Іванов Артем Олександрович
студент 4 курсу, групи 46Бд-Комп
Житомирського державного університету імені Івана Франка
(063)434-81-33
Art_Iv3@ukr.net
Науковий керівник
Кривонос Олександр Миколайович
доцент кафедри комп'ютерних наук та інформаційних технологій
Житомирського державного університету імені Івана Франка
(098)742-02-28
krypton@zu.edu.ua

ПОРІВНЯННЯ АЛГОРИТМІВ СОРТУВАННЯ

Постановка задачі. Об'єми цифрових даних стрімко зростають, як і час на їх обробку. І покращення алгоритмів їх обробки дозволяє заощаджувати на «залізі», покращуючи ефективність. Одним з, можливо, найважливіших алгоритмів обробки даних є сортування. Цих алгоритмів існує безліч, і всі вони відрізняються своїми структурами даних, найгіршими випадками, використанням пам'яті тощо. Порівняння цих алгоритмів необхідне для розуміння того, який алгоритм краще використовувати в поставленій задачі.

Мета дослідження. Метою дослідження є аналіз і порівняння за часом виконання і використанням пам'яті відомих алгоритмів сортування: bubble sort, merge sort, quick sort, heap sort.

Результати дослідження. Масив даних називається відсортованим, якщо його елементи впорядковані за деякою властивістю. Це може бути не просто сортування чисел, а і сортування цілих структур даних. В залежності від поставленої задачі і обмежень, можуть використовуватись різні алгоритми сортування, наприклад: для виконання сортування на сервері, до якого звертаються тисячі користувачів, пріоритетом є швидкість; а при сортуванні даних на мікропроцесорі з обмеженою оперативною пам'яттю, пріоритетом може бути зменшення використання ресурсів ОЗП. Далі будуть розглянуті і порівняні самі основні алгоритми сортування.

Бульбашкове сортування. Самий примітивний і в той же час простий алгоритм сортування, в основі якого лежить «підняття» максимального елементу в кінець невідсортованої частини масиву. Оскільки всі операції відбуваються безпосередньо на вихідному масиві, використання додаткової пам'яті цим алгоритмом константне $O(n) = 1$. Найкращий випадок у разі бульбашкового сортування є заздалегідь відсортований масив, тому найкращий час буде $\Omega(n) = n$, оскільки алгоритму необхідно мінімум один раз пройтись по масиву. В найгіршому випадку, масив буде відсортований в оберненому порядку, через що кожен елемент буде переміщуватись окремо, і в такому випадку час виконання буде

$$O(n) = (n-1) + (n-2) + \cdots + 1 = \frac{(n-1)(n-2)}{2} = n^2.$$

Сортування злиттям. В основі цього алгоритму стоїть принцип «розділяй та володарюй», коли ми рекурсивно розділяємо масив на дві рівні частини, і розв'язуємо цю саму задачу для них, потім поєднуючи їх за лінійний час. Для обрахунку використання пам'яті цим алгоритмом, помітимо, що під час рекурсії виділяється масив довжиною n для збереження проміжних результатів; і при використанні рекурсивного підходу додаткова пам'ять виділяється на стек викликів, і оскільки при кожному виклику ми зменшуємо розмір кожного підмасиву вдвічі, розмір стеку буде рівним $\log n$. Комбінуючи використання пам'яті, отримуємо $O(n) = \Omega(n) = n + \log n = n$. Для обрахунку використання часу використовується теорема Акра-Баззі про рекурентні співвідношення типу «розділяй і володарюй». В результаті, час виконання алгоритму в найгіршому випадку та найкращому випадках будуть рівними $\Omega(n) = O(n) = n \log n$. Також варто зазначити, що на відміну від наступного алгоритму, дане сортування є стабільним, тобто таким, яке зберігає порядок однакових елементів.

Швидке сортування. Даний алгоритм також відноситься до сімейства «розділяй і володарюй», але використовує дещо інший підхід. На кожному етапі сортування, ми вибираємо деяку опорну точку, і розміщуємо всі інші елементи, які більші з нього, в один бік, а які більші – в інший. В кінці всі підмасиви з'єднуються. Ефективність алгоритму залежить від вибору опорних точок. Хоча в рекурсивному випадку алгоритм не створює додаткові масиви для збереження проміжних даних, в найгіршому сценарії – при незбалансованому виборі опорних точок – буде надмірно заповнюватись стек викликів, що призведе до використання пам'яті $O(n) = n$. В найкращому випадку – при збалансованому виборі опорних точок – масив розділятиметься навпіл, тому використання пам'яті буде $\Omega(n) = \log n$.

При збалансованому виборі опорних точок, ми можемо розрахувати час виконання алгоритму швидкого сортування як для найкращого випадку з використанням теореми Акра-Баззі, що дасть $\Omega(n) = n \log n$. При незбалансованому виборі, ми вже не можемо використати цей алгоритм через відсутність чітких підзадач, на які розділятиметься попередня. В найгіршому випадку ми можемо грубо розрахувати час виконання алгоритму як $O(n) = O(n-1) + cn$, де cn – вартість поділу масиву. Тоді асимптотика для найгіршого випадку дасть нам $O(n) = \frac{n(n-1)}{2} = n^2$.

Сортування кучами. Цей алгоритм є вдосконаленим варіантом сортування вибором, який оригінально має квадратну часову асимптотику. Алгоритм полягає в інтерпретації масиву як повного збалансованого бінарного дерева. Далі воно перетворюється в мінімальну або максимальну кучу в залежності від порядку сортування. В мінімальній кучі значення кожного вузла менше за його дочірніх, і відповідно більше для максимальної кучі. Всі обрахунки відбуваються на вихідному масиві, тому використання пам'яті константне $O(n) = 1$. Алгоритм не має найгіршого сценарію роботи, і завжди виконується за один час. Побудова min/max кучі використовує час $O(n)$ і для відновлення властивостей кучі до

кожного елементу виконується функція за $O(\log n)$. З цього випливає загальна асимптотика алгоритму $O(n) = n \log n$.

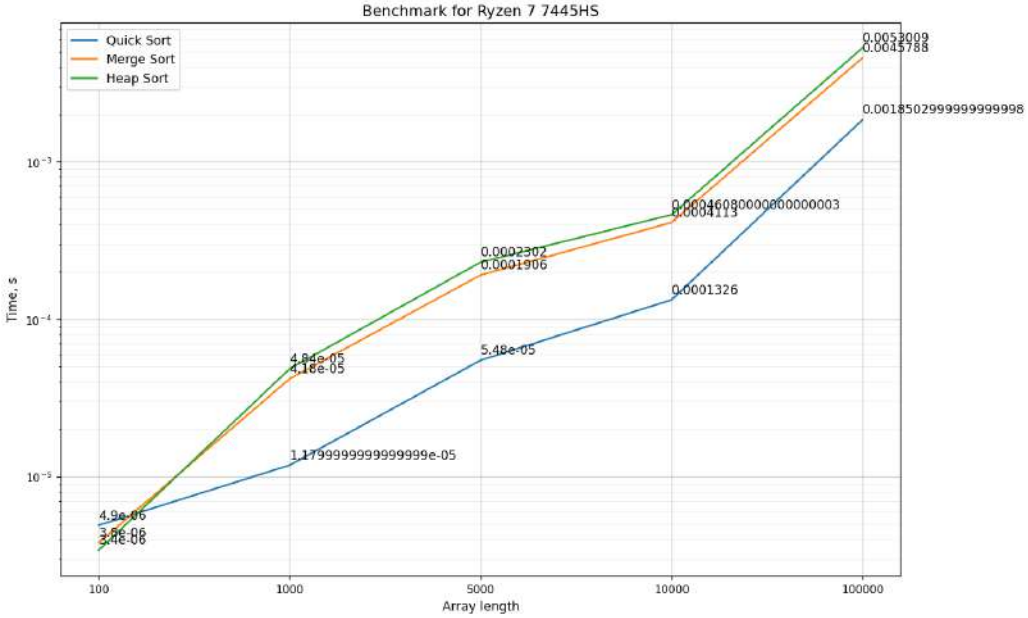

Рис. 1 Тест швидкості різних алгоритмів асимптотики n log n з логарифмічною шкалою часу

Висновки та перспективи. В результаті порівняння швидкості виконання алгоритмів сортування і їхнього використання пам'яті, ми склали наступну таблицю

Таблиця 1.

Порівняння алгоритмів сортування за асимптотичною швидкістю та використанням пам'яті

Алгоритм	Час			Пам'ять	
	$O(n)$	$\theta(n)$	$\Omega(n)$	$O(n)$	$\Omega(n)$
Bubble Sort	n^2	n^2	n	1	
Merge Sort	$n \log n$			n	
Quick Sort	n^2	$n \log n$	$n \log n$	n	$\log n$
Heap Sort	$n \log n$			1	

Як результат маємо, що найкращими алгоритмами сортування з точки зору швидкодії є сортування злиттям і кучами, а з точки зору використання пам'яті – сортування бульбашкою і кучами. Проте це не значить, що сортування кучами є самим ефективним, адже тести швидкості (див. Рис. 1) показують, що сортування кучами і злиттям значно програють по швидкості швидкому сортуванню. Це можна пояснити значними відмінностями в константних факторах, які виникають при розрахунку асимптотик, і які мають значний вплив при порівнянні алгоритмів однакових асимптотик. З цього слідує, що алгоритми сортування слід обирати адаптивно в залежності від поставленої задачі. Наприклад, в бібліотеці numpy це реалізовано у вигляді типу сортування stable, який адаптивно обирає tim sort або radix sort в залежності від даних.

Список використаних джерел

1. **Time and Space Complexity Analysis of Bubble Sort** [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/dsa/time-and-space-complexity-analysis-of-bubble-sort/ (дата звернення: 27.11.2025)
2. **Time Complexity of Merge Sort: A Detailed Analysis** [Електронний ресурс]. – Режим доступу: https://www.codecademy.com/article/time-complexity-of-merge-sort (дата звернення: 27.11.2025)
3. **Akra-Bazzi Method of Recurrence Time Complexity** [Електронний ресурс]. – Режим доступу: https://medium.com/@compuxela/akra-bazzi-method-of-recurrence-time-complexity-46a6a8814b47 (дата звернення: 27.11.2025)
4. **Quick Sort** [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/dsa/quick-sort-algorithm/ (дата звернення: 28.11.2025)
5. **Time and Space Complexity Analysis of Quick Sort** [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/dsa/time-and-space-complexity-analysis-of-quick-sort/ (дата звернення: 28.11.2025)
6. OLADIPUPO, E., TAIWO, A. O., CHRISTIANAH, A. Comparative study of two divide and conquer sorting algorithms: quicksort and mergesort // *Procedia Computer Science*. – 2020. – Vol. 171. – P. 2532–2540. – DOI: 10.1016/j.procs.2020.04.274
7. **Heap Sort** [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/dsa/heap-sort/ (дата звернення: 28.11.2025)