9Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики

Кафедра програмування

Лабораторна робота №1

АЛГОРИТМИ СОРТУВАННЯ

з курсу “Алгоритми та структури даних”

Виконав:
студент групи ПМА-11
Ковтун Віталій Олегович

Львів 2024

Сортування злиттям (Merge Sort)

Складність роботи алгоритму: O(n log n)

Просторова складність: O(n)

Стабільність алгоритму: стабільний

Алгоритм (один цикл рекурсії):

1. Розділюємо вхідний список навпіл, отримуємо два підсписки.
2. Рекурсивно викликаємо алгоритм для обох підсписків.
3. Об’єднуємо відсортовані підсписки в один відсортований список:

    3.1. порівнюємо перший елемент обох підсписків;

    3.2. менший елемент додається до кінця нового списку, індекс відповідного списку збільшується;

    3.3. продовжуємо, порівнюючи та додаючи елементи до нового списку.

Розмір допоміжного масиву: такий же, як і розмір вхідного масиву.

Приклад №1

Дано: array1 = {6, 4, 8, 1, 9}, SIZE_1 = 5.

```
Unsorted array 1:
6 4 8 1 9

Left array: 6
Right array: 4
Merged: 4 6

Left array: 4 6
Right array: 8
Merged: 4 6 8

Left array: 1
Right array: 9
Merged: 1 9

Left array: 4 6 8
Right array: 1 9
Merged: 1 4 6 8 9

Sorted array 1:
1 4 6 8 9
```

Приклад №2

Дано: $\text{array1} = \{0, 1, 2, 4, 3\}, \text{SIZE\_1} = 5$.

```
Unsorted array 1:
0 1 2 4 3

Left array: 0
Right array: 1
Merged: 0 1

Left array: 0 1
Right array: 2
Merged: 0 1 2

Left array: 4
Right array: 3
Merged: 3 4

Left array: 0 1 2
Right array: 3 4
Merged: 0 1 2 3 4

Sorted array 1:
0 1 2 3 4
```

У цьому прикладі подано частково відсортований масив, який сортується злиттям з такою ж кількістю операцій, як і хаотично заповнений масив.

Висновок: сортування злиттям є надійним та ефективним алгоритмом, але великі набори даних потребують багато пам’яті. Також даний алгоритм не адаптивний (не використовує вже встановлений порядок елементів), що робить швидкість сортування частково відсортованих масивів такою ж, як швидкість сортування хаотичних масивів.

Швидке сортування (Quick Sort)

Складність роботи алгоритму:

В середньому – O(n log n);

В найгіршому випадку – O(n^2);

Просторова складність: O(log n)

Стабільність алгоритму: нестабільний

Алгоритм:

1. Вибираємо опорний елемент. Зазвичай це перший, останній, чи середній елемент списку.
2. Розподіляємо елементів так, що всі елементи, які менші за опорний, переходять у лівий бік списку.
3. Для лівого і правого підсписків, що утворилися, алгоритм рекурсивно викликає сам себе.
4. Рекурсія викликається до тих пір, поки не залишиться по одному елементу в кожному підсписку.

Алгоритм продовжується, доки всі підсписки не стануть одноелементними.

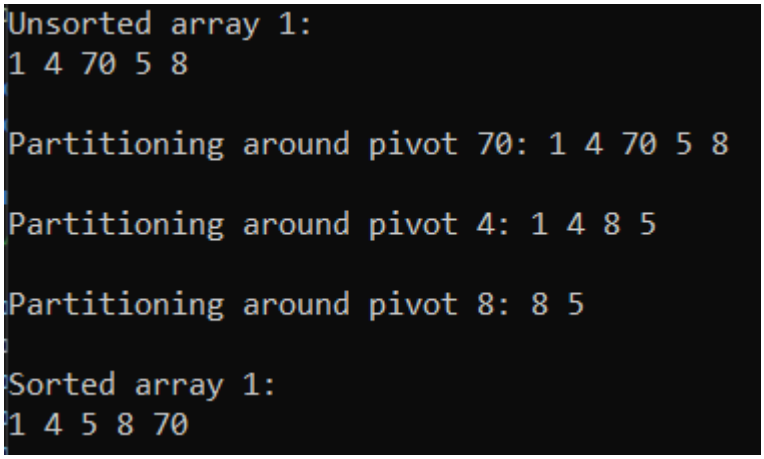
Приклад №1

Дано: array1 = {9, 0, 3, 4, 9}; SIZE_1 = 5.

```
Unsorted array 1:
9 0 3 4 9

Partitioning around pivot 3: 9 0 3 4 9

Partitioning around pivot 3: 3 0

Partitioning around pivot 4: 9 4 9

Partitioning around pivot 9: 9 9

Sorted array 1:
0 3 4 9 9
```

Приклад №2

Дано: array1 = {1, 4, 70, 5, 8}, SIZE_1 = 5.



У даному випадку опорним елементом виявився найбільший елемент масиву. В такому випадку отримуємо найбільшу складність роботи алгоритму O(n^2).

Висновок: швидке сортування є ефективним з точки зору пам’яті та гнучким алгоритмом, який можна компактно реалізувати. В середньому випадку, коли опорним елементом не є найбільший чи найменший елемент масиву, це один з найшвидчих алгоритмів для сортування великих обсягів даних.