

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ ІМЕНІ МИХАЙЛА
ОСТРОГРАДСЬКОГО

Кафедра комп'ютерної інженерії та електроніки

ЗВІТ З ПРАКТИЧНОЇ РОБОТИ № 3

з навчальної дисципліни

«Алгоритми та структури даних»

Тема «Алгоритми сортування та їх складність.

Порівняння алгоритмів сортування»

Студентка гр. КН-24-1 Бояринцова П. С.

Викладач Сидоренко В. М.

Тема роботи: Алгоритми сортування та їх складність. Порівняння алгоритмів сортування

1.1 Постановка завдання

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

Завдання: реалізувати алгоритми сортування

1.2 Розв'язання завдань

Завдання 1.

1. Записати алгоритм бульбашкового сортування. Оцінити асимптотику алгоритму. Порівняти бульбашковий алгоритм з алгоритмом сортування вставлянням.
2. Чому на практиці бульбашковий алгоритм виявляється менш ефективним у порівнянні з сортуванням методом зливанням?

Розв'язання:

def bubble_sort(nums):
 n = len(nums)
 for i in range(n-1):
 swapped = False
 for j in range(n-i-1):
 if nums[j] > nums[j+1]:
 nums[j], nums[j+1] = nums[j+1], nums[j]
 swapped = True
 if not swapped:
 break

У найгіршому випадку:

1) C_1	4) $\sum_{i=0}^{n-2} C_4 \cdot (n-i-1) = C_4 \cdot \frac{(n-1)n}{2}$
2) $C_2(n-1)$	5) $\sum_{i=0}^{n-2} C_5(n-i-1) = C_5 \cdot \frac{(n-1)n}{2}$
3) $C_3(n-1)$	6) $C_6 \cdot \frac{(n-1)n}{2}$
	7) $C_7 \cdot \frac{(n-1)n}{2}$
	8) $C_8(n-1)$
	9) C_9

$$T_{worst}(n) = C_1 + C_2(n-1) + C_3(n-1) + C_4 \frac{(n-1)n}{2} + C_5 \frac{(n-1)n}{2} + C_6 \frac{(n-1)n}{2} + C_7 \frac{(n-1)n}{2} + C_8(n-1) + C_9$$

$$= (C_2 + C_3 + C_4 + C_5 + C_6 + C_7) \cdot \frac{n^2}{2} - (C_2 + C_3 + C_4 + C_5 + C_6 + C_7) \cdot n + C_1 + C_8(n-1) + C_9$$

$$= \frac{(C_2 + C_3 + C_4 + C_5 + C_6 + C_7)}{2} n^2 - \frac{(C_2 + C_3 + C_4 + C_5 + C_6 + C_7)}{2} n + C_1 + C_8(n-1) + C_9$$

$-(C_2 + C_3 + C_4 + C_5 + C_6 + C_7) = an^2 + bn - c = n^2$
 $n \rightarrow \infty$
 $T_{worst}(n) = O(n^2)$

У найкращому випадку:

1) C_1	4) $C_4(n-1)$
2) $C_2(n-1)$	5) 0; 6) 0; 7) 0
3) $C_3(n-1)$	8) $C_8(n-1)$; 9) C_9

$$T_{best}(n) = C_1 + C_2(n-1) + C_3(n-1) + C_4(n-1) + 0 + 0 + 0 + C_8(n-1) + C_9 = (C_2 + C_3 + C_4 + C_8)n - (C_2 + C_3 + C_4 + C_8) + C_1 + C_9$$

$$= (C_2 + C_3 + C_4 + C_8)n - (C_2 + C_3 + C_4 + C_8) + C_1 + C_9$$

$$n \rightarrow \infty$$

$$T_{best}(n) = O(n)$$

Рисунок 1 – Бульбашкове сортування

def insertion_sort(nums):
 for i in range(1, len(nums)):
 key = nums[i]
 j = i - 1
 while j >= 0 and nums[j] > key:
 nums[j+1] = nums[j]
 j -= 1
 nums[j+1] = key

Case	ki-ma pass
C ₁	n-1
C ₂	n-1
C ₃	n-1
C ₄	$\sum_{j=1}^{n-1} t_j$
C ₅	$\sum_{j=1}^{n-1} t_j$
C ₆	$\sum_{j=1}^{n-1} t_j$
C ₇	n-1

$$1) C_1(n-1) \quad 4) \sum_{j=1}^{n-1} t_j = \frac{(n-1)n}{2}; \quad C_4 \cdot \frac{(n-1)n}{2}$$

$$2) C_2(n-1) \quad 5) C_5 \cdot \frac{(n-1)n}{2}; \quad 6) C_6 \cdot \frac{(n-1)n}{2}$$

$$3) C_3(n-1) \quad 7) C_7(n-1)$$

$$T_{worst}(n) = C_1(n-1) + C_2(n-1) + C_3(n-1) + C_4 \cdot \frac{(n-1)n}{2} + C_5 \cdot \frac{(n-1)n}{2} + C_6 \cdot \frac{(n-1)n}{2} + C_7(n-1) =$$

$$= (C_1 + C_2 + C_3 + C_7) \cdot (n-1) + \left(\frac{C_4 + C_5 + C_6}{2} \right) \cdot (n-1)n =$$

$$= (C_1 + C_2 + C_3 + C_7) \cdot (n-1) + (C_4 + C_5 + C_6) \cdot \frac{(n-1)n}{2} =$$

$$= (C_1 + C_2 + C_3 + C_7) \cdot n - (C_1 + C_2 + C_3 + C_7) + \frac{(C_4 + C_5 + C_6)}{2} \cdot n^2 - \frac{(C_4 + C_5 + C_6)}{2} \cdot n =$$

$$= \left(\frac{C_4 + C_5 + C_6}{2} \right) \cdot n^2 - \frac{(C_4 + C_5 + C_6)}{2} \cdot n + (C_1 + C_2 + C_3 + C_7) \cdot n - (C_1 + C_2 + C_3 + C_7) =$$

$$-(C_1 + C_2 + C_3 + C_7) = an^2 + bn - c = n^2$$

$$n \rightarrow \infty; T_{worst}(n) = O(n^2)$$

Найкращий випадок

$$1) C_1(n-1) \quad 4) C_4(n-1) \quad 7) C_7(n-1)$$

$$2) C_2(n-1) \quad 5) 0$$

$$3) C_3(n-1) \quad 6) 0$$

$$T_{best} = C_1(n-1) + C_2(n-1) + C_3(n-1) + C_4(n-1) + 0 + 0 + C_7(n-1) = (C_1 + C_2 + C_3 + C_4 + C_7)(n-1) =$$

$$= (C_1 + C_2 + C_3 + C_4 + C_7) \cdot n - (C_1 + C_2 + C_3 + C_4 + C_7) =$$

$$= an - c = n$$

$$n \rightarrow \infty$$

$$T_{best}(n) = O(n)$$

Рисунок 2 – Алгоритм вставлянням

Чому Bubble Sort менш ефективний, ніж Merge Sort?

- Bubble Sort завжди проходить по масиву кілька разів, роблячи велику кількість перестановок.
- Merge Sort працює по принципу "розділяй і володарюй" та виконує лише мінімальну кількість копіювань.
- Merge Sort має гарантовану складність $O(n \log n)$, яка набагато краща за $O(n^2)$ у Bubble Sort.

Завдання 2.

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

$$\begin{aligned}
 a &= 2 \\
 b &= 2 \\
 d &= 1 \\
 T(n) &= 2T\left(\frac{n}{2}\right) + O(n) \\
 F(n) &= \begin{cases} O(n^d), & \text{якщо } d > \log_b a \\ O(n^d \log n), & \text{якщо } d = \log_b a \\ O(n^{\log_b a}), & \text{якщо } d < \log_b a \end{cases} \\
 1 &= \log_2 2 \\
 1 &= 1 \\
 O(n^1 \log n) \\
 T(n) &= O(n \log n)
 \end{aligned}$$

Рисунок 3 – Сортвання зливанням

Завдання 3.

Записати алгоритм швидкого сортування. Оцінити асимптотичну складність алгоритму.

Швидке сортування
Псевдокод:

```

QuickSort(A, p, q)
if p > q return;
r ← A[p];
i ← p-1;
j ← q+1;
while i < j do
  repeat
    i ← i+1
  until A[i] ≥ r
  repeat
    j ← j-1
  until A[j] ≤ r
  if i < j then Swap A[i] ↔ A[j]
QuickSort(A, p, i-1)
QuickSort(A, j+1, q)

```

Рекурсивний метод
Найгірший випадок:

$$T(n) = T(n-1) + T(0) + O(n) = T(n-1) + O(n)$$

$$T(n) = O(n^2)$$

Найкращий випадок:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$\begin{aligned}
 a &= 2 \\
 b &= 2 \\
 d &= 1 \\
 1 &= \log_2 2 \\
 1 &= 1 \\
 O(n^1 \log n) \\
 T(n) &= O(n \log n)
 \end{aligned}$$

Рисунок 4 – Швидке сортування

1.3 Відповіді на контрольні питання

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

Асимптотична складність показує, скільки часу або пам'яті потребує алгоритм, коли розмір даних (n) стає дуже великим.

Чому це важливо:

Дає змогу порівнювати алгоритми між собою. Наприклад, один алгоритм працює швидше за інший при великих масивах, і це видно саме по складності.

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

Алгоритми з $O(n^2)$:

Бульбашкове сортування (Bubble Sort)

Сортування вибором (Selection Sort)

Сортування вставками (Insertion Sort)

Чому погано:

При великих масивах (наприклад, з мільйоном елементів) — ці алгоритми працюють дуже повільно. Це може займати години замість секунд.

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

Сортування злиттям (Merge Sort) завжди має складність $O(n \log n)$ — швидко.

Сортування вставками (Insertion Sort) може працювати як $O(n^2)$ — повільно.

Висновок:

Сортування злиттям краще підходить для великих масивів, бо працює стабільно і швидше.

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

- Python: Timsort (поєднання злиття та вставок)
- Java: Timsort для об'єктів, QuickSort для чисел
- C++: Introsort (поєднання QuickSort, HeapSort і вставок)

Ці алгоритми підібрані так, щоб працювати швидко в реальних задачах.

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

	Merge Sort	Quick Sort
Складність	Завжди $O(n \log n)$	Найгірше — $O(n^2)$, але часто $O(n \log n)$
Швидкість	Стабільна	Швидший у середньому
Пам'ять	Більше використовує	Менше використовує
Стабільність	Так	Ні

- Merge Sort — якщо важлива стабільність (однакові елементи не міняють порядок).
- Quick Sort — якщо треба швидко відсортувати дані у пам'яті.

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

Скільки даних?

Чи треба стабільність?

Чи обмежена пам'ять?

Чи дані вже трохи відсортовані?

Чи критичний найгірший випадок?

Наприклад:

- Якщо даних небагато — можна брати простий алгоритм.
- Якщо даних багато — краще брати Merge Sort або Quick Sort.