МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Навчалньно-науковий інститут електричної інженерії та інформаційних технологій

Кафедра автоматизації та інформаційних систем

**ПРАКТИЧНА РОБОТА**

Виконав: студент групи КІ-23-1

Черниш В’ячеслав Олександрович

Перевірив:

Сидоренко Валерій Миколайович

м. Кременчук

2024 рік

**Практична робота № 3**

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

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

**Завдання**

**1.** Вивчити самостійно і записати (будь-яким способом) алгоритм бульбашкового сортування. Оцінити асимптотику алгоритму сортування методом бульбашки в найгіршому і в найкращому випадку. Порівняти за цими показниками бульбашковий алгоритм з алгоритмом сортування вставлянням. Чому на практиці бульбашковий алгоритм виявляється менш ефективним у порівнянні з сортуванням методом зливанням?

**Хід роботи**

**1.** Вивчаємо і записуємо (будь-яким способом) алгоритм бульбашкового сортування.

def bubble\_sort(arr):

n = len(arr)

for i in range(n - 1):

for j in range(n - i - 1):

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

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

return arr

**1.2** Оцінюємо асимптотику алгоритму сортування методом бульбашки в найгіршому і в найкращому випадку.

**Найгірший випадок**:  
Коли масив відсортований у зворотному порядку, кожен елемент потрібно переміщувати до кінця масиву. Всього буде здійснено O(n2) порівнянь і обмінів..

Сумарна кількість порівнянь у цьому випадку буде приблизно:

Отже, асимптотична складність у найгіршому випадку: O(n2).

**Найкращий випадок**:  
Якщо масив вже відсортований, в кожному проході не буде жодного обміну. Але все одно потрібно пройти через весь масив, що дає складність O(n2).

У такому випадку зовнішній цикл виконується лише один раз, а внутрішній проходить по n - 1 елементам.

Отже, асимптотична складність у найкращому випадку: O(n).

**1.3** Порівнюємо за цими показниками бульбашковий алгоритм з алгоритмом сортування вставлянням.

**Найгірший випадок для сортування вставками**: O(n2) коли елементи масиву вже відсортовані у зворотному порядку.

**Найкращий випадок для сортування вставками**: O(n) якщо масив вже відсортований, оскільки кожен новий елемент буде лише порівнюватися з останнім елементом.

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

Так як алгоритм зливання має асимптотичну складність nlog(n) (рисунок 1),

є величезна різниця у кількості операцій для великих n.

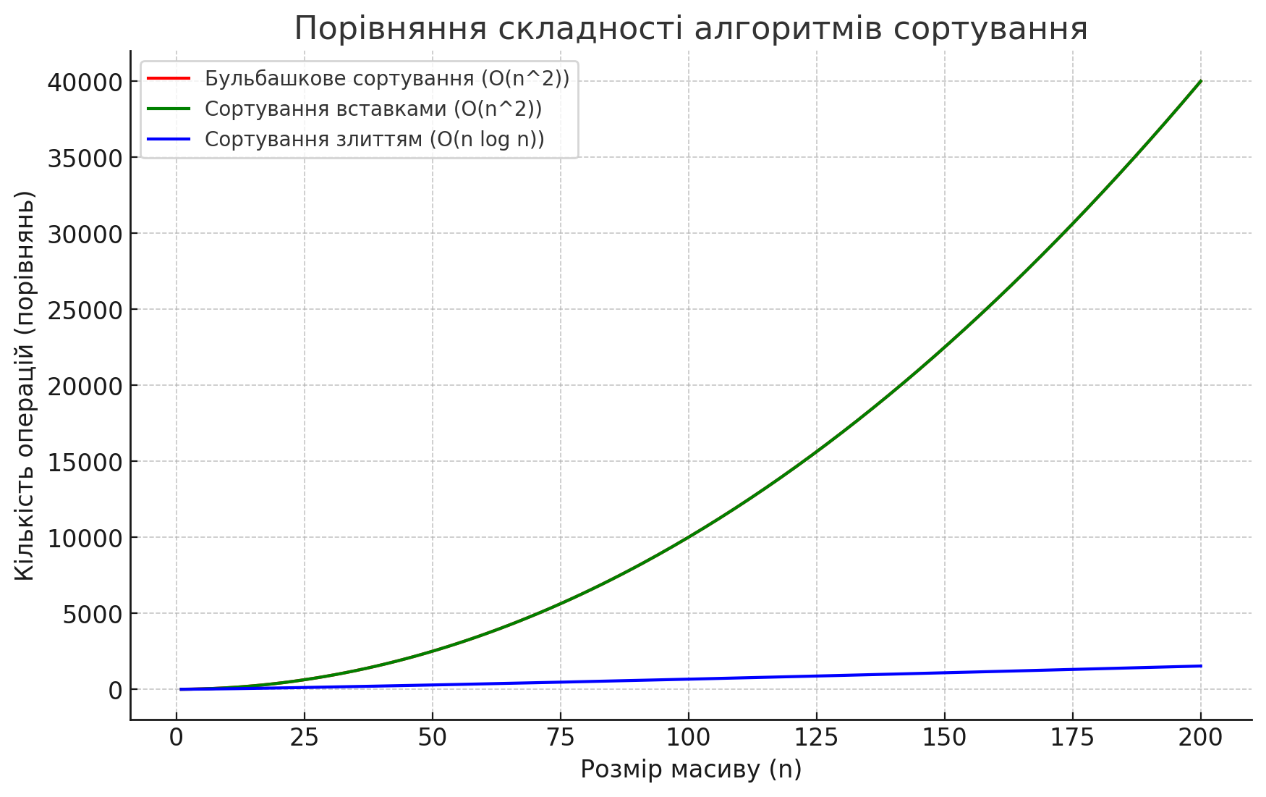


Рисунок 1 – Графік порівняння алгоритмів

**2.** Оцінюємо асимптотичну складність алгоритму сортування зливанням, користуючись основною теоремою рекурсії.

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2 # Знайдемо середину масиву

left\_half = arr[:mid] # Ліва половина

right\_half = arr[mid:] # Права половина

merge\_sort(left\_half) # Рекурсивно сортуємо ліву частину

merge\_sort(right\_half) # Рекурсивно сортуємо праву частину

i = j = k = 0

# Зливаємо два відсортовані масиви в один

while i < len(left\_half) and j < len(right\_half):

if left\_half[i] < right\_half[j]:

arr[k] = left\_half[i]

i += 1

else:

arr[k] = right\_half[j]

j += 1

k += 1

# Якщо залишились елементи в лівому масиві

while i < len(left\_half):

arr[k] = left\_half[i]

i += 1

k += 1

# Якщо залишились елементи в правому масиві

while j < len(right\_half):

arr[k] = right\_half[j]

j += 1

k += 1

return arr

Оцінка складності алгоритму сортування злиттям. Рекурсивний процес. На кожному кроці рекурсії масив розбивається на дві частини, тому кожна підзадача має розмір . Розділення триває до тих пір, поки розмір підмасиву не зменшиться до 1. Кількість рівнів рекурсії становить log2​(n), оскільки на кожному рівні масив поділяється на дві частини.

Операція злиття. Кожен раз, коли два відсортовані підмасиви об'єднуються, ми виконуємо O(n) операцій, оскільки потрібно пройти по всіх елементах обох підмасивів і розмістити їх в правильному порядку. Рекурсивне рівняння для цього алгоритму можна записати так:

де: a = 2 — ми розділяємо масив на дві частини;

b = 2 — кожна підзадача має розмір ​;

O(n) — час на злиття двох відсортованих підмасивів.

Тепер можна застосувати основну теорему рекурсії для оцінки складності:

Згідно з теоремою рекурсії, для рівнянь, колиз a = bd

( a = 2, b = 2 і d = 1), складність буде:

Отже асимптотична складність алгоритму сортування зливанням у найгіршому, найкращому та середньому випадках є однаковою і дорівнює:

Це значно ефективніше, ніж алгоритми з складністю O(n2), як бульбашкове сортування або сортування вставками, особливо для великих масивів.

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

Вивчаємо і записуємо (будь-яким способом) самостійно алгоритм швидкого сортування.

def quick\_sort(arr):

if len(arr) <= 1:

return arr

pivot = arr[len(arr) // 2] # Вибір опорного елемента

left = [x for x in arr if x < pivot] # Елементи менші за опорний

middle = [x for x in arr if x == pivot] # Елементи, рівні опорному

right = [x for x in arr if x > pivot] # Елементи більші за опорний

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

**3.1** Оцінюємо асимптотичну складність алгоритму швидкого сортування, скориставшись основною теоремою рекурсії.

Рекурсивне рівняння для алгоритму швидкого сортування можна записати як:

де:

a=2— кількість рекурсивних підзадач на кожному рівні;

b=2— кожна підзадача має розмір приблизно , якщо опорний елемент вибраний вдало;

O(nd)= = O(n)— час, необхідний для розбиття масиву відносно опорного елемента.

Отже, рівняння виглядає так:

Тепер застосуємо основну теорему рекурсії для рівняння вигляду

У нашому випадку:

a=2,

b=2,

nd=n, отже, d=1.

Обчислюємо bd:

bd=21=2

Порівнюємо значення aaa та bd:

1. Якщо a<bd, то T(n)=O(nd);
2. Якщо a=bd, то T(n)=O(ndlogn));
3. Якщо a>bd, то T(n)=O().

Оскільки a=bd, застосовується другий випадок, і складність алгоритму швидкого сортування буде:

T(n)=O(nlogn)

**Контрольні питання**

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

Асимптотична складність — це характеристика, яка описує, як змінюється час виконання або споживання пам'яті алгоритму в залежності від розміру вхідних даних. Важливість:

Оцінка ефективності: Порівняння алгоритмів без урахування конкретної реалізації чи апаратного забезпечення.

Масштабованість: Визначає, як алгоритм поводиться з великими обсягами даних.

Вибір алгоритму: Допомагає вибрати оптимальний алгоритм для конкретної задачі.

1. **Які алгоритми сортування мають квадратичну складність у найгіршому випадку?**

Алгоритми з квадратичною складністю O(n2)у найгіршому випадку:

Bubble Sort — O(n2) порівнянь при зворотньому порядку.

Selection Sort — O(n2) порівнянь незалежно від порядку.

Insertion Sort — O(n2) при зворотньому порядку.

Проблеми квадратичної складності: Швидке зростання витрат часу на великих даних і погана ефективність для великих наборів.

**Перевага сортування злиттям над сортуванням вставками для великих наборів даних?**

Сортування злиттям має складність O(nlogn), що робить його значно ефективнішим для великих даних порівняно з сортуванням вставками, яке має складність O(n2). Сортування злиттям стабільне незалежно від початкового порядку даних.

.

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

Таблиця 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Мова | Функція | Алгоритм | Стабільність | Часова складність |
| Python | sorted(), .sort() | Timsort | Так |  |
| Java | Arrays.sort() (прим.) | Dual-Pivot QuickSort | Ні |  |
| Java | Arrays.sort() (об'єкти) | Timsort | Так |  |
| C++ | std::sort() | Introsort (QuickSort + HeapSort) | Ні |  |
| C++ | std::stable\_sort() | Merge Sort | Так |  |

1. **Яка різниця між алгоритмами сортування злиттям і швидким сортуванням?**

**У яких випадках краще використовувати кожен з цих алгоритмів?**

**Сортування злиттям:**Підходить для задач, де важлива стабільність, робота із зовнішньою пам'яттю або уникнення найгіршого часу.

Коли використовувати:Коли потрібна стабільність (наприклад, сортування записів бази даних).

Для дуже великих даних, коли масив не може вміститися в пам'ять (зовнішнє сортування).

Для прогнозованої продуктивності, якщо важливо уникнути погіршення в найгіршому випадку.

**Швидке сортування**:Кращий вибір у більшості практичних задач, якщо виконуються оптимізації (наприклад, вибір хорошого опорного елемента).

Коли використовувати:Для високої швидкості в середньому випадку.

Якщо важлива продуктивність і не потрібно забезпечувати стабільність.

Коли обмежені ресурси пам'яті (потрібно менше пам'яті, ніж у Merge Sort).

Для малих наборів даних, де накладні витрати швидкого сортування менші.

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

**При виборі алгоритму сортування варто враховувати такі фактори**:

Розмір даних.

Структура вихідних даних.

Вимоги до стабільності сортування.

Обмеження на пам'ять.

Час виконання в найгіршому випадку.

Потреба в багатопоточності.

Зовнішнє застосування.

Реалізація та підтримка.

Вибір алгоритму залежить від специфіки даних, середовища виконання та вимог до продуктивності й пам'яті. Для загальних задач зазвичай рекомендується використовувати оптимізовані реалізації стандартних бібліотек.

.