Практична робота № 3

Ляпко Анастасія КН-23-1

Тема. Алгоритми сортування та їх складність. Порівняння алгоритмів сортування

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

Постановка задачі.

Опис алгоритму бульбашкового сортування:

1. Перебираємо масив з початку до кінця.
2. Порівнюємо кожну пару сусідніх елементів.
3. Якщо поточний елемент більше за наступний, обмінюємо їх місцями.
4. Повторюємо кроки 1-3, доки масив не буде відсортовано.

Псевдокод:

```
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]
```

Оцінка асимптотичної складності:

- **Найгірший випадок (масив відсортований у зворотному порядку)**:
  - Часова складність: $O(n^2)$
  - Кількість порівнянь і обмінів є максимальною.
- **Найкращий випадок (масив вже відсортований)**:
  - Часова складність: $O(n)$ при використанні оптимізації (флаг для перевірки відсортованості).
  - Кількість порівнянь мінімальна, і обмінів немає.

Порівняння з алгоритмом сортування вставлянням:

- **Сортування вставлянням (Insertion Sort)**:
  - Найгірший випадок: $O(n^2)$

    - Найкращий випадок: $O(n)$O(n) (якщо масив вже відсортований)
- У найгіршому випадку обидва алгоритми мають квадратичну складність $O(n2)$O(n2), але на практиці сортування вставлянням зазвичай працює краще через меншу кількість обмінів.

Чому бульбашкове сортування менш ефективне, ніж сортування злиттям?

- **Сортування злиттям (Merge Sort)**:
    - Часова складність: $O(nlogn)$O(nlogn) як у найгіршому, так і в найкращому випадку.
    - Використовує техніку "розділяй і володарюй", розбиваючи масив на підмасиви, сортує їх і об'єднує.

Бульбашкове сортування є менш ефективним, ніж сортування злиттям, через квадратичну часову складність $O(n2)$O(n2) у найгіршому випадку, в той час як сортування злиттям забезпечує $O(nlogn)$O(nlogn) навіть у найгіршому випадку, що набагато швидше для великих масивів.

2. Оцінка асимптотичної складності алгоритму сортування злиттям

Сортування злиттям має рекурсивну структуру. Розділяє масив на дві частини, сортує кожну частину і зливає їх.

Рекурсивне рівняння:

$T(n)=2T(n/2)+O(n)$де T(n) - час виконання для масиву розміру nn, $2T(n/2)$ - два підмасиви розміру $n/2$, $O(n)$ - час для злиття підмасивів.

Основна теорема рекурсії:

Для рівняння виду:

$T(n)=aT(n/b)+f(n)$

де a=2, b=2, f(n)=O(n).

З теореми випливає, що якщо $f(n)=O(n^c)$, де $c<\log ba$, то $T(n)=O(n^(\log ba))$. У нашому випадку:

$\log ba=\log 2=1$

Таким чином, $T(n)=O(n \log n)$.

3. Алгоритм швидкого сортування

Опис алгоритму швидкого сортування:

5. Вибираємо опорний елемент (pivot).
6. Розділяємо масив на дві частини: елементи, менші за опорний, та елементи, більші за опорний.
7. Рекурсивно застосовуємо швидке сортування до кожної частини.

Псевдокод:

```
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)
```

Оцінка асимптотичної складності:

- **Найгірший випадок (опорний елемент вибирається погано, наприклад, найбільший або найменший елемент)**:
  - Часова складність: $O(n$^2)
  - Рекурсія розділяє масив на нерівні частини (одна частина майже порожня).
- **Середній та найкращий випадок (опорний елемент вибирається добре, розділяючи масив на рівні частини)**:
  - Часова складність: $O(n \log n)$
  - Рекурсивне рівняння: $T(n)=2T(n/2)+O(n)$

Основна теорема рекурсії:

Для середнього та найкращого випадку:

$T(n)=2T(n/2)+O(n)$ де $a=2$, b=2, f(n)=O(n). З теореми випливає:

$T(n)=O(n \log n)$

Отже, у середньому та найкращому випадку швидке сортування має асимптотичну складність $O(n\log n)$, що робить його дуже ефективним для практичного використання, хоча у найгіршому випадку воно може досягати $O(n^2)$.

Контрольні питання:

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

Асимптотична складність алгоритму сортування – це міра, яка показує, як змінюється час виконання або обсяг пам'яті, що використовується алгоритмом, у залежності від розміру вхідних даних, коли цей розмір прямує до нескінченності. Вона важлива для порівняння алгоритмів, оскільки дозволяє оцінити їх ефективність незалежно від реалізаційних деталей та апаратних засобів. Асимптотична складність допомагає визначити, як алгоритм поводиться з великими обсягами даних і який з алгоритмів буде ефективнішим у довгостроковій перспективі.

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

До алгоритмів сортування з квадратичною складністю у найгіршому випадку належать:

- **Бульбашкове сортування (Bubble Sort)**: *O(n^2)*
- **Сортування вставками (Insertion Sort)**: *O(n^2)*
- **Сортування вибором (Selection Sort)**: *O(n^2)O*

Ця квадратична складність означає, що час виконання алгоритму пропорційний квадрату кількості елементів. Для великих обсягів даних це призводить до дуже довгого часу виконання, оскільки навіть незначне збільшення розміру вхідних даних спричиняє значне зростання часу виконання.

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

Перевага сортування злиттям над сортуванням вставками полягає в тому, що сортування злиттям має гарантовану асимптотичну складність *O(nlogn)* у всіх випадках (найгірший, середній, найкращий), тоді як сортування вставками має найгірший випадок *O(n2)*. Для великих наборів даних сортування злиттям є значно ефективнішим, оскільки забезпечує набагато менший час виконання, ніж сортування вставками.

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

- **Python**: Використовує алгоритм Timsort, який є комбінацією сортування злиттям та сортування вставками, з асимптотичною складністю $O(n \log n)$ у середньому та найгіршому випадках.
- **Java**: Для масивів використовується модифікований алгоритм Quicksort (Dual-Pivot Quicksort) з середньою складністю $O(n \log n)$, а для списків – Timsort.
- **C++**: Стандартна бібліотека використовує алгоритм IntroSort, який починає з Quicksort і переходить до HeapSort, коли глибина рекурсії стає надто великою. Має асимптотичну складність $O(n \log n)$.

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

- **Сортування злиттям (Merge Sort)**:
  - Часова складність: $O(n \log n)$ у найгіршому, середньому та найкращому випадках.
  - Просторова складність: $O(n)$ додаткової пам'яті.
  - Стабільний: Зберігає порядок однакових елементів.
  - Використовується для сортування великих масивів або списків, де важлива стабільність і гарантія часу виконання.
- **Швидке сортування (Quick Sort)**:
  - Часова складність: $O(n2)$ у найгіршому випадку, $O(n \log n)$ у середньому та найкращому випадках.
  - Просторова складність: $O(\log n)$ додаткової пам'яті.
  - Нестабільний: Не зберігає порядок однакових елементів.
  - Використовується для сортування масивів, де важлива швидкість у середньому випадку і немає необхідності в стабільності.

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

При виборі алгоритму сортування слід враховувати:

- **Розмір вхідних даних**: Для великих обсягів даних кращі алгоритми з асимптотичною складністю $O(n \log n)$.
- **Стабільність**: Якщо потрібно зберігати порядок однакових елементів, слід вибирати стабільні алгоритми.
- **Просторова складність**: Обмеження пам'яті можуть впливати на вибір алгоритму.
- **Складність реалізації**: Деякі алгоритми легше реалізувати і налагодити.

- **Характеристики даних**: Деякі алгоритми працюють краще на вже частково відсортованих або невпорядкованих даних.
- **Час виконання в середньому випадку**: Вибір алгоритму може залежати від того, як часто трапляються найгірші або найкращі випадки у практиці.