МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Кафедра комп’ютерної інженерії та електроніки

ЗВІТ З ПРАКТИЧНИХ РОБІТ

з навчальної дисципліни

«Алгоритми та методи обчислень»

Тема «Асимптотична складність алгоритмів. О-нотація»

Студент гр. КН-23-1 ПІБ Люлін В.О

Викладач к. т. н., доц. ПІБ Сидоренко В.М

Кременчук 2024

**ЗМІСТ**

[1 Розв'язання задачі (варіант 13) 3](#_Toc160564053)

[2 Розв'язання задачі (варіант 3) 4](#_Toc160564054)

[3 Відповіді на контрольні запитання 4](#_Toc160564055)

[4 Висновок 6](#_Toc160564056)

# Розв'язання задачі (варіант 13)

Постановка задачі: Задано функції 𝑓(𝑛) = 100𝑛 3 + 8 та 𝑔(𝑛) = 𝑛^3 . Доведіть, що 𝑓(𝑛) = 𝑂(𝑔(𝑛)).

Маю функцію 𝑓 (𝑛) = 100𝑛 3 + 8 та 𝑔(𝑛) = 𝑛^3. Ми хочемо довести, що 𝑓(𝑛) = 𝑂(𝑔(𝑛)). потрібно показати, що існують такі додатні константи c і *n*0​, що для всіх 𝑛≥𝑛0 ​ виконується нерівність: *f*(*n*)≤*c* ⋅ *g*(*n*).

Спочатку спростимо нерівність:

100*n^*3+8≤*c*⋅ *n^*3.

Винесемо 𝑛^3 за дужки з лівої сторони:

*n^*3(100+*8/n^3*​) ≤ *c*⋅ *n^*3.

Поділимо обидві сторони на 𝑛^3 (за умови, що 𝑛>0):

(100+ (*8/ n^3))* ​≤*c.*

Тепер розглянемо доданок 8 / 𝑛^3. Як *n* зростає, цей доданок прямує до нуля. Отже, для достатньо великого 𝑛, 8 / 𝑛^3​ буде дуже малим.

8 / 1^3= 8.

Таким чином, при *n*≥1 нерівність набуває вигляду:

100 + 8 ≤ c.

108 ≤ c.

Отже, якщо вибрати 𝑐=108, то для всіх *n*≥1 нерівність 100𝑛^3 + 8 ≤ 108^3 буде виконуватися.

Таким чином, я довів, що 𝑓(𝑛)=100𝑛^3+8 належить до 𝑂(𝑛^3) з константами 𝑐=108 і *n*0​=1.

# Розв'язання задачі (варіант 3)

Постановка задачі: Дано функцію 𝑓(𝑛) = 𝑙𝑜𝑔(𝑛) + 2𝑛 2 + 11. Знайти асимптотичну складність у 𝑂-нотації.

Щоб знайти асимптотичну складність функції 𝑓(𝑛) = 𝑙𝑜𝑔(𝑛) + 2𝑛 2 + 11 у *O*-нотації, потрібно визначити, який із доданків визначає поведінку функції при великих значеннях *n*.

Розглянемо всі доданки функції:

1. log(𝑛)— логарифмічний доданок.
2. 2𝑛^2 — квадратичний доданок.
3. 11 — константа.

При великих значеннях *n* квадратичний доданок 2𝑛^2 буде найбільшим у порівнянні з іншими. А саме:

* Логарифмічний доданок log(𝑛) зростає набагато повільніше, ніж квадратичний.
* Константа 11 не впливає на асимптотичну поведінку функції при великих значеннях *n*.

Отже, для великих значень *n* доданок 2𝑛^2 домінує і визначає асимптотичну складність функції 𝑓(𝑛).

Ми можемо записати:

𝑓(𝑛)=log(𝑛)+2𝑛^2+11

Для великих значень *n*:

𝑓(𝑛)≈2𝑛^2

Отже, асимптотична складність функції 𝑓(𝑛) в *O*-нотації буде:

𝑓(𝑛)=𝑂(𝑛^2)

Таким чином, я довів, що, функція 𝑓(𝑛) має асимптотичну складність 𝑂(𝑛^2)

# Відповіді на контрольні запитання

1. Що таке асимптотична складність алгоритму?

**Асимптотична складність алгоритму - це поняття з теорії обчислень, яке визначає швидкість зростання часових та/або просторових вимог алгоритму залежно від розміру вхідних даних. Коли ми говоримо про асимптотичну складність, ми дивимося на те, як швидко зростає кількість операцій, які виконує алгоритм, або як швидко збільшується потреба в пам'яті при збільшенні розміру вхідних даних.**

1. Яким чином визначається 𝑂-нотація і яка її сутність?

**Асимптотична складність алгоритму визначає, як змінюється час виконання або кількість операцій алгоритму при збільшенні розміру вхідних даних. 𝑂-нотація використовується для оцінки верхньої межі складності алгоритму та вказує, як швидко зростає функція при збільшенні розміру вхідних даних.**

1. Які основні правила використання 𝑂-нотації при аналізі алгоритмів?

**- Визначення границі: 𝑂-нотація визначає верхню межу складності алгоритму. Вона описує, як швидко зростає час виконання або кількість операцій алгоритму при збільшенні розміру вхідних даних.**

**- Відсутність точності: 𝑂-нотація надає асимптотичну оцінку складності алгоритму, а не точний аналіз. Вона вказує на тенденцію росту складності при збільшенні вхідних даних.**

**- Фокус на найгіршому випадку: Зазвичай 𝑂-нотація оцінює складність алгоритму у найгіршому випадку. Це допомагає забезпечити гарантії щодо швидкості виконання алгоритму у всіх сценаріях використання.**

**- Відкидання констант: При визначенні 𝑂-нотації ігноруються константи та менш значущі члени функцій. Основна увага приділяється головним компонентам, що впливають на складність алгоритму при зростанні розміру вхідних даних.**

**- Розділення складностей: Якщо алгоритм складається з декількох етапів або підзадач, то складність кожного етапу аналізується окремо, а потім об'єднується для отримання загальної складності алгоритму.**

1. Що означають вирази 𝑂(1), 𝑂(𝑛), 𝑂(𝑛^2 ) в контексті асимптотичної складності?

**У контексті асимптотичної складності, такі вирази, як 𝑂(1), 𝑂(𝑛), 𝑂(𝑛^2) використовуються для визначення верхньої межі часової складності алгоритмів у відношенні до розміру вхідних даних (𝑛).**

1. Яким чином визначити асимптотичну складність алгоритму за його кодом або математичним виразом?

**Визначення асимптотичної складності алгоритму за його кодом або математичним виразом використовується для визначення того, як швидко зростає час виконання алгоритму зі збільшенням розміру вхідних даних.**

# Висновок

В аналізі алгоритмів велике значення має розуміння асимптотичної складності. Цей підхід дозволяє оцінити, як швидко зростає час виконання або кількість операцій алгоритму при збільшенні розміру вхідних даних.

У вищезгаданих прикладах було продемонстровано визначення асимптотичної складності за допомогою O-нотації. В обох випадках функції було показано, що вони належать до класу O(n^3), тобто зростають не швидше, ніж квадратично при збільшенні обсягу вхідних даних.

У першому прикладі функція 𝑓(𝑛) = 100𝑛 3 + 8 та 𝑔(𝑛) = 𝑛^3 та у другому - 𝑓(𝑛) = 𝑙𝑜𝑔(𝑛) + 2𝑛 2 + 11були обмежені зверху функціями n^3, знайденими відповідними константами C та n0. Це дозволяє робити висновки про те, що обидва алгоритми мають квадратичну асимптотичну складність.

Аналіз асимптотичної складності є важливим етапом в процесі розробки програмного забезпечення, оскільки він допомагає вибрати найефективніші алгоритмічні рішення та прогнозувати їхню продуктивність при роботі з різними обсягами даних. Такий підхід дозволяє розробникам та інженерам створювати оптимальні програмні рішення, які ефективно працюватимуть у реальних умовах використання.