Міністерство 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 ПМІ-16

Бевз Маркіян Юрійович

Львів - 2024

Сортування Бульбашкою

Оцінка часової складності:

1. **Найкращий випадок:**
    - Якщо масив вже відсортований, то алгоритм виконає один прохід і перевірить, що немає обмінів.
    - Кількість порівнянь: O(n)
    - Кількість обмінів: 0
    - Оцінка: O(n)
2. **Середній випадок:**
    - Загальна кількість ітерацій: O(n^2)
    - За кожну ітерацію внутрішнього циклу (для порівнянь та обмінів) потрібно пройти через усі n елементів.
    - Оцінка: O(n^2)
3. **Найгірший випадок:**
    - Так само, як і в середньому випадку, загальна кількість ітерацій: O(n^2)
    - Оцінка: O(n^2)

Оцінка просторової складності:

- Алгоритм не використовує додаткову пам'ять, окрім константної кількості для збереження змінних. Таким чином, просторова складність є O(1).

Стабільність:

- Є стабільним алгоритмом, оскільки порівняння та обміни виконуються тільки між сусідніми елементами, які є рівними.

Покроковий аналіз алгоритму:

1. **Введення:**
    - Масив елементів для сортування.
    - Розмір масиву.
2. **Ініціалізація:**
    - Створення змінної типу bool(прапорець) і встановлення її значення на false.
3. **Цикл сортування:**
    - Виконувати цикл сортування, доки прапорець залишається false.

- У кожному проході встановлюємо прапорець на true (вважаємо, що масив вже відсортований).

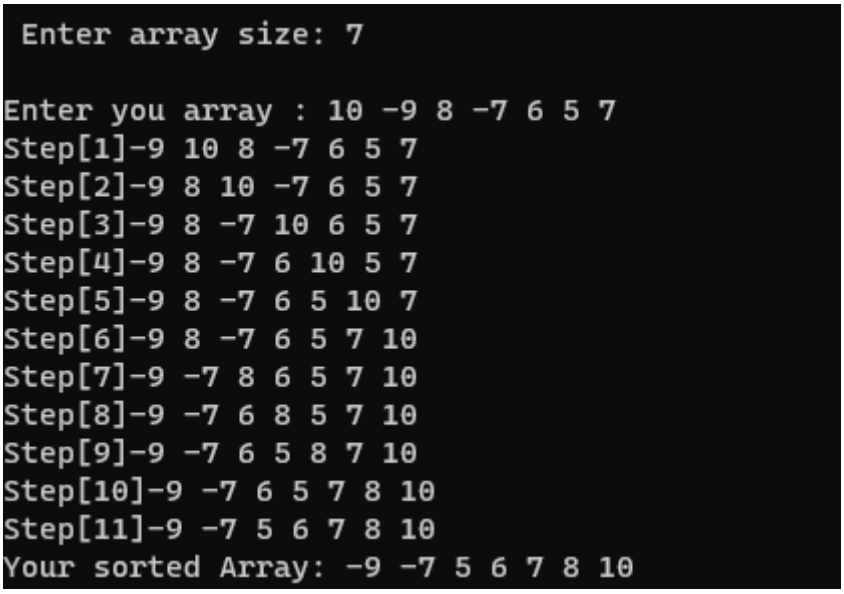
4. **Внутрішній цикл для порівнянь та обмінів:**
    - Проходження через усі елементи масиву за допомогою внутрішнього циклу.
    - Для кожної пари сусідніх елементів порівнюємо їх значення.
    - Якщо лівий елемент більший за правий, викликаємо функцію swap для їх обміну і встановлюємо прапорець на false.
5. **Перевірка відсортованості:**
    - Після завершення внутрішнього циклу перевіряємо, чи був хоча б один обмін під час поточного проходу.
    - Якщо не було обмінів, виходимо із зовнішнього циклу (масив відсортований).
6. **Повторення процесу:**
    - Повторюємо процес, поки весь масив не буде відсортований (тобто поки внутрішній цикл не пройде без жодного обміну).

Приклад № 1
Дано: Arr = {15, 24, 14, -4, 3}, size = 5.

```
 Enter array size: 5

Enter you array : 15 24 14 -4 3
Step[1]15 14 24 -4 3
Step[2]15 14 -4 24 3
Step[3]15 14 -4 3 24
Step[4]14 15 -4 3 24
Step[5]14 -4 15 3 24
Step[6]14 -4 3 15 24
Step[7]-4 14 3 15 24
Step[8]-4 3 14 15 24
Your sorted Array: -4 3 14 15 24
```

Приклад № 2
Дано: Arr = {10, -9, 8, -7, 6, 5, 7}, size = 7**.**



Висновок: Сортування бульбашкою - найпростіший алгоритм сортування, який працює шляхом порівнянь та обмінів сусідніх елементів. Він є легко зрозумілим, проте не ефективним для великих наборів даних через квадратичну часову складність. Алгоритм стабільний, що робить його підходящим для ситуацій, де важливий порядок елементів з однаковими значеннями. Хоча він залишається не оптимальним для багатьох випадків, вивчення його допомагає в розумінні базових принципів сортування та алгоритмів взагалі.

Сортування Злиттям

Оцінка часової складності:

Кожний рекурсивний виклик розділення масиву забезпечує O(log n) шарів, кожен шар вимагає O(n) операцій злиття. Таким чином, загальна часова складність є O(n log n).

Просторова складність:

- O(n) (використовується додатковий масив для збереження даних під час злиття).

Стабільність:

- Алгоритм є стабільним. Порядок елементів з однаковими значеннями не змінюється.

Покроковий аналіз алгоритму:

1. **Розділити масив на дві половини:**
    - Знайти середину масиву.
    - Розділити масив на дві половини: від початку до середини і від середини до кінця.

2. **Сортувати кожну половину рекурсивно:**
    - Застосувати сортування злиттям до першої половини.
    - Застосувати сортування злиттям до другої половини.
3. **Злиття :**
    - Порівняти елементи з двох відсортованих підмасивів і об'єднати їх в новий відсортований масив.
    - Повторювати цей процес, доки всі елементи не будуть об'єднані в новий відсортований масив.

Приклад № 1

Дано: Arr = {100, -9, 8, -7, 6, 5, 7, 12, 205, 193}, size = 10**.**

```
 Enter array size: 10

Enter you array : 100 -9 8 -7 6 5 7 12 205 193
Step[1]-9 8 100
Step[2]-9 -7 6 8 100
Step[3]-9 -7 6 8 100 5 7 12
Step[4]-9 -7 6 8 100 5 7 12 193 205
Step[5]-9 -7 5 6 7 8 12 100 193 205
Your sorted Array: -9 -7 5 6 7 8 12 100 193 205
```

Приклад № 2

Дано: Arr = {102, -933, 84, -73, 63, 54, 72, 34, 21, 23,41,1,2,3,4}, size = 15**.**

```
 Enter array size: 15

Enter you array : 102 -933 84 -73 63 54 72 34 21 23 41 1 2 3 4
Step[1]-933 -73 84 102
Step[2]-933 -73 84 102 34 54 63 72
Step[3]-933 -73 34 54 63 72 84 102
Step[4]-933 -73 34 54 63 72 84 102 1 21 23 41
Step[5]-933 -73 34 54 63 72 84 102 1 21 23 41 2 3 4
Step[6]-933 -73 34 54 63 72 84 102 1 2 3 4 21 23 41
Step[7]-933 -73 1 2 3 4 21 23 34 41 54 63 72 84 102
Your sorted Array: -933 -73 1 2 3 4 21 23 34 41 54 63 72 84 102
```

Висновок: Алгоритм сортування злиттям - ефективний та стабільний алгоритм, який вирішує завдання сортування шляхом розділення масиву на менші частини, сортує їх окремо, та об'єднує у спільний масив. Завдяки часовій складності O(n log n), він підходить для великої кількості даних. Основна ідея алгоритму полягає в рекурсивному розбитті та об'єднанні, що робить його ефективним та широко використовується на практиці.