МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ   
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРИЧНОЇ ІНЖЕНЕРІЇ   
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА АВТОМАТИЗАЦІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

НАВЧАЛЬНА ДИСЦИПЛІНА  
Алгоритми та методи обчислень

ЗВІТ

З ПРАКТИЧНИХ РОБІТ

Виконав:

студент групи КН-23-1

Сидоренко А.Ю.

Кременчук 2024

**Практична робота № 3**

**Тема.** Алгоритми сортування та їх складність. Порівняння алгоритмів сортування

**Мета:** опанувати основні алгоритми сортування та навчитись методам аналізу їх асимптотичної складності.

**Хід роботи**

**Завдання:**

1. Вивчити самостійно і записати (будь-яким способом) алгоритм бульбашкового сортування:

**def** bubble\_sort(arr):

n **=** len(arr)

**for** i **in** range(n):

swapped **=** **False**

**for** j **in** range(0, n**-**i**-**1):

**if** arr[j] **>** arr[j**+**1]:

arr[j], arr[j**+**1] **=** arr[j**+**1], arr[j]

swapped **=** **True**

**if** **not** swapped:

**break**

**return** arr

Оцінити асимптотику алгоритму сортування методом бульбашки в найгіршому і в найкращому випадку:

У найкращому випадку, коли масив уже відсортовано, асимптотична складність сортування бульбашки:

У найгіршому випадку, коли масив відсортовано в зворотному порядку, асимптотична складність сортування бульбашки:

Порівняти за цими показниками бульбашковий алгоритм з алгоритмом сортування вставлянням:

Згідно графіку, на невеликій кількості чисел масиву, оба алгоритми є ефективними, але на великих об’ємах чисел, алгоритм вставляння є більш ефективним. Алгоритм вставляння потребує більше пам’яті, але має часову складність O(log n), що працює швидше ніж

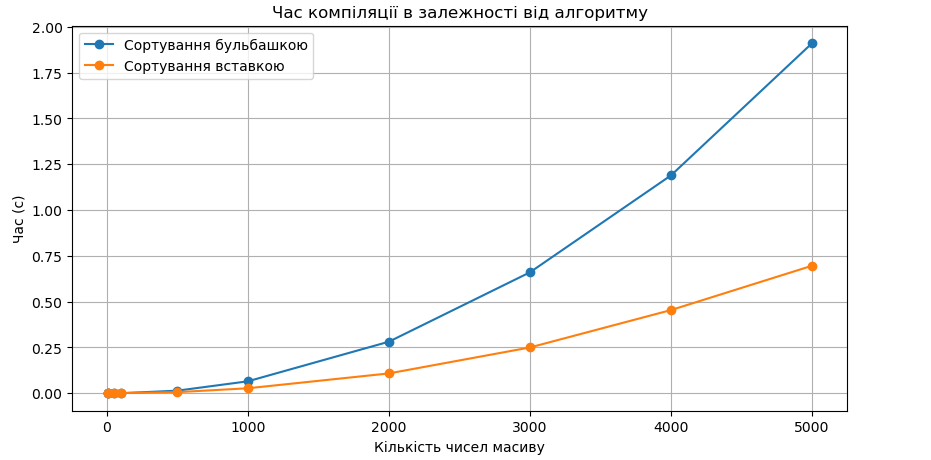


Рис 1 Графік залежності з Лаб3

Чому на практиці бульбашковий алгоритм виявляється менш ефективним у порівнянні з сортуванням методом зливанням? :

Сортування методом зливанням виявляється більш ефективним у більшості практичних застосувань завдяки своїй оптимізованій часовій складності, незважаючи на більшу потребу в пам'яті.

1. Оцінити асимптотичну складність алгоритму сортування зливанням, скориставшись основною теоремою рекурсії.

Бо відштовхуючись від рекурентного співвідношення для часової складності алгоритму сортування злиттям: T(n) = 2 \* T(n/2) + O(n) та скориставшись основною теоремою рекурсії отримуємо:

**Асимптотична складність алгоритму сортування злиттям є O(n \* log n).**

1. Вивчити і записати (будь-яким способом) самостійно алгоритм швидкого сортування. Оцінити асимптотичну складність алгоритму швидкого сортування, скориставшись основною теоремою рекурсії.

def quick\_sort(arr):

if len(arr) <= 1:

return arr

pivot = arr[len(arr) // 2]

left = [x for x in arr if x < pivot]

middle = [x for x in arr if x == pivot]

right = [x for x in arr if x > pivot]

return quick\_sort(left) + middle + quick\_sort(right)

Записавши рекурентне співвідношення T(n) = T(k) + T(n-k-1) + O(n), і скориставшись скориставшись основною теоремою рекурсії отримуємо:

**Що асимптотична складність алгоритму сортування злиттям у найкращому випадку, коли опорний елемент ділить масив на дві приблизно рівні частини є O(n \* log n)**

**А у найгіршому, опорний елемент є найменшим або найбільшим елементом масиву**

**Контрольні питання**

1. **Що таке асимптотична складність алгоритму сортування і чому вона важлива для порівняння алгоритмів?**

Асимптотична складність алгоритму сортування - це спосіб оцінки часової складності алгоритму в залежності від розміру вхідних даних.

Вона важлива для порівняння алгоритмів сортування, оскільки дозволяє зрозуміти, як швидко зростає час виконання або необхідна пам'ять при збільшенні розміру вхідних даних. Це допомагає вибрати найбільш ефективний алгоритм для конкретних задач, враховуючи обмеження ресурсів, такі як час виконання і доступна пам'ять.

1. **Які алгоритми сортування мають квадратичну складність у найгіршому випадку? Поясніть, чому це може бути проблемою для великих обсягів даних.**

Алгоритми які мають квадратичну складність у найгіршому випадку:

* Сортування бульбашкою
* Сортування вставкою
* Сортування вибором

Алгоритми сортування з квадратичною складністю у найгіршому випадку можуть бути неефективними для великих обсягів даних через значний час виконання, що зростає квадратично з розміром масиву.

Виконання алгоритму з квадратичною складністю може вимагати значних ресурсів процесора, для дуже великих масивів час виконання може стати непрактично довгим і квадратичні алгоритми можуть бути прийнятними для невеликих масивів, але для великих масивів значно ефективнішими є алгоритми з лінійно-логарифмічною складністю.

1. **В чому полягає перевага сортування злиттям над сортуванням вставками для великих наборів даних?**

Для великих наборів даних сортування злиттям має значну перевагу над сортуванням вставками завдяки своїй нижчій асимптотичній складності, ефективності в обробці великих масивів даних, можливості паралелізму та стабільності. Сортування вставками, хоча й простий і ефективний алгоритм для невеликих або майже відсортованих масивів, не є придатним для роботи з великими обсягами даних через свою квадратичну складність.

1. **Які алгоритми сортування використовуються для сортування списків у стандартних бібліотеках мов програмування, таких як Python, Java або C++?**

Python:

В стандартній бібліотеці Python для сортування списків за замовчуванням використовується алгоритм Timsort. Timsort є комбінацією сортування злиттям (Merge Sort) та сортування вставками (Insertion Sort). Він ефективний для різноманітних типів даних та добре пристосовується до різних розмірів масивів.

Java:

В стандартній бібліотеці Java для сортування використовується метод Arrays.sort(), який зазвичай використовує алгоритм Dual-Pivot Quicksort для примітивних типів даних та Timsort для об'єктів. Dual-Pivot Quicksort - це вдосконалена версія алгоритму швидкого сортування (Quicksort), яка забезпечує швидку та стабільну роботу на практиці.

C++:

У стандартній бібліотеці C++ для сортування використовується функція sort(), яка часто реалізована з використанням адаптивного алгоритму сортування, як от Introsort або Timsort. Introsort - це комбінація алгоритмів швидкого сортування (Quicksort), сортування злиттям (Merge Sort) та сортування вставками (Insertion Sort), що забезпечує швидке сортування для різних випадків даних.

1. **Яка різниця між алгоритмами сортування злиттям і швидким сортуванням? У яких випадках краще використовувати кожен з цих алгоритмів?**

Сортування злиттям може бути кращим вибором, коли потрібно забезпечити стабільність сортування або коли оброблюються дуже великі масиви даних, які не вміщуються в оперативну пам'ять.

Швидке сортування може бути більш ефективним, коли робиться припущення, що вхідні дані рівномірно розподілені та коли пам'ять обмежена, оскільки воно використовує менше додаткової пам'яті. Воно також може бути корисним, коли важливо мінімізувати час виконання у середньому, а не в найгіршому випадку.

1. **Які фактори слід враховувати при виборі алгоритму сортування для конкретної задачі?**

Вибір алгоритму сортування для конкретної задачі залежить від різних факторів, які можуть включати обсяг даних, тип даних, характеристики пам'яті, швидкодію, стабільність та інші вимоги до роботи програми.

Ми обираємо алгоритми за такими параметрами як: швидкодія, стабільність, використання пам’яті, розмір масиву пам’яті та дані які використовуємо