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 методи обчислень»

Тема «Алгоритми сортування та їх складність. Порівняння алгоритмів сортування»

Студент гр. КІ-24-1, Варакута О.О
Викладач, Сидоренко В. М

Кременчук 2025

Тема. Алгоритми сортування та їх складність. Порівняння алгоритмів сортування

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

Короткі теоретичні відомості

Постановка задачі сортування

Сортування є одним із фундаментальних завдань у області обробки даних та алгоритмів. Його мета полягає в упорядкуванні набору елементів у відповідності до певного критерію. Під критерієм може розумітися зростання або спадання значень, лексикографічне або числове порівняння, а також інші параметри відповідно до специфіки задачі. Розглянемо деякі з основних алгоритмів сортування масивів і технологію оцінки та аналізу їх асимптотичної складності.

Перелік існуючих алгоритмів:

На сьогоднішній день існує велика кількість різних алгоритмів сортування, які можуть бути використані для розв’язання різноманітних завдань. Серед найпоширеніших можна виділити такі:

1. Сортування вибором (Selection Sort). Цей алгоритм вибирає найменший елемент зі списку і переміщує його на початок. Потім він повторює
цей процес для решти списку.

2. Сортування бульбашкою (Bubble Sort). Простий алгоритм, який порівнює сусідні елементи і переміщує найбільший елемент у кінець списку. Повторює цей процес для всіх елементів до тих пір, поки список не буде

відсортований.

3. Сортування вставлянням (Insertion Sort). Алгоритм, який збільшує відсортовану частину списку, додаючи кожен наступний елемент на правильне місце.

4. Сортування злиттям (Merge Sort). Рекурсивний алгоритм, який розбиває список навпіл, сортує кожну половину, а потім об'єднує їх впорядкованим чином.

5. Швидке сортування (Quick Sort). Ефективний алгоритм, який вибирає опорний елемент, розбиває список на дві підсписки, менший і більший за опорний елемент, і потім рекурсивно сортує кожен з них.

Це лише декілька з найбільш відомих алгоритмів сортування. Кожен з них має свої переваги і недоліки, а вибір конкретного алгоритму залежить від вхідних даних, обсягу списку, а також вимог до швидкодії та пам’яті.

Алгоритм сортування вибором:

Розглянемо найпростіший алгоритм сортування вибором, оцінимо його складність у найгіршому та найкращому випадку, та асимптотичну складність окремих його процедур.

Алгоритм сортування вибором полягає в наступному:

Для кожного елемента починаючи з першого до передостаннього:

1) Знайти індекс найменшого елемента в залишку списку.

2) Поміняти поточний елемент з найменшим елементом.

Хід роботи

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?

Алгоритм бульбашкового сортування

Алгоритм сортування бульбашкою реалізація за допомогою мови праграмування Python:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

Алгоритм послідовно порівнює сусідні елементи масиву і міняє їх місцями, якщо вони розташовані в неправильному порядку.

Візьмемо масив чисел «5 1 4 2 8», і за допомогою даного алгоритму, відсортуємо його від найменшого до найбільшого елементу. На кожному кроці, елементи, виділені жирним шрифтом будуть порівнюватись.

Перший прохід:
(**5 1** 4 2 8) (**1 5** 4 2 8) Тут, алгоритм порівнює перші два елементи, і міняє їх місцями.
(1 **5 4** 2 8) (1 **4 5** 2 8)
(1 4 **5 2** 8) (1 4 **2 5** 8)
(1 4 2 **5 8**) (1 4 2 **5 8**)

Тут порівнювані елементи знаходяться на своїх місцях, тож алгоритм не міняє їх місцями.

Після першого проходу масиву найбільший елемент гарантовано опинився на останній позиції, тому надалі можна виключити його з роботи.

Другий прохід:

(**1 4** 2 5 8) (**1 4** 2 5 8)

(1 **4 2** 5 8) (1 **2 4** 5 8)

(1 2 **4 5** 8) (1 2 **4 5** 8) Після другого проходу два найбільших елементи масиву гарантовано опинилися на своїх позиціях, тому надалі можна виключити їх з роботи.

Тепер наш масив повністю відсортований, однак, алгоритм цього ще не знає. Йому потрібен ще один «пустий» прохід, під час якого він не поміняє місцями жодного елементу.

Третій прохід:

(**1 2** 4 5 8) (**1 2** 4 5 8)

(1 **2 4** 5 8) (1 **2 4** 5 8)

Нарешті, масив відсортовано, і алгоритм може припинити свою роботу.

Математичний аналіз:

Для масиву розміром n, загальна кількість порівнянь:

$$C(n)=\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}$$

Наприклад для n=5 : $\frac{5\times 4}{2}=10$

Кількість обмінів:

Найгірший випадок : $S_{max}(n)=\frac{n(n-1)}{2}$

Найкращий випадок (масив відсортовано): $S_{min}(n)=0$

Середній випадок: $S(n)=\frac{n(n-1)}{4}$

Асимптотична складність:

Найкращий випадок: O(n) - масив вже відсортований

Найгірший випадок: $O(n^2)$ - масив відсортований у зворотному порядку

Середній випадок: $O(n^2)$

Порівняння з сортуванням вставками:

Беремо за приклад середній випадок для обох алгоритмів:

Бульбашкове сортування :

Порівннянь: $\frac{n(n-1)}{2}$

Обмінів :$\frac{n(n-1)}{4}$

Алгоритм вставки : Порівнянь- $\frac{n(n-1)}{4}$; Обмінів: $\frac{n(n-1)}{4}$

Приклад n=1000:

Бульбашкове сортування : C= $\frac{1000 \times 999}{2} = 499500\, порівнянь$

Сортування вставками : $C = \frac{1000 \times 999}{4}$ = 249750 порівнянь

Чому бульбашкове сортування менш ефективне ніж злиттям?

Асимптотична складність**:** у бульбашки -$O(n^2)$, а у алгоритму злиття - Θ (n log n), що

Кількість операцій: При n=1000: ~500,000 операцій vs ~10,000 операцій

Ефективність: Навіть оптимізована версія бульбашкового сортування не може перевершити $O(n^2)$

Завдання 2.Оцінити асимптотичну складність алгоритму сортування зливанням, скориставшись основною теоремою рекурсії.

Сортування злиттям -алгоритм сортування, в основі якого лежить принцип «Розділяй та володарюй». В основі цього способу сортування лежить злиття двох упорядкованих ділянок масиву в одну впорядковану ділянку іншого масиву.

Математичний аналіз сортування злиттям за основною теоремою рекурсії:

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

Параметри основної теореми: a = 2 - кількість підзадач; b = 2 -коефіцієнт зменшення розміру підзадачі; $f(n)=\Theta(n)$, $f(n)=\Theta(n^d)$ з $d=1$;

Обчислення критичного показника: $\log_b a=\log_2 2=1$.

Оскільки $d=\log_b a=1$, то це випадок 2 основної теореми рекурсії.

У випадку 2: якщо $f(n)=\Theta\left(n^{\log_b a}\right)$, то $T(n)=\Theta\left(n^{\log_b a}\log n\right)=\Theta(n\log n)$.

Точна асимптотична складність: $(T(n)=\Theta(n\log n)$.

Аналіз дерева рекурсії:

Рівень 0: $T(n)=2T\left(\frac{n}{2}\right)+cn$

Рівень 1: $2\cdot T\left(\frac{n}{2}\right)=2\cdot\left[2T\left(\frac{n}{4}\right)+c\cdot\frac{n}{2}\right]=4T\left(\frac{n}{4}\right)+cn$

Рівень 2: $4\cdot T\left(\frac{n}{4}\right)=4\cdot\left[2T\left(\frac{n}{8}\right)+c\cdot\frac{n}{4}\right]=8T\left(\frac{n}{8}\right)+cn$

на кожному рівні загальна вартість = cn

Глибина рекурсії: $k = \log_2 n$ (оскільки розмір підзадачі зменшується удвічі на кожному кроці)

Загальна вартість: $T(n)=cn\cdot\log_2 n=\Theta(n\log n)$

приклад для n =1024: $\log_2 1024=10$

$T(n) = 1024 \cdot 10 = 10240$

Реалізація мовою Python алгоритму сортування зливанням:

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])

    return merge(left, right)

def merge(left, right):
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])
    return result
```

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

Алгоритм сортування записаний мовою програмування Python:

```
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)

Аналіз складності:

Найкращий випадок (ідеальне розбиття):$T(n)=2T\left(\frac{n}{2}\right)+cn=O(n\log n)$

Найгірший випадок : $T(n)=T(n-1)+cn=O(n^2)$

Середній випадок: $E[T(n)]=O(n\log n)$

$Pr[x_{i_j}=1]=\frac{2}{j-i+1}$

$E[T(n)]=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1}=O(n\log n)$

n = 1024 = $1.39\times 1024\times 10\approx 14234$

Експериментальне порівняння 3 алгоритмів:

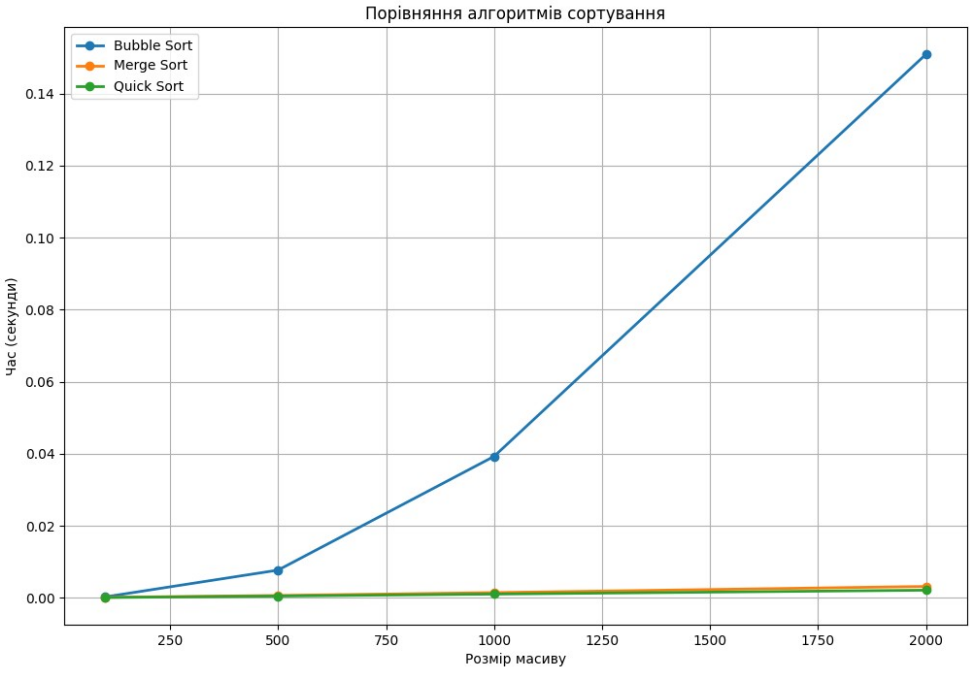
Код python який реалізує порівняння:

```
import random
import time
import matplotlib.pyplot as plt

def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
```

```
    right = merge_sort(arr[mid:])
    return merge(left, right)

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result

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)

def measure_time(sort_func, arr):
    start = time.time()
    sort_func(arr.copy())
    end = time.time()
    return end - start

# Порівняльний аналіз
sizes = [100, 500, 1000, 2000]
algorithms = {
    'Bubble Sort': bubble_sort,
    'Merge Sort': merge_sort,
    'Quick Sort': quick_sort
}

results = {name: [] for name in algorithms}

print("Час виконання алгоритмів сортування (секунди):")
print(f"{'Розмір':<8} {'Bubble':<10} {'Merge':<10} {'Quick':<10}")
print("-" * 45)

for size in sizes:
    arr = [random.randint(1, 1000) for _ in range(size)]

    times = {}
    for name, func in algorithms.items():
        time_taken = measure_time(func, arr)
        results[name].append(time_taken)
        times[name] = time_taken

    print(f"{size:<8} {times['Bubble Sort']:<10.4f} {times['Merge
Sort']:<10.4f} {times['Quick Sort']:<10.4f}")

# Побудова графіка
plt.figure(figsize=(12, 8))
for name, times in results.items():
    plt.plot(sizes, times, marker='o', label=name, linewidth=2)

plt.xlabel('Розмір масиву')
```

```
plt.ylabel('Час (секунди)')
plt.title('Порівняння алгоритмів сортування')
plt.legend()
plt.grid(True)
plt.show()
```

Результати Порівняння :



Час виконання алгоритмів сортування (секунди):

Розмір	Bubble	Merge	Quick
100	0.0003	0.0002	0.0001
500	0.0077	0.0007	0.0005
1000	0.0393	0.0015	0.0011
2000	0.1510	0.0032	0.0022

Порівняльна таблиця алгоритмів:

Параметр	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort
Найкращий	O(n)	$O(n^2)$	O(n)	O(n log n)	O(n log n)
Середній	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(n log n)	O(n log n)
Найгірший	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(n log n)	$O(n^2)$
Пам’ять	O(1)	O(1)	O(1)	O(n)	O(log n)

Стабільність	Так	Ні	Так	Так	Ні

Висновки

Бульбашкове сортування найпростіше, але практично непридатне для великих даних через $O(n^2)$

Сортування злиттям гарантує стабільну O(n log n) продуктивність у всіх випадках

Швидке сортування найшвидше на практиці, але вимагає обережного вибору опорного елемента

Математичний аналіз підтверджує теоретичні оцінки та допомагає вибирати оптимальний алгоритм для конкретних потреб

Експериментальні результати корелюють з теоретичними прогнозами, демонструючи перевагу O(n log n) алгоритмів на великих наборах даних

Відповіді на контрольні питання

1. **Що таке асимптотична складність алгоритму сортування?**

Це міра того, як зростає час виконання алгоритму зі збільшенням розміру вхідних даних.

2. **Які алгоритми мають квадратичну складність?**

Bubble Sort, Selection Sort, Insertion Sort мають $O(n^2)$ у найгіршому випадку.

3. **Чому Merge Sort ефективніший за Insertion Sort?**

O(n log n) vs $O(n^2)$ - експоненційна різниця на великих даних.

4. **Які алгоритми використовуються у стандартних бібліотеках?**

Python: TimSort (гібрид Merge Sort та Insertion Sort)

C++: IntroSort (гібрид Quick Sort, Heap Sort та Insertion Sort)

5. **Різниця між Merge Sort та Quick Sort?**

Merge Sort: гарантовано O(n log n), але O(n) пам'яті

Quick Sort: в середньому швидший, але $O(n^2)$ у найгіршому випадку

6. **Фактори вибору алгоритму сортування:**

Розмір даних, важливість стабільності, обмеження пам'яті, характер даних