ***O-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція $f(n)$ зі збільшенням числа n?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "***$f(x)$ є о-велике від $g(x)$***", і називають ***O-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція $f(n)$ зі збільшенням числа n?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "***$f(x)$ є о-велике від $g(x)$***", і називають ***O-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція $f(n)$ зі збільшенням числа n?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "***$f(x)$ є о-велике від $g(x)$***", і називають ***O-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція $f(n)$ зі збільшенням числа n?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "***$f(x)$ є о-велике від $g(x)$***", і називають ***O-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція $f(n)$ зі збільшенням числа n?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "$f(x)$ ***є о-велике від*** $g(x)$", і називають ***O-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція $f(n)$ зі збільшенням числа n?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають ***"$f(x)$ є о-велике від $g(x)$"***, і називають ***O-оцінкою***.

Як синонім терміну "вхідні дані" часто використовують термін "входи". Означимо ***часову складність алгоритму***, яку подальше будемо просто називати ***складністю алгоритму***, як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжини n. Іншими словами, $f(n)$ — максимальний час роботи алгоритму по всіх входах довжини n. Довжина входу, як вже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконується на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні такого питання: ***як швидко зростає функція*** $f(n)$ ***зі збільшенням числа*** n***?*** Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ використовують таке поняття.

Нехай f і g — функції з множини цілих чисел $\mathbb{Z}$ (чи з множини дійсних чисел $\mathbb{R}$) в множину дійсних чисел $\mathbb{R}$. Будемо писати, що

$$f(x) = O(g(x)),$$

якщо існують такі сталі C і k, що

$$|f(x)| \leqslant C \cdot |g(x)|, \qquad \text{для всіх} \quad x > k.$$

У цьому випадку пишуть також, що $f(x) \in O(g(x))$ і читають "$f(x)$ ***є о-велике від*** $g(x)$", і називають O***-оцінкою***.

У випадку використання поняття $O(g(x))$ функцію g у співвідношенні $f(x) = O(g(x))$ вибирають настільки малою (у розумінні повільного зростання), настільки це можливо. Здебільшого функцію g вибирають з множини функцій, які вважаються еталонними, як, наприклад, x^m, де $m \in \mathbb{N}$, чи $\log x$.

У подальшому ми майже завжди будемо мати справу з додатно визначеними функціями. Всі позначення абсолютних величин в оцінках для таких функцій можна опустити. На рис. проілюстровано співвідношення $f(x) = O(g(x))$.

У випадку використання поняття $O(g(x))$ функцію g у співвідношенні $f(x) = O(g(x))$ вибирають настільки малою (у розумінні повільного зростання), настільки це можливо. Здебільшого функцію g вибирають з множини функцій, які вважаються еталонними, як, наприклад, x^m, де $m \in \mathbb{N}$, чи $\log x$.

У подальшому ми майже завжди будемо мати справу з додатно визначеними функціями. Всі позначення абсолютних величин в оцінках для таких функцій можна опустити. На рис. проілюстровано співвідношення $f(x) = O(g(x))$.

У випадку використання поняття $O(g(x))$ функцію g у співвідношенні $f(x) = O(g(x))$ вибирають настільки малою (у розумінні повільного зростання), настільки це можливо. Здебільшого функцію g вибирають з множини функцій, які вважаються еталонними, як, наприклад, x^m, де $m \in \mathbb{N}$, чи $\log x$.

У подальшому ми майже завжди будемо мати справу з додатно визначеними функціями. Всі позначення абсолютних величин в оцінках для таких функцій можна опустити. На рис. проілюстровано співвідношення $f(x) = O(g(x))$.

У випадку використання поняття $O(g(x))$ функцію g у співвідношенні $f(x) = O(g(x))$ вибирають настільки малою (у розумінні повільного зростання), настільки це можливо. Здебільшого функцію g вибирають з множини функцій, які вважаються еталонними, як, наприклад, x^m, де $m \in \mathbb{N}$, чи $\log x$.

У подальшому ми майже завжди будемо мати справу з додатно визначеними функціями. Всі позначення абсолютних величин в оцінках для таких функцій можна опустити. На рис. проілюстровано співвідношення $f(x) = O(g(x))$.

У випадку використання поняття $O(g(x))$ функцію g у співвідношенні $f(x) = O(g(x))$ вибирають настільки малою (у розумінні повільного зростання), настільки це можливо. Здебільшого функцію g вибирають з множини функцій, які вважаються еталонними, як, наприклад, x^m, де $m \in \mathbb{N}$, чи $\log x$.

У подальшому ми майже завжди будемо мати справу з додатно визначеними функціями. Всі позначення абсолютних величин в оцінках для таких функцій можна опустити. На рис. проілюстровано співвідношення $f(x) = O(g(x))$.

У випадку використання поняття $O(g(x))$ функцію g у співвідношенні $f(x) = O(g(x))$ вибирають настільки малою (у розумінні повільного зростання), настільки це можливо. Здебільшого функцію g вибирають з множини функцій, які вважаються еталонними, як, наприклад, x^m, де $m \in \mathbb{N}$, чи $\log x$.

У подальшому ми майже завжди будемо мати справу з додатно визначеними функціями. Всі позначення абсолютних величин в оцінках для таких функцій можна опустити. На рис. проілюстровано співвідношення $f(x) = O(g(x))$.

У випадку використання поняття $O(g(x))$ функцію g у співвідношенні $f(x) = O(g(x))$ вибирають настільки малою (у розумінні повільного зростання), настільки це можливо. Здебільшого функцію g вибирають з множини функцій, які вважаються еталонними, як, наприклад, x^m, де $m \in \mathbb{N}$, чи $\log x$.

У подальшому ми майже завжди будемо мати справу з додатно визначеними функціями. Всі позначення абсолютних величин в оцінках для таких функцій можна опустити. На рис. проілюстровано співвідношення $f(x) = O(g(x))$.

У випадку використання поняття $O(g(x))$ функцію g у співвідношенні $f(x) = O(g(x))$ вибирають настільки малою (у розумінні повільного зростання), настільки це можливо. Здебільшого функцію g вибирають з множини функцій, які вважаються еталонними, як, наприклад, x^m, де $m \in \mathbb{N}$, чи $\log x$.

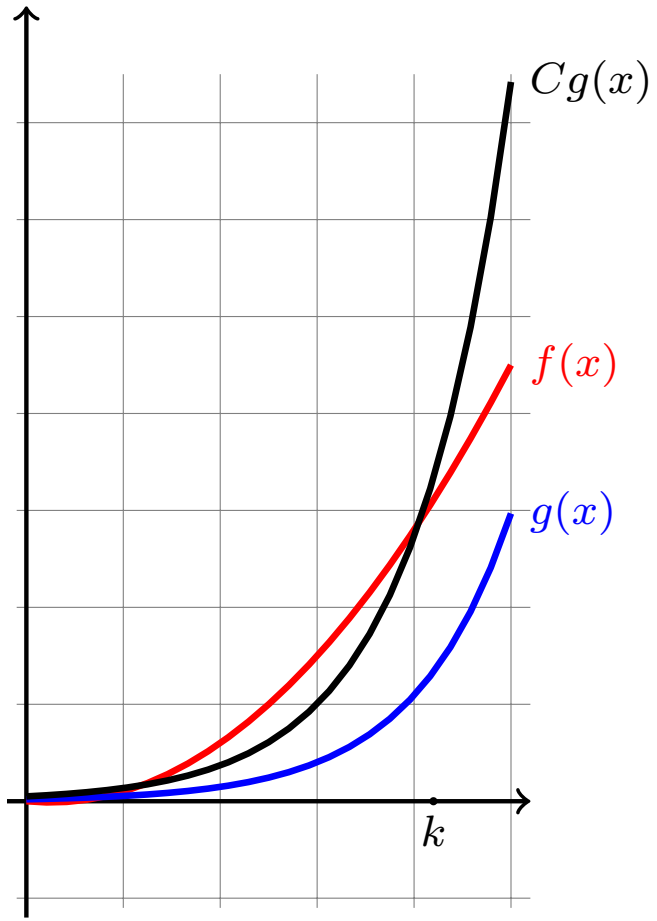
У подальшому ми майже завжди будемо мати справу з додатно визначеними функціями. Всі позначення абсолютних величин в оцінках для таких функцій можна опустити. На рис. проілюстровано співвідношення $f(x) = O(g(x))$.

У випадку використання поняття $O(g(x))$ функцію g у співвідношенні $f(x) = O(g(x))$ вибирають настільки малою (у розумінні повільного зростання), настільки це можливо. Здебільшого функцію g вибирають з множини функцій, які вважаються еталонними, як, наприклад, x^m, де $m \in \mathbb{N}$, чи $\log x$.

У подальшому ми майже завжди будемо мати справу з додатно визначеними функціями. Всі позначення абсолютних величин в оцінках для таких функцій можна опустити. На рис. проілюстровано співвідношення $f(x) = O(g(x))$.

У випадку використання поняття $O(g(x))$ функцію g у співвідношенні $f(x) = O(g(x))$ вибирають настільки малою (у розумінні повільного зростання), настільки це можливо. Здебільшого функцію g вибирають з множини функцій, які вважаються еталонними, як, наприклад, x^m, де $m \in \mathbb{N}$, чи $\log x$.

У подальшому ми майже завжди будемо мати справу з додатно визначеними функціями. Всі позначення абсолютних величин в оцінках для таких функцій можна опустити. На рис. проілюстровано співвідношення $f(x) = O(g(x))$.



Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \quad \text{і} \quad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n\text{–разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Приклад 1.10.1

Знайдіть оцінку для функцій

$$f_1(n) = n! \qquad \text{і} \qquad f_2(n) = \log n!,$$

де n — натуральне число.

Розв'язок. Очевидно, що

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leqslant \underbrace{n \cdot n \cdot n \cdot \ldots \cdot n}_{n-\text{разів}} = n^n.$$

З цієї нерівності випливає, що

$$n! = O(n^n).$$

Взявши логарифм від обох частин нерівності $n! \leqslant n^n$, отримуємо

$$\log n! \leqslant \log n^n = n \log n,$$

звідки випливає, що

$$\log n! = O(n \log n).$$

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \quad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \quad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \quad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \quad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \quad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \quad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \quad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \quad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ■

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ■

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.2

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) + f_2(x) = O(g(x)),$$

де $g(x) = \max\{g_1(x), g_2(x)\}$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) + f_2(x)| &\leqslant |f_1(x)| + |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g(x)| + C_2 \cdot |g(x)| \leqslant \\ &\leqslant (C_1 + C_2) \cdot |g(x)| = \\ &= C|g(x)|, \end{aligned}$$

де $g(x) = \max\{g_1(x), g_2(x)\}$ і $C = C_1 + C_2$.

Отож,

$$|f_1(x) + f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ■

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \quad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \quad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \quad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \quad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \qquad \text{і} \qquad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \qquad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ∎

Теорема 1.10.3

Нехай $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$. Тоді

$$f_1(x) \cdot f_2(x) = O(g(x)),$$

де $g(x) = g_1(x) \cdot g_2(x)$.

Доведення. Якщо $f_1(x) = O(g_1(x))$ і $f_2(x) = O(g_2(x))$, то існують такі сталі C_1, C_2, k_1 і k_2, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)|, \qquad \text{якщо } x > k_1$$

і

$$|f_2(x)| \leqslant C_2 \cdot |g_2(x)|, \qquad \text{якщо } x > k_2.$$

Тоді, очевидно, що

$$|f_1(x)| \leqslant C_1 \cdot |g_1(x)| \quad \text{і} \quad |f_2(x)| \leqslant C_2 \cdot |g_2(x)| \quad \text{для } x > k,$$

де $k = \max\{k_1, k_2\}$. Отже, для $x > k$ маємо

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leqslant |f_1(x)| \cdot |f_2(x)| \leqslant \\ &\leqslant C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| \leqslant \\ &\leqslant C_1 \cdot C_2 \cdot |g(x)| = \\ &= C \cdot |g(x)|, \end{aligned}$$

де $g(x) = g_1(x) \cdot g_2(x)$ і $C = C_1 \cdot C_2$. Отож,

$$|f_1(x) \cdot f_2(x)| \leqslant C \cdot |g(x)| \qquad \text{для} \quad x > k = \max\{k_1, k_2\},$$

що і треба було довести. ■

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n \log(n!) + (n^3 + 2n^2 + 3) \log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n \log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n \log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n \log(n!) = O(n^2 \log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3) \log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3) \log n = O(n^3 \log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n \log(n!) + (n^3 + 2n^2 + 3) \log n = \\ &= O\left(\max\left\{n^2 \log n, n^3 \log n\right\}\right) = \\ &= O(n^3 \log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n \log(n!) + (n^3 + 2n^2 + 3) \log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n \log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n \log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n \log(n!) = O(n^2 \log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3) \log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3) \log n = O(n^3 \log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n \log(n!) + (n^3 + 2n^2 + 3) \log n = \\ &= O\left(\max\left\{n^2 \log n, n^3 \log n\right\}\right) = \\ &= O(n^3 \log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n\log(n!) + (n^3 + 2n^2 + 3)\log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n\log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n\log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n\log(n!) = O(n^2\log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3)\log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3)\log n = O(n^3\log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n\log(n!) + (n^3 + 2n^2 + 3)\log n = \\ &= O\left(\max\left\{n^2\log n, n^3\log n\right\}\right) = \\ &= O(n^3\log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n\log(n!) + (n^3 + 2n^2 + 3)\log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n\log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n\log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n\log(n!) = O(n^2\log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3)\log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3)\log n = O(n^3\log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n\log(n!) + (n^3 + 2n^2 + 3)\log n = \\ &= O\left(\max\left\{n^2\log n, n^3\log n\right\}\right) = \\ &= O(n^3\log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n\log(n!) + (n^3 + 2n^2 + 3)\log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n\log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n\log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n\log(n!) = O(n^2\log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3)\log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3)\log n = O(n^3\log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n\log(n!) + (n^3 + 2n^2 + 3)\log n = \\ &= O\left(\max\left\{n^2\log n, n^3\log n\right\}\right) = \\ &= O(n^3\log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n\log(n!) + (n^3 + 2n^2 + 3)\log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n\log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n\log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n\log(n!) = O(n^2\log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3)\log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3)\log n = O(n^3\log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n\log(n!) + (n^3 + 2n^2 + 3)\log n = \\ &= O\left(\max\left\{n^2\log n, n^3\log n\right\}\right) = \\ &= O(n^3\log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n\log(n!) + (n^3 + 2n^2 + 3)\log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n\log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n\log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n\log(n!) = O(n^2\log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3)\log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3)\log n = O(n^3\log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n\log(n!) + (n^3 + 2n^2 + 3)\log n = \\ &= O\left(\max\left\{n^2\log n, n^3\log n\right\}\right) = \\ &= O(n^3\log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n \log(n!) + (n^3 + 2n^2 + 3) \log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n \log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n \log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n \log(n!) = O(n^2 \log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3) \log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3) \log n = O(n^3 \log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n \log(n!) + (n^3 + 2n^2 + 3) \log n = \\ &= O\left(\max\left\{n^2 \log n, n^3 \log n\right\}\right) = \\ &= O(n^3 \log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n \log(n!) + (n^3 + 2n^2 + 3) \log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n \log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n \log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n \log(n!) = O(n^2 \log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3) \log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3) \log n = O(n^3 \log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n \log(n!) + (n^3 + 2n^2 + 3) \log n = \\ &= O\left(\max\left\{n^2 \log n, n^3 \log n\right\}\right) = \\ &= O(n^3 \log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n \log(n!) + (n^3 + 2n^2 + 3) \log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n \log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n \log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n \log(n!) = O(n^2 \log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3) \log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3) \log n = O(n^3 \log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n \log(n!) + (n^3 + 2n^2 + 3) \log n = \\ &= O\left(\max\left\{n^2 \log n, n^3 \log n\right\}\right) = \\ &= O(n^3 \log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n\log(n!) + (n^3 + 2n^2 + 3)\log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n\log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n\log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n\log(n!) = O(n^2\log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3)\log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3)\log n = O(n^3\log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n\log(n!) + (n^3 + 2n^2 + 3)\log n = \\ &= O\left(\max\left\{n^2\log n, n^3\log n\right\}\right) = \\ &= O(n^3\log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n \log(n!) + (n^3 + 2n^2 + 3) \log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n \log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n \log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n \log(n!) = O(n^2 \log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3) \log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3) \log n = O(n^3 \log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n \log(n!) + (n^3 + 2n^2 + 3) \log n = \\ &= O\left(\max\left\{n^2 \log n, n^3 \log n\right\}\right) = \\ &= O(n^3 \log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n \log(n!) + (n^3 + 2n^2 + 3) \log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n \log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n \log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n \log(n!) = O(n^2 \log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3) \log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3) \log n = O(n^3 \log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n \log(n!) + (n^3 + 2n^2 + 3) \log n = \\ &= O\left(\max\left\{n^2 \log n, n^3 \log n\right\}\right) = \\ &= O(n^3 \log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n \log(n!) + (n^3 + 2n^2 + 3) \log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n \log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n \log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n \log(n!) = O(n^2 \log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3) \log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3) \log n = O(n^3 \log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n \log(n!) + (n^3 + 2n^2 + 3) \log n = \\ &= O\left(\max\left\{n^2 \log n, n^3 \log n\right\}\right) = \\ &= O(n^3 \log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n \log(n!) + (n^3 + 2n^2 + 3) \log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n \log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n \log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n \log(n!) = O(n^2 \log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3) \log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3) \log n = O(n^3 \log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n \log(n!) + (n^3 + 2n^2 + 3) \log n = \\ &= O\left(\max\left\{n^2 \log n, n^3 \log n\right\}\right) = \\ &= O(n^3 \log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n \log(n!) + (n^3 + 2n^2 + 3) \log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n \log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n \log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n \log(n!) = O(n^2 \log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3) \log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3) \log n = O(n^3 \log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n \log(n!) + (n^3 + 2n^2 + 3) \log n = \\ &= O\left(\max\left\{n^2 \log n, n^3 \log n\right\}\right) = \\ &= O(n^3 \log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n \log(n!) + (n^3 + 2n^2 + 3) \log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n \log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n \log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n \log(n!) = O(n^2 \log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3) \log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3) \log n = O(n^3 \log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n \log(n!) + (n^3 + 2n^2 + 3) \log n = \\ &= O\left(\max\left\{n^2 \log n, n^3 \log n\right\}\right) = \\ &= O(n^3 \log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n\log(n!) + (n^3 + 2n^2 + 3)\log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n\log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n\log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n\log(n!) = O(n^2\log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3)\log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3)\log n = O(n^3\log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n\log(n!) + (n^3 + 2n^2 + 3)\log n = \\ &= O\left(\max\left\{n^2\log n, n^3\log n\right\}\right) = \\ &= O(n^3\log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n\log(n!) + (n^3 + 2n^2 + 3)\log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n\log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n\log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n\log(n!) = O(n^2\log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3)\log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3)\log n = O(n^3\log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n\log(n!) + (n^3 + 2n^2 + 3)\log n = \\ &= O\left(\max\left\{n^2\log n, n^3\log n\right\}\right) = \\ &= O(n^3\log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n \log(n!) + (n^3 + 2n^2 + 3) \log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n \log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n \log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n \log(n!) = O(n^2 \log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3) \log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3) \log n = O(n^3 \log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n \log(n!) + (n^3 + 2n^2 + 3) \log n = \\ &= O\left(\max\left\{n^2 \log n, n^3 \log n\right\}\right) = \\ &= O(n^3 \log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n\log(n!) + (n^3 + 2n^2 + 3)\log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n\log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n\log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n\log(n!) = O(n^2\log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3)\log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3)\log n = O(n^3\log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n\log(n!) + (n^3 + 2n^2 + 3)\log n = \\ &= O\left(\max\left\{n^2\log n, n^3\log n\right\}\right) = \\ &= O(n^3\log n). \end{aligned}$$

Приклад 1.10.4

Знайдіть оцінку для функцій

$$f(n) = 5n\log(n!) + (n^3 + 2n^2 + 3)\log n,$$

де n — натуральне число.

Розв'язок. Оцінимо спочатку функцію $5n\log(n!)$. З виведеної в прикладі 1.10.1 оцінки функції $\log n!$ маємо

$$\log n! = O(n\log n).$$

Беручи до уваги цю оцінку і той факт, що

$$5n = O(n),$$

з теореми 1.10.3 випливає, що

$$5n\log(n!) = O(n^2\log n).$$

Оцінимо тепер функцію $(n^3 + 2n^2 + 3)\log n$. Оскільки

$$n^3 + 2n^2 + 3 < 3n^3 \qquad \text{для} \quad n > 3,$$

то маємо, що

$$n^3 + 2n^2 + 3 = O(n^3).$$

З теореми 1.10.3 випливає, що

$$(n^3 + 2n^2 + 3)\log n = O(n^3\log n).$$

Використавши теорему 1.10.2, остаточно отримуємо

$$\begin{aligned} f(n) &= 5n\log(n!) + (n^3 + 2n^2 + 3)\log n = \\ &= O\left(\max\left\{n^2\log n, n^3\log n\right\}\right) = \\ &= O(n^3\log n). \end{aligned}$$

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned} |p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\ &\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\ &= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\ &\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right) \end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned} |p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\ &\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\ &= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\ &\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right) \end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned} |p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\ &\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\ &= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\ &\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right) \end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned} |p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\ &\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\ &= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\ &\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right) \end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned} |p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\ &\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\ &= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\ &\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right) \end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned} |p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\ &\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\ &= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\ &\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right) \end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned} |p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\ &\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\ &= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\ &\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right) \end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned} |p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\ &\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\ &= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\ &\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right) \end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned} |p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\ &\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\ &= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\ &\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right) \end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ∎

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned} |p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\ &\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\ &= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\ &\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right) \end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ∎

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned}
|p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\
&\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\
&= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\
&\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right)
\end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned} |p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\ &\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\ &= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\ &\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right) \end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned}
|p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\
&\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\
&= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\
&\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right)
\end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned}
|p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\
&\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\
&= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\
&\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right)
\end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned}
|p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\
&\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\
&= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\
&\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right)
\end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned}
|p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\
&\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\
&= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\
&\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right)
\end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ∎

Поліноми часто використовуються для оцінок зростання функцій. Наведемо твердження, яке завжди можна використати для оцінки зростання полінома.

Теорема 1.10.5

Нехай

$$p(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0,$$

де $a_m, a_{m-1}, \ldots, a_1, a_0$ — дійсні числа. Тоді

$$p(x) = O(x^m).$$

Доведення. Враховуючи нерівність, що модуль суми не перевищує суми модулів дійсних чисел, то маємо

$$\begin{aligned}
|p(x)| &= \left|a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0\right| \leqslant \\
&\leqslant |a_m| x^m + |a_{m-1}| x^{m-1} + \cdots + |a_1| x + |a_0| = \\
&= x^m \left(|a_m| + \frac{|a_{m-1}|}{x} + \cdots + \frac{|a_1|}{x^{m-1}} + \frac{|a_0|}{x^m}\right) \leqslant \\
&\leqslant x^m \left(|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|\right)
\end{aligned}$$

для $x > 1$. Звідси випливає, що

$$|p(x)| \leqslant C x^m,$$

де $C = |a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$ і $x > 1$. Отже, ми довели, що для поліному $p(x)$ виконується оцінка $p(x) = O(x^m)$. ■

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Вираз "складність алгоритму є (відп., дорівнює, складає) $O(g(n))$" означає, що функцією $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n — довжина входу. Як еталони оцінок складності алгоритмів використовують такі функції:

$$1,\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

Ці функції ми записали в порядку зростання складності.

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу. Алгоритм зі складністю $O(n)$ називають ***лінійним***. Такий алгоритм для переважної більшості задач є найкращим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — деякий поліном, називають ***поліноміальним***. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m — стала. Особливу роль поліноміальних алгоритмів ми опише далі. Поняття "поліноміальний алгоритм" тепер є найпоширенішою формалізацією поняття "ефективний алгоритм". Зауважимо, що всі задачі дискретної математики, які вважаються важкими для алгоритмічного розв'язування, на даний час не мають поліноміальних алгоритмів.

Задача в дискретній математиці називається ***важко розвя'зуваною***, якщо для її розв'язку не існує поліноміального алгоритму. Алгоритми, часова складність яких не піддається подібній оцінці, називаються ***експоненціальними***. Більшість експоненціальних алгоритмів — це просто варіанти "повного перебору", тоді як поліноміальні алгоритми здебільшого можна побудувати лише в тому випадку, коли можна заглибитися в сутність задачі.

Вправа 1.10.1

Нехай $F(x)$, $g(x)$ і $h(x)$ — такі функції, що $f(x) = O(g(x))$ і $g(x) = O(h(X))$. Доведіть, що $f(x) = O(h(x))$.

Вправа 1.10.2

Для довільного натурального числа k доведіть оцінку

$$1^n + 2^n + \ldots + n^k = O(n^{k+1}).$$

Вправа 1.10.3

Для наведених нижче функцій дайте найкращу O-оцінку, наскільки це можливо:

1) $(n^2 + 8)(n + 12)$;
2) $(n \log n + n^2)(n^2 + 3)$;
3) $(n! + 2^n)(n^3 + \log(n^2 + 21))$.

Задача в дискретній математиці називається ***важко розвя'зуваною***, якщо для її розв'язку не існує поліноміального алгоритму. Алгоритми, часова складність яких не піддається подібній оцінці, називаються ***експоненціальними***. Більшість експоненціальних алгоритмів — це просто варіанти "повного перебору", тоді як поліноміальні алгоритми здебільшого можна побудувати лише в тому випадку, коли можна заглибитися в сутність задачі.

Вправа 1.10.1

Нехай $F(x)$, $g(x)$ і $h(x)$ — такі функції, що $f(x) = O(g(x))$ і $g(x) = O(h(X))$. Доведіть, що $f(x) = O(h(x))$.

Вправа 1.10.2

Для довільного натурального числа k доведіть оцінку

$$1^n + 2^n + \ldots + n^k = O(n^{k+1}).$$

Вправа 1.10.3

Для наведених нижче функцій дайте найкращу O-оцінку, наскільки це можливо:

1) $(n^2 + 8)(n + 12)$;
2) $(n \log n + n^2)(n^2 + 3)$;
3) $(n! + 2^n)(n^3 + \log(n^2 + 21))$.

Задача в дискретній математиці називається ***важко розвя'зуваною***, якщо для її розв'язку не існує поліноміального алгоритму. Алгоритми, часова складність яких не піддається подібній оцінці, називаються ***експоненціальними***. Більшість експоненціальних алгоритмів — це просто варіанти "повного перебору", тоді як поліноміальні алгоритми здебільшого можна побудувати лише в тому випадку, коли можна заглибитися в сутність задачі.

Вправа 1.10.1

Нехай $F(x)$, $g(x)$ і $h(x)$ — такі функції, що $f(x) = O(g(x))$ і $g(x) = O(h(X))$. Доведіть, що $f(x) = O(h(x))$.

Вправа 1.10.2

Для довільного натурального числа k доведіть оцінку

$$1^n + 2^n + \ldots + n^k = O(n^{k+1}).$$

Вправа 1.10.3

Для наведених нижче функцій дайте найкращу O-оцінку, наскільки це можливо:

1) $(n^2 + 8)(n + 12)$;
2) $(n \log n + n^2)(n^2 + 3)$;
3) $(n! + 2^n)(n^3 + \log(n^2 + 21))$.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Задача в дискретній математиці називається ***важко розвя'зуваною***, якщо для її розв'язку не існує поліноміального алгоритму. Алгоритми, часова складність яких не піддається подібній оцінці, називаються ***експоненціальними***. Більшість експоненціальних алгоритмів — це просто варіанти "повного перебору", тоді як поліноміальні алгоритми здебільшого можна побудувати лише в тому випадку, коли можна заглибитися в сутність задачі.

Вправа 1.10.1

Нехай $F(x)$, $g(x)$ і $h(x)$ — такі функції, що $f(x) = O(g(x))$ і $g(x) = O(h(X))$. Доведіть, що $f(x) = O(h(x))$.

Вправа 1.10.2

Для довільного натурального числа k доведіть оцінку

$$1^n + 2^n + \ldots + n^k = O(n^{k+1}).$$

Вправа 1.10.3

Для наведених нижче функцій дайте найкращу O-оцінку, наскільки це можливо:

1) $(n^2 + 8)(n + 12)$;
2) $(n \log n + n^2)(n^2 + 3)$;
3) $(n! + 2^n)(n^3 + \log(n^2 + 21))$.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Задача в дискретній математиці називається ***важко розвя'зуваною***, якщо для її розв'язку не існує поліноміального алгоритму. Алгоритми, часова складність яких не піддається подібній оцінці, називаються ***експоненціальними***. Більшість експоненціальних алгоритмів — це просто варіанти "повного перебору", тоді як поліноміальні алгоритми здебільшого можна побудувати лише в тому випадку, коли можна заглибитися в сутність задачі.

Вправа 1.10.1

Нехай $F(x)$, $g(x)$ і $h(x)$ — такі функції, що $f(x) = O(g(x))$ і $g(x) = O(h(X))$. Доведіть, що $f(x) = O(h(x))$.

Вправа 1.10.2

Для довільного натурального числа k доведіть оцінку

$$1^n + 2^n + \ldots + n^k = O(n^{k+1}).$$

Вправа 1.10.3

Для наведених нижче функцій дайте найкращу O-оцінку, наскільки це можливо:

1) $(n^2 + 8)(n + 12)$;
2) $(n \log n + n^2)(n^2 + 3)$;
3) $(n! + 2^n)(n^3 + \log(n^2 + 21))$.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Задача в дискретній математиці називається ***важко розвя'зуваною***, якщо для її розв'язку не існує поліноміального алгоритму. Алгоритми, часова складність яких не піддається подібній оцінці, називаються ***експоненціальними***. Більшість експоненціальних алгоритмів — це просто варіанти "повного перебору", тоді як поліноміальні алгоритми здебільшого можна побудувати лише в тому випадку, коли можна заглибитися в сутність задачі.

Вправа 1.10.1

Нехай $F(x)$, $g(x)$ і $h(x)$ — такі функції, що $f(x) = O(g(x))$ і $g(x) = O(h(X))$. Доведіть, що $f(x) = O(h(x))$.

Вправа 1.10.2

Для довільного натурального числа k доведіть оцінку

$$1^n + 2^n + \ldots + n^k = O(n^{k+1}).$$

Вправа 1.10.3

Для наведених нижче функцій дайте найкращу O-оцінку, наскільки це можливо:

1) $(n^2 + 8)(n + 12)$;
2) $(n \log n + n^2)(n^2 + 3)$;
3) $(n! + 2^n)(n^3 + \log(n^2 + 21))$.

Задача в дискретній математиці називається ***важко розвя'зуваною***, якщо для її розв'язку не існує поліноміального алгоритму. Алгоритми, часова складність яких не піддається подібній оцінці, називаються ***експоненціальними***. Більшість експоненціальних алгоритмів — це просто варіанти "повного перебору", тоді як поліноміальні алгоритми здебільшого можна побудувати лише в тому випадку, коли можна заглибитися в сутність задачі.

Вправа 1.10.1

Нехай $F(x)$, $g(x)$ і $h(x)$ — такі функції, що $f(x) = O(g(x))$ і $g(x) = O(h(X))$. Доведіть, що $f(x) = O(h(x))$.

Вправа 1.10.2

Для довільного натурального числа k доведіть оцінку

$$1^n + 2^n + \ldots + n^k = O(n^{k+1}).$$

Вправа 1.10.3

Для наведених нижче функцій дайте найкращу O-оцінку, наскільки це можливо:

1) $(n^2 + 8)(n + 12)$;
2) $(n \log n + n^2)(n^2 + 3)$;
3) $(n! + 2^n)(n^3 + \log(n^2 + 21))$.

Лекція 13: Зростання функцій. Оцінки складності алгоритмів

Задача в дискретній математиці називається ***важко розвя'зуваною***, якщо для її розв'язку не існує поліноміального алгоритму. Алгоритми, часова складність яких не піддається подібній оцінці, називаються ***експоненціальними***. Більшість експоненціальних алгоритмів — це просто варіанти "повного перебору", тоді як поліноміальні алгоритми здебільшого можна побудувати лише в тому випадку, коли можна заглибитися в сутність задачі.

Вправа 1.10.1

Нехай $F(x)$, $g(x)$ і $h(x)$ — такі функції, що $f(x) = O(g(x))$ і $g(x) = O(h(X))$. Доведіть, що $f(x) = O(h(x))$.

Вправа 1.10.2

Для довільного натурального числа k доведіть оцінку

$$1^n + 2^n + \ldots + n^k = O(n^{k+1}).$$

Вправа 1.10.3

Для наведених нижче функцій дайте найкращу O-оцінку, наскільки це можливо:

1) $(n^2 + 8)(n + 12)$;
2) $(n \log n + n^2)(n^2 + 3)$;
3) $(n! + 2^n)(n^3 + \log(n^2 + 21))$.

Задача в дискретній математиці називається ***важко розвя'зуваною***, якщо для її розв'язку не існує поліноміального алгоритму. Алгоритми, часова складність яких не піддається подібній оцінці, називаються ***експоненціальними***. Більшість експоненціальних алгоритмів — це просто варіанти "повного перебору", тоді як поліноміальні алгоритми здебільшого можна побудувати лише в тому випадку, коли можна заглибитися в сутність задачі.

Вправа 1.10.1

Нехай $F(x)$, $g(x)$ і $h(x)$ — такі функції, що $f(x) = O(g(x))$ і $g(x) = O(h(X))$. Доведіть, що $f(x) = O(h(x))$.

Вправа 1.10.2

Для довільного натурального числа k доведіть оцінку

$$1^n + 2^n + \ldots + n^k = O(n^{k+1}).$$

Вправа 1.10.3

Для наведених нижче функцій дайте найкращу O-оцінку, наскільки це можливо:

1) $(n^2 + 8)(n + 12)$;
2) $(n \log n + n^2)(n^2 + 3)$;
3) $(n! + 2^n)(n^3 + \log(n^2 + 21))$.

Дякую за увагу!!!