МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРИЧНОЇ ІНЖЕНЕРІЇ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА АВТОМАТИЗАЦІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

НАВЧАЛЬНА ДИСЦИПЛІНА
«**АЛГОРИТМИ І СТРУКТУРИ ДАНИХ**»

ЗВІТ
З ПРАКТИЧНОЇ РОБОТИ №3

Виконав:
студент групи КН-24-1
Соломка Б. О.

Перевірив:
доцент кафедри АІС
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

Хід роботи

1. Бульбашкове сортування

Алгоритм бульбашкового сортування:

Порівнюємо сусідні елементи масиву.

Якщо елементи не впорядковані, міняємо їх місцями.

Проходимо по масиву n-1 разів, де n - кількість елементів.

З кожним проходом найбільший елемент "спливає" у кінець масиву.

Реалізація алгоритму бульбашкового сортування:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        # Оптимізація: якщо за прохід не було обмінів, масив відсортований
        swapped = False

        # Після i-го проходу i найбільших елементів вже знаходяться на своїх місцях
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True

        # Якщо за прохід не відбулося обмінів, масив відсортований
        if not swapped:
            break

    return arr
```

Асимптотична складність бульбашкового сортування:

У найгіршому випадку (масив відсортований у зворотному порядку): $O(n^2)$

У найкращому випадку (масив вже відсортований): O(n) з оптимізацією

Порівняння з сортуванням вставками:

Сортування вставками має ту ж саму асимптотику: $O(n^2)$ у найгіршому випадку і $O(n)$ у найкращому.

Однак на практиці сортування вставками працює швидше, оскільки виконує менше обмінів елементів.

Порівняння з сортуванням злиттям:

Бульбашкове сортування менш ефективне, ніж сортування злиттям, з наступних причин:

Сортування злиттям має асимптотичну складність $O(n \log n)$ у найгіршому та середньому випадках, що краще, ніж $O(n^2)$ для бульбашкового сортування.

Бульбашкове сортування виконує багато надлишкових операцій порівняння та обміну.

Бульбашкове сортування має низьку кеш-локальність, що уповільнює його на сучасних комп'ютерах.

2. Сортування злиттям

Алгоритм сортування злиттям:

Розділяємо масив на дві рівні частини.

Рекурсивно сортуємо кожну частину.

Зливаємо дві відсортовані частини в один відсортований масив.

Реалізація алгоритму сортування злиттям:

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

    # Розділення масиву на дві частини
    mid = len(arr) // 2
    left = arr[:mid]
    right = arr[mid:]

    # Рекурсивне сортування підмасивів
    left = merge_sort(left)
    right = merge_sort(right)
```

```
    # Злиття відсортованих підмасивів
    return merge(left, right)

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

    # Порівнюємо елементи з обох масивів і додаємо менший до результату
    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

    # Додаємо залишки елементів
    result.extend(left[i:])
    result.extend(right[j:])

    return result
```

Асимптотична складність сортування злиттям (за основною теоремою рекурсії):

Час: $T(n) = 2T(n/2) + O(n)$

a = 2 (2 рекурсивних виклики)

b = 2 (розмір підзадачі зменшується вдвічі)

$f(n) = O(n)$ (час на злиття)

Оскільки $a = b^1$, за основною теоремою рекурсії: $T(n) = O(n \log n)$

Просторова складність: $O(n)$

3. Швидке сортування (QuickSort)

Алгоритм швидкого сортування:

Вибираємо опорний елемент (pivot) з масиву.

Розділяємо масив на два підмасиви: елементи менші за опорний і елементи більші або рівні опорному.

Рекурсивно застосовуємо швидке сортування до кожного з підмасивів.

Реалізація алгоритму швидкого сортування:

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

    # Вибір опорного елемента (для простоти - перший елемент)
    pivot = arr[0]

    # Розділення масиву на частини
    less = [x for x in arr[1:] if x <= pivot]
    greater = [x for x in arr[1:] if x > pivot]

    # Рекурсивне сортування і об'єднання результатів
    return quick_sort(less) + [pivot] + quick_sort(greater)
```

Асимптотична складність швидкого сортування (за основною теоремою рекурсії):

У найкращому та середньому випадках:

$T(n) = 2T(n/2) + O(n)$ (якщо pivot розділяє масив приблизно навпіл)

$a = 2, b = 2, f(n) = O(n)$

За основною теоремою рекурсії: $T(n) = O(n \log n)$

У найгіршому випадку (наприклад, коли масив вже відсортований):

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

Це дає асимптотику $O(n^2)$

Висновки:

Під час виконання практичної роботи були зроблені наступні висновки:

Бульбашкове сортування має просту реалізацію, але найгіршу асимптотичну складність $O(n^2)$, що робить його неефективним для великих масивів даних.

Сортування злиттям забезпечує стабільну асимптотичну складність $O(n \log n)$ у всіх випадках, але використовує додаткову пам'ять.

Швидке сортування має асимптотичну складність $O(n \log n)$ у середньому, але може деградувати до $O(n^2)$ у найгіршому випадку. При цьому

воно не потребує додаткової пам'яті (за винятком стеку виклику рекурсії).

Бульбашкове сортування менш ефективне порівняно з сортуванням злиттям через квадратичну складність та велику кількість обмінів елементів.

Для практичного використання рекомендується застосовувати алгоритми з асимптотичною складністю O(n log n) - сортування злиттям або швидке сортування, з урахуванням специфіки задачі та структури вхідних даних.