МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Кафедра комп’ютерної інженерії та електроніки

ЗВІТ З ПРАКТИЧНИХ РОБІТ

з навчальної дисципліни

«Алгоритми та методи обчислень»

Тема «Алгоритми сортування та їх складність. Порівняння алгоритмів сортування.»

Студент гр. КН-23-1 ПІБ Варич А.І

Викладач к. т. н., доц. ПІБ Сидоренко В.М

Кременчук 2024

ЗМІСТ

[1 Перше завдання. Сортування бульбашкою. 3](#_Toc161752653)

[2 Друге завдання. Сортування злиттям. 5](#_Toc161752654)

[3 Третє завдання. Швидке сортування. 6](#_Toc161752655)

[4 Контрольні питання 7](#_Toc161752656)

# Перше завдання. Сортування бульбашкою.

Постановка задачі: записати 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 (рис. 1 - сортування бульбашкою).

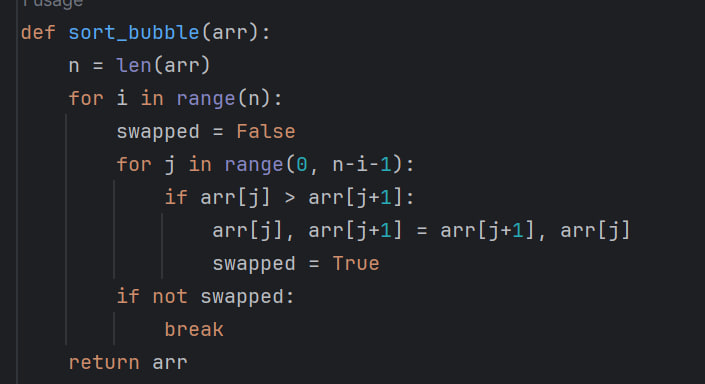


рис. 1 - сортування бульбашкою

У найгіршому випадку, кожен елемент масиву потребує проходження через усі інші елементи, щоб досягти своєї позиції відповідної до порядку сортування. Це призводить до асимптотики у найгіршому випадку, коли масив вже відсортований у зворотньому порядку або майже відсортований.

У найкращому випадку, коли масив вже відсортований, бульбашкове сортування може мати асимптотику O(n), оскільки жодних обмінів не потрібно виконувати.

Порівняно з алгоритмом сортування вставками, бульбашкове сортування зазвичай менш ефективне, особливо на великих масивах або в випадках, коли масив частково відсортований (рис. 2 - сортування вставками).

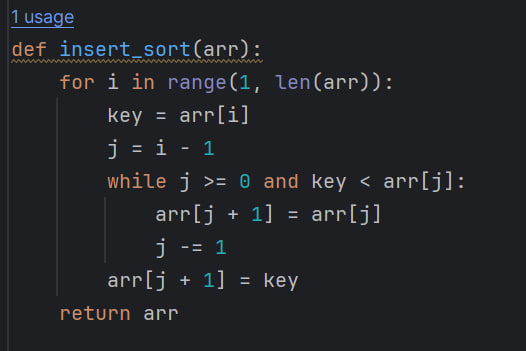


рис. 2 - сортування вставками

Результатом бульбашкового сортування є відсортований масив. Він порівнює кожну пару сусідніх елементів у масиві і, якщо вони не знаходяться в правильному порядку, міняє їх місцями. Цей процес повторюється доти, доки не відбудеться повний прохід без жодних обмінів елементів. Результатом цього є відсортований масив (рис. 3 - результат сортування бульбашкою).

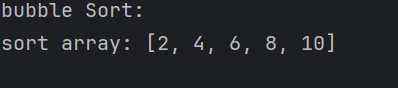


рис. 3 - результат сортування бульбашкою

# Друге завдання. Сортування злиттям.

Постановка задачі: Оцінити асимптотичну складність алгоритму сортування зливанням, скориставшись основною теоремою рекурсії.

Основна теорема рекурсії для алгоритму сортування злиттям стверджує, що час сортування списку розміром буде пропорційним сумі часу сортування двох списків розміром плюс час злиття цих двох списків.

У коді, я писала функції для сортування злиттям, які рекурсивно розбивають масив на половини, сортують кожну половину окремо, а потім зливають їх разом (рис. 4 - сортування злиттям).

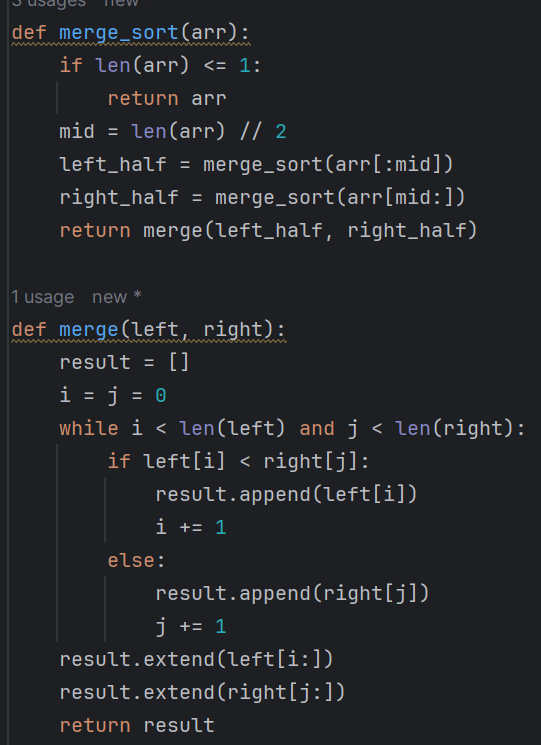


рис. 4 - сортування злиттям

Результатом алгоритму сортування злиттям є відсортований масив. Спочатку розбиває вхідний масив на менші частини, сортує кожну з них рекурсивно, а потім об'єднує відсортовані підмасиви в один великий відсортований масив. У результаті я отримую масив, у якому всі елементи розташовані у відсортованому порядку, що і є бажаним результатом сортування (рис. 5 - результат сортування злиттям).

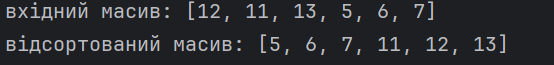


рис. 5 - результат сортування злиттям

# Третє завдання. Швидке сортування.

Постановка задачі: Записати самостійно алгоритм швидкого сортування. Оцінити асимптотичну складність алгоритму швидкого сортування, скориставшись основною теоремою рекурсії.

Асимптотична складність алгоритму швидкого сортування зазвичай є в середньому та в найгіршому випадку. Основна теорема рекурсії підтверджує цю складність, оскільки кожен рекурсивний виклик розбиває масив пополам, а потім об'єднує вже відсортовані підмасиви (рис. 6 - швидке сортування).

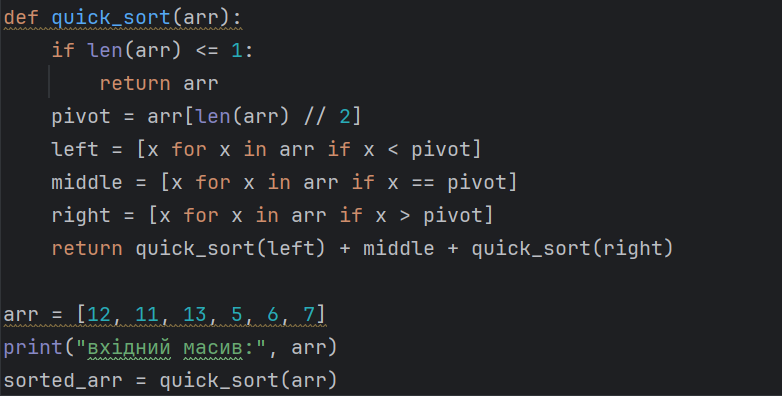


рис. 6 - швидке сортування

Результатом швидкого сортування є відсортований масив. Алгоритм працює наступним чином: спочатку вибирається елемент масиву як опорний . Потім розбивається вхідний масив на два підмасиви: один, що містить елементи менші за опорний, інший - більші або рівні опорному. На кожному рівні рекурсії масив розбивається на дві частини, що дозволяє швидко зближуватися до відсортованого масиву. У кінцевому результаті, коли розмір підмасивів стає 1 або 0, масив вважається відсортованим (рис. 7 - результат швидкого сортування).

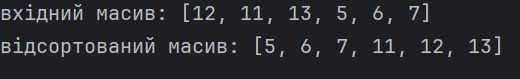


рис. 7 - результат швидкого сортування

# Контрольні питання

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

Асимптотична складність алгоритму сортування - це міра того, як швидко зростає час виконання або кількість операцій алгоритму при збільшенні об'єму вхідних даних. Асимптотична складність є важливою для порівняння алгоритмів сортування, оскільки дозволяє визначити, як кожен алгоритм поводиться на великих наборах даних. Знання асимптотичної складності дозволяє розуміти, як алгоритми поведуться у реальних ситуаціях, коли розмір вхідних даних зростає.

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

Алгоритми сортування з квадратичною складністю у найгіршому випадку: Бульбашкове сортування, Сортування вибором, Сортування вставками. Це може бути проблемою для великих обсягів даних, оскільки час виконання зростає квадратично зі збільшенням розміру вхідних даних, що призводить до повільної роботи алгоритмів на великих обсягах даних.

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

Сортування злиттям має кращу асимптотичну складність у порівнянні з сортуванням вставками для великих наборів даних.

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

Python: Timsort (комбінація сортувань злиттям та вставками), Java: TimSort (також комбінація сортувань злиттям та вставками), C++: std::sort (зазвичай Quick Sort або Introsort, який комбінує Quick Sort та Heap Sort).

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

Сортування злиттям ефективніше на великих наборах даних і гарантує стабільність сортування, тоді як швидке сортування (Quick Sort) швидше на малих наборах даних і зазвичай потребує менше додаткової пам'яті.

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

Розмір вхідних даних, тип даних (числа, рядки тощо), ефективність алгоритму на конкретних типах даних, часові обмеження, доступна пам'ять, стабільність сортування, потреба у додатковій пам'яті, вплив на порядок елементів, можливість наявності повторюваних елементів.