**Практична робота № 3**

**Тема. Алгоритми сортування та їх складність. Порівняння алгоритмів сортування**

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

**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?

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

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

**Хід роботи**

**1.** Вивчаємо і записуємо (будь-яким способом) алгоритм бульбашкового сортування.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3.1 – Вигляд алгоритму бульбашкового сортування

**1.2** Оцінюємо асимптотику алгоритму сортування методом бульбашки в найгіршому і в найкращому випадку.

Найгірший випадок. У найгіршому випадку масив відсортований у зворотному порядку, тобто кожен елемент необхідно пересунути на своє місце. У цьому випадку алгоритм повинен виконати всі ітерації циклів.

Кожна ітерація зовнішнього циклу (for i in range(0, len(array))) виконується приблизно n разів.

На кожній ітерації внутрішній цикл (for j in range(0, len(array) - i - 1)) проходить по всіх елементах, що залишилися, тобто приблизно n - i разів.

Сумарна кількість порівнянь у цьому випадку буде приблизно:

Отже, асимптотична складність у найгіршому випадку: O(n2).

Найкращий випадок. У найкращому випадку масив уже відсортований за зростанням, і жодних перестановок не потрібно. У деяких реалізаціях алгоритму передбачена оптимізація, яка припиняє роботу, якщо за одну ітерацію жодної перестановки не відбулося.

У такому випадку зовнішній цикл виконується лише один раз, а внутрішній проходить по n - 1 елементам.

Отже, асимптотична складність у найкращому випадку: O(n).

**1.3** Порівнюємо за цими показниками бульбашковий алгоритм з алгоритмом сортування вставлянням.

Найгірший випадок. Обидва алгоритми мають однакову асимптотику O(n2). Але на практиці сортування вставками часто працює швидше, оскільки воно потребує менше пересувань елементів, ніж бульбашкове сортування.

Найкращий випадок. Обидва алгоритми мають лінійну складність O(n), але сортування вставками зазвичай виконується швидше, особливо для майже відсортованих масивів.

**1.4** Чому на практиці бульбашковий алгоритм виявляється менш ефективним у порівнянні з сортуванням методом зливанням?

Так як алгоритм зливання має асимптотичну складність nlog(n) (рисунок 3.2), є величезна різниця у кількості операцій для великих n.

Изображение выглядит как текст, снимок экрана, График, линия

Автоматически созданное описание

Рисунок 3.2 – Графік порівняння алгоритмів

**2.** Оцінюємо асимптотичну складність алгоритму сортування зливанням, користуючись основною теоремою рекурсії.

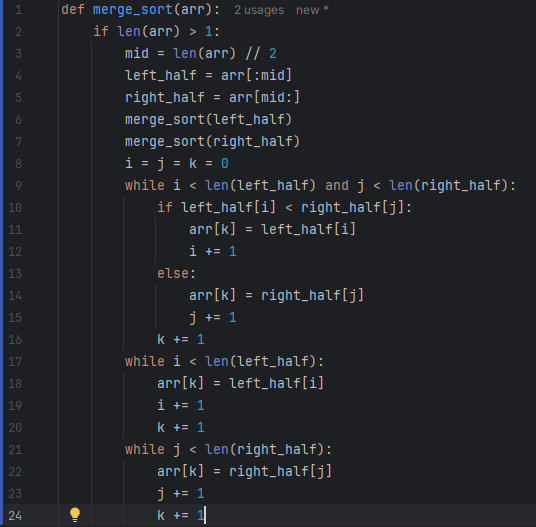


Рисунок 3.3 – Вигляд алгоритму сортування зливанням

Оцінка складності. Рекурсивна частина. На кожному рівні рекурсії масив поділяється навпіл, отже, кожна рекурсивна підзадача має розмір ​. Алгоритм продовжує розділяти масив, поки розмір підмасиву не стане 1.

Кількість рівнів рекурсії буде log2​(n), оскільки на кожному рівні ми ділимо масив на дві частини.

Операція злиття. Під час кожного злиття двох відсортованих підмасивів ми виконуємо O(n) операцій, оскільки треба пройти через всі елементи обох частин масиву і розмістити їх у правильному порядку. Рекурсивне рівняння. Рекурсивна частина алгоритму має вигляд:

де: a = 2 — ми розділяємо масив на дві частини;

b = 2 — кожна підзадача має розмір ​;

O(n) — час на злиття двох відсортованих підмасивів.

Використання основної теореми рекурсії. Тепер можна застосувати основну теорему рекурсії для оцінки складності. У рівнянні:

Згідно з основною теоремою рекурсії, для рівнянь такого вигляду, коли  
a = bd (у нашому випадку a = 2, b = 2 і d = 1), складність буде:

Висновок

Асимптотична складність алгоритму сортування зливанням у найгіршому, найкращому та середньому випадках є однаковою і дорівнює:

Це значно ефективніше, ніж алгоритми з складністю O(n2), як бульбашкове сортування або сортування вставками, особливо для великих масивів.

**3.** Вивчаємо і записуємо (будь-яким способом) самостійно алгоритм швидкого сортування.

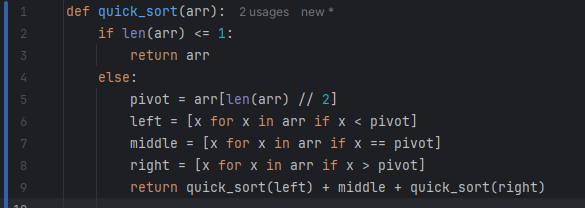


Рисунок 3.4 – Вигляд алгоритму швидкого сортування

**3.1** Оцінюємо асимптотичну складність алгоритму швидкого сортування, скориставшись основною теоремою рекурсії.

Рекурсивне рівняння. Асимптотична складність швидкого сортування залежить від розподілу елементів відносно опорного. Рекурсивне рівняння має вигляд:

де: a = 2 — кількість рекурсивних підзадач на кожному рівні;

b = 2 — кожна підзадача має розмір приблизно ​, якщо опорний елемент обраний вдало;

O(nd) = O(n) — це час, необхідний для розбиття масиву відносно опорного елемента.

Отже, рівняння можна записати як:

Застосовуємо основну теорему рекурсії для рівняння

У нашому випадку:

- a = 2;

- b = 2;

- nd = n, отже, d = 1

Обчислюємо bd:

Порівняємо значення a i bd:

Якщо a < bd, то ;

Якщо a = bd, то ;

Якщо a > bd, то .

Оскільки a = bd, застосовується другий випадок, і складність буде:

**Контрольні питання**

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

Асимптотична складність алгоритму сортування - це математична характеристика, яка описує, як змінюється час виконання алгоритму або споживання пам’яті залежно від розміру вхідних даних (наприклад, кількості елементів, що потрібно відсортувати). Ця складність зазвичай виражається за допомогою *O-нотації* (наприклад, O(n2), *O*(*n*log*n*)), яка відображає порядок зростання ресурсоємності алгоритму.

Важливість асимтотичної складності:

- оцінка продуктивності – вона дозволяє порівнювати алгоритми незалежно від конкретної реалізації чи швидкості апаратного забезпечення;

- масштабованість – асимптотична складність показує, наскільки ефективно алгоритм працює з великими наборами даних;

- вибір алгоритму – для різних задач і обмежень можна обрати найбільш ефективний алгоритм.

**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 *O*(*n*2) у найгіршому випадку належать:

- Bubble Sort (сортування бульбашкою). Кожен елемент порівнюється з сусіднім, і якщо вони не у правильному порядку, їх міняють місцями. Найгірший випадок: масив відсортований у зворотному порядку. Для цього потрібні *n*−1 проходів і до *n*(*n*−1)/2 порівнянь;

**-** Selection Sort (сортування вибором).На кожній ітерації знаходиться мінімальний елемент із невідсортованої частини масиву і переміщається на свою позицію.Найгірший випадок: незалежно від структури даних, потрібно виконати приблизно *n*2 порівнянь, бо алгоритм завжди переглядає весь невідсортований масив.

**-** Insertion Sort (сортування вставками)На кожній ітерації елемент вставляється у правильну позицію серед раніше відсортованих елементів.Найгірший випадок: коли масив відсортований у зворотному порядку. Кожен новий елемент доведеться порівнювати з усіма попередніми, що призводить до *O*(*n*2) операцій.

Квадратична складність є проблемою для великих обсягів даних, тому що має:

- швидке зростання витрат часу;

- мала ефективність на практиці;

- проблеми масштабованості.

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

Перевага сортування злиттям (Merge Sort) над сортуванням вставками (Insertion Sort) для великих наборів даних полягає в тому, що сортування злиттям має кращу асимптотичну часову складність і більш передбачувану продуктивність.

Сортування злиттям є кращим вибором для великих наборів даних завдяки його ефективності (*O*(*n*log*n*)) та стабільній продуктивності незалежно від початкового впорядкування даних.

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

Таблиця 3.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Мова | Функція | Алгоритм | Стабільність | Часова складність |
| Python | sorted(), .sort() | Timsort | Так |  |
| Java | Arrays.sort() (прим.) | Dual-Pivot QuickSort | Ні |  |
| Java | Arrays.sort() (об'єкти) | Timsort | Так |  |
| C++ | std::sort() | Introsort (QuickSort + HeapSort) | Ні |  |
| C++ | std::stable\_sort() | Merge Sort | Так |  |

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

Сортування злиттям підходить для задач, де важлива стабільність, необхідно працювати із зовнішньою пам’яттю або уникнути ризику найгіршого часу.

Швидке сортування є кращим вибором у більшості практичних задач, якщо виконуються оптимізації (наприклад, вибір хорошого опорного елемента або перехід до Insertion Sort для малих підмасивів).

Використання сортування злиттям:

- стабільність важлива. Наприклад, коли сортуються записи бази даних із дотриманням первинного порядку елементів;

- дані дуже великі. Алгоритм підходить для зовнішнього сортування, коли масив не може бути повністю завантажений у пам'ять;

- прогнозована продуктивність. Якщо необхідно уникнути погіршення продуктивності у найгіршому випадку.

Використання швидкого сортування:

- висока швидкість у середньому випадку. Підходить для задач, де важлива продуктивність і немає потреби у стабільності;

- обмеження на додаткову пам'ять. Quick Sort використовує значно менше пам’яті, ніж Merge Sort;

- вхідні дані невеликі. Для невеликих наборів даних Quick Sort працює ефективніше через менші накладні витрати.

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

Вибір алгоритму сортування залежить від багатьох факторів, пов’язаних із природою даних, вимогами до продуктивності та обмеженнями середовища виконання. Ось ключові фактори, які слід враховувати:

- розмір даних;

- структура вихідних даних;

- вимога стабільності;

- обмеження на пам’ять;

- час виконання у найгіршому випаду;

- потреба в багатопоточності;

- зовнішнє застосування;

- реалізація та підтримка.

Вибір алгоритму сортування залежить від характеристик даних, середовища виконання та вимог до продуктивності й пам’яті. Для більшості загальних задач рекомендується використовувати оптимізовані реалізації стандартних бібліотек.