ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ОЛЕСЯ ГОНЧАРА

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

Лабораторна робота

з курсу «Оптимізація сервісної архітектури програмних додатків»

на тему «Сортування»

Виконала:

студентка групи ПА-22-2

Крюкова Маргарита Вікторівна

Дніпро, 2024

Зміст

Постановка завдання

Розробити програму, яка визначає час сортування масиву.

Результат виводити у вигляді таблиці, де назви стовпців - це кількість елементів масиву, а назви рядків - назва методу, що використовувався для сортування. Для порівняння використати не менше 6 різних алгоритмів сортування.

Рекомендовані методи сортування:

1. **Insertion Sort або Selection Sort**
2. Bubble Sort та / або Cocktail Sort
3. **Quick Sort**
4. Merge Sort
5. **Count Sort**
6. **Radix Sort**
7. Basket Sort
8. **Timsort**
9. **Binary tree sort**

1) Порівняти час сортування для масивів з кількістю елементів 10, 1000, 10 000, 1 000 000, 100 000 000 (рандомні з великого інтервалу)

2) Порівняти час сортування для масивів з кількістю елементів 10, 1000, 10 000, 1 000 000, 100 000 000 (рандомні з малого інтервалу)

3) Порівняти час сортування за зростанням для масивів з кількістю елементів 10, 1000, 10 000, 1 000 000, 100 000 000 (рандомні з великого інтервалу, які попередньо впорядковані за спаданням)

4) Порівняти час сортування за зростанням для масивів з кількістю елементів 10, 1000, 10 000, 1 000 000, 100 000 000 (рандомні з малого інтервалу, які попередньо впорядковані за спаданням)

Зробити висновки, де будуть написані рекомендації щодо використання випробуваних алгоритмів сортування.

Опис розв’язку

Insertion Sort.

Алгоритм працює так, що на кожному кроці він бере один елемент і вставляє його у вже відсортовану частину масиву, переміщаючи більші елементи праворуч.

Починаємо з другого елемента масиву (індекс 1), оскільки перший елемент вважається вже відсортованим. Порівнюємо їх і якщо поточний елемент менший за попередній, то їх обмінюємо місцями. Переходимо до наступного елемента і продовжуємо порівняння допоки весь масив не буде відсортованим.

Bubble Sort.

Алгоритм сортування працює за принципом поступового "виштовхування" найбільших елементів у кінець масиву. Він порівнює сусідні елементи та обмінює їх місцями, якщо вони розташовані в неправильному порядку.

Починаємо з першого елемента масиву і порівнюємо його з наступним елементом. Якщо поточний елемент більший за наступний, обмінюємо їх місцями. Продовжуємо порівнювати сусідні елементи по всьому масиву, просуваючись від початку до кінця. Алгоритм завершується, коли під час проходу по масиву не відбувається жодних обмінів, що означає, що масив повністю відсортовано

Quick Sort.

Обираючи опорний елемент, розбиваючи масив на дві частини: одну з елементами меншими за опорний, а іншу — з більшими. Потім цей процес повторюється рекурсивно для кожної частини. Як тільки всі під масиви будуть відсортовані, алгоритм буде завершений.

Merge Sort.

Алгоритм працює шляхом рекурсивного розбиття масиву на дві частини, сортування кожної частини окремо, а потім об'єднання (злиття) відсортованих частин у один відсортований масив.

Рекурсивно поділяємо масив на дві частини, поки кожна частина не містить лише один елемент. Коли частини масиву вже розбиті до одиничних

елементів, вони починають об'єднуватися. На кожному кроці з двох відсортованих під масивів формуються більші відсортовані масиви. Злиття здійснюється за допомогою порівняння елементів з двох масивів і вставлення їх у правильному порядку.

Count Sort.

Алгоритм, який працює за допомогою підрахунку кількості елементів в масиві, які мають певні значення, і потім використовує ці підрахунки для побудови відсортованого масиву.

Підраховуємо, скільки разів кожен елемент зустрічається в масиві. Для цього створюємо допоміжний масив (лінійний масив), в якому індекси відповідають можливим значенням елементів, а значення елементів — кількості їхніх появ. Після підрахунку кількості кожного елементу, створюємо кумулятивний масив, де кожен елемент зберігає суму всіх попередніх підрахунків. Це дає змогу визначити, на якому місці в відсортованому масиві має бути кожен елемент. Виходячи з отриманих підрахунків, алгоритм формує відсортований масив, розміщуючи елементи на їх правильні місця згідно з їх частотою.

Timsort.

Спочатку сортуємо кожен під масив за допомогою сортування вставками. Ми розбиваємо масив на під масиви розміру RUN і сортуємо їх за допомогою insertion_sort. Потім починаємо зливати ці відсортовані під масиви у більші під масиви, поки весь масив не стане відсортованим.

Після написання функцій для сортувальних алгоритмів, були написані функції для автоматичного створення масиву з рандомними елементами, для підрахунку часу, що був витрачений на сортування та для впорядкування елементів за спаданням, щоб потім застосувати алгоритми сортування.

У головній функції для запуску програми, виводимо назву таблиці та її шапку, викликаємо функцію створення масиву, сортуємо його з засіканням часу, виводимо результат у таблицю і повторюємо для кожного алгоритму і розміру.

Таких таблиць створюємо чотири - «Таблиця для часу сортування масиву з малим інтервалом», «Таблиця для часу сортування масиву з великим інтервалом», «Таблиця для часу сортування масиву з малим інтервалом, що відсортований за спаданням» та «Таблиця для часу сортування масиву з великим інтервалом, що відсортований за спаданням».

Керівництво користувача

Під час запуску програми, на екран виводиться назва таблиці – «Таблиця для часу сортування масиву з малим інтервалом» («Таблиця для часу сортування масиву з великим інтервалом», «Таблиця для часу сортування масиву з малим інтервалом, що відсортований за спаданням» або «Таблиця для часу сортування масиву з великим інтервалом, що відсортований за спаданням»).

Після виводу назви таблиці, виводиться шапка з назвами методів сортування. На наступному рядку спочатку виводиться розмір масиву, а потім час, що був витрачений на його сортування методом, під яким цей час і вивівся. Сортування відбувається для масивів з кількістю елементів 10, 1000, 10000 та 100000.

Тестові приклади

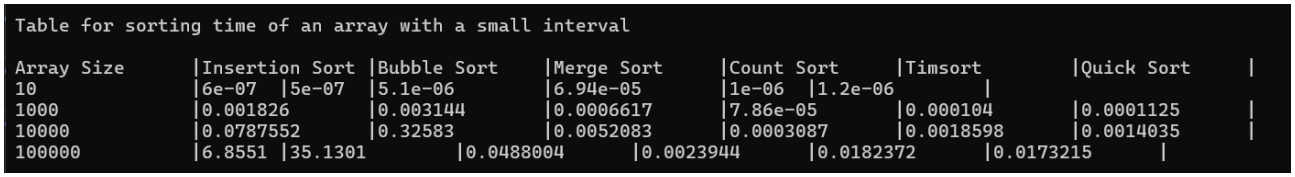


Table for sorting time of an array with a small interval

Array Size	Insertion Sort	Bubble Sort	Merge Sort	Count Sort	Timsort	Quick Sort
10	6e-07 |5e-07	5.1e-06	6.94e-05	1e-06 |1.2e-06		
1000	0.001826	0.003144	0.0006617	7.86e-05	0.000104	0.0001125
10000	0.0787552	0.32583	0.0052083	0.0003087	0.0018598	0.0014035
100000	6.8551 |35.1301	0.0488004	0.0023944	0.0182372	0.0173215	

Рисунок 1 - Таблиця для часу сортування масивів розміром 10, 1000, 10000, 100000 з малим інтервалом

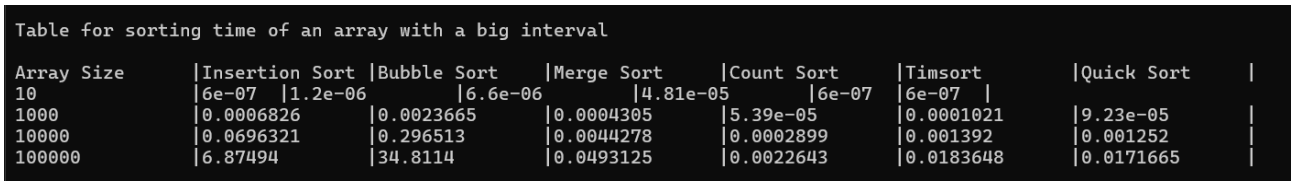


Table for sorting time of an array with a big interval

Array Size	Insertion Sort	Bubble Sort	Merge Sort	Count Sort	Timsort	Quick Sort
10	6e-07 |1.2e-06	6.6e-06	4.81e-05	6e-07	6e-07 |	
1000	0.0006826	0.0023665	0.0004305	5.39e-05	0.0001021	9.23e-05
10000	0.0696321	0.296513	0.0044278	0.0002899	0.001392	0.001252
100000	6.87494	34.8114	0.0493125	0.0022643	0.0183648	0.0171665

Рисунок 2 - Таблиця для часу сортування масивів розміром 10, 1000, 10000, 100000 з великим інтервалом

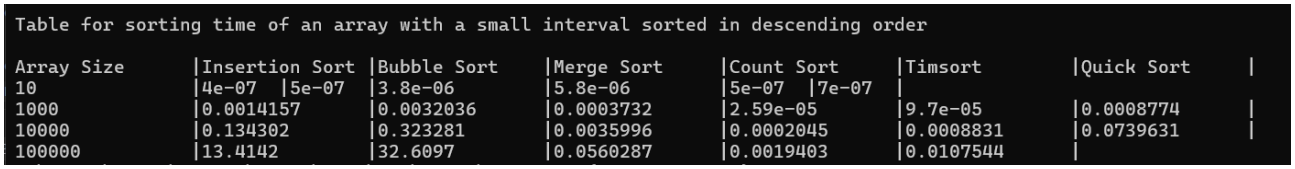
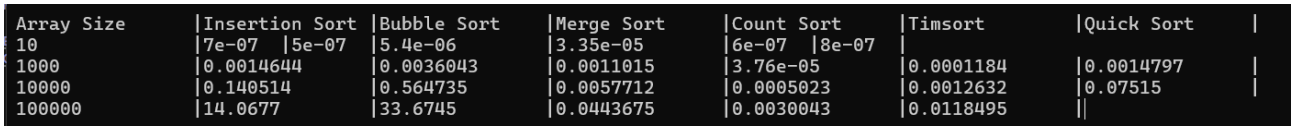


Table for sorting time of an array with a small interval sorted in descending order

Array Size	Insertion Sort	Bubble Sort	Merge Sort	Count Sort	Timsort	Quick Sort
10	4e-07 |5e-07	3.8e-06	5.8e-06	5e-07 |7e-07		
1000	0.0014157	0.0032036	0.0003732	2.59e-05	9.7e-05	0.0008774
10000	0.134302	0.323281	0.0035996	0.0002045	0.0008831	0.0739631
100000	13.4142	32.6097	0.0560287	0.0019403	0.0107544	

Рисунок 3 - Таблиця для часу сортування масивів розміром 10, 1000, 10000, 100000 з малим інтервалом, що відсортований за спаданням



Array Size	Insertion Sort	Bubble Sort	Merge Sort	Count Sort	Timsort	Quick Sort
10	7e-07 |5e-07	5.4e-06	3.35e-05	6e-07 |8e-07		
1000	0.0014644	0.0036043	0.0011015	3.76e-05	0.0001184	0.0014797
10000	0.140514	0.564735	0.0057712	0.0005023	0.0012632	0.07515
100000	14.0677	33.6745	0.0443675	0.0030043	0.0118495	

Рисунок 4 - Таблиця для часу сортування масивів розміром 10, 1000, 10000, 100000 з великим інтервалом, що відсортований за спаданням

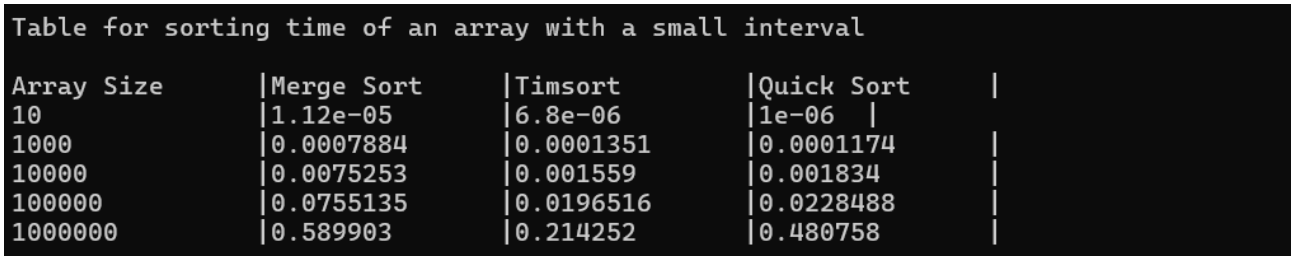


Table for sorting time of an array with a small interval

Array Size	Merge Sort	Timsort	Quick Sort
10	1.12e-05	6.8e-06	1e-06 |
1000	0.0007884	0.0001351	0.0001174
10000	0.0075253	0.001559	0.001834
100000	0.0755135	0.0196516	0.0228488
1000000	0.589903	0.214252	0.480758

Рисунок 5 - Таблиця для часу сортування Timsort, Merge Sort і Quick Sort масивів розміром 10, 1000, 10000, 100000, 1000000 з малим інтервалом

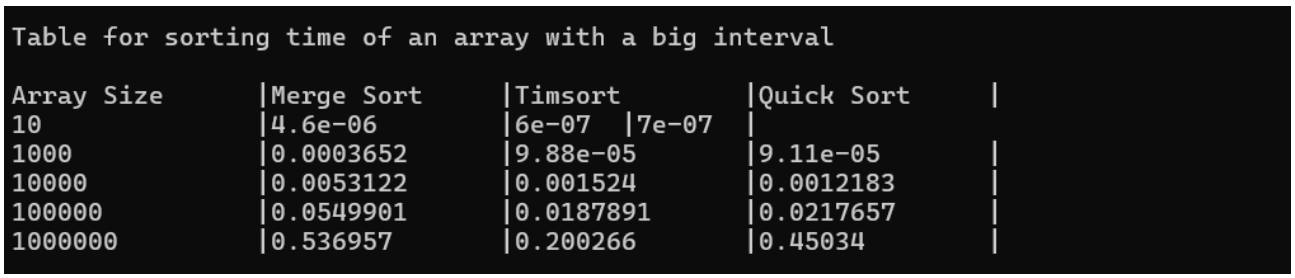


Рисунок 6 - Таблиця для часу сортування Timsort, Merge Sort і Quick Sort масивів розміром 10, 1000, 10000, 100000, 1000000 з великим інтервалом

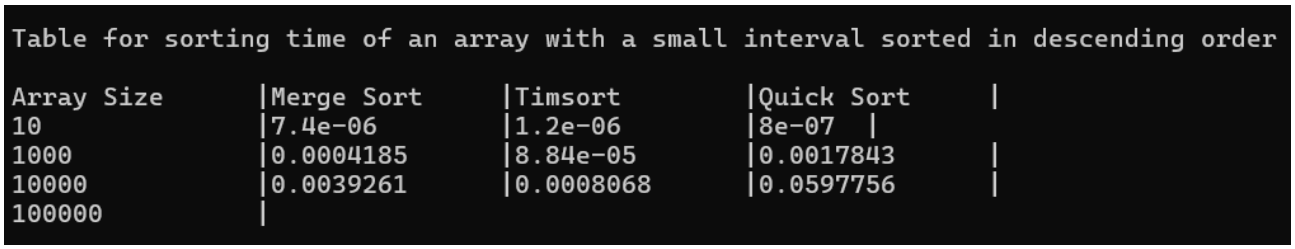


Рисунок 7 - Таблиця для часу сортування Timsort, Merge Sort і Quick Sort масивів розміром 10, 1000, 10000, 100000, 1000000 з малим інтервалом, що відсортований за спаданням

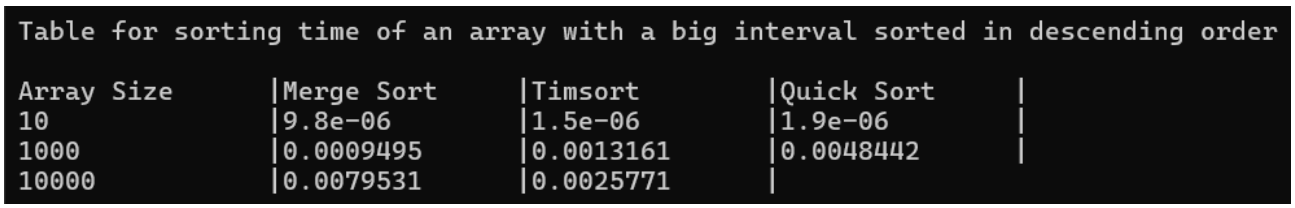


Рисунок 8 - Таблиця для часу сортування Timsort, Merge Sort і Quick Sort масивів розміром 10, 1000, 10000, 100000, 1000000 з великим інтервалом, що відсортований за спаданням

Висновок

Під час виконання лабораторної роботи, я попрацювала з декількома алгоритмами сортування, а саме з Insertion Sort, Bubble Sort, Quick Sort, Merge Sort, Count Sort, Timsort.

Insertion Sort і Bubble Sort мають квадратичну часову складність $O(n^2)$ що робить їх **дуже повільними** для великих масивів. Наприклад, для масиву розміром 100 000 час виконання цих алгоритмів буде дуже великий, і вони не здатні ефективно обробляти такі розміри даних, що видно на Рис.1-4. Для більших розмірів стається перевантаження алгоритмів, також саме створення масивів займає набагато більше часу.

Merge Sort, Quick Sort і Timsort мають часову складність $O(n \log n)$, що значно ефективніше для великих масивів. Вони будуть працювати швидше, але все одно можуть потребувати значного часу при обробці таких великих обсягів даних, що можна побачити на Рис.5-7. Але для масивів з 10 000 елементів з великим інтервалом, що відсортований за спаданням стається перевантаження для Quick Sort, хоча час на сортування залишається досить малим (Рис.8).

Count Sort може бути ефективним, якщо значення елементів масиву обмежені невеликим діапазоном. Але для масивів великого розміру та великого діапазону чисел, він може вимагати великої кількості пам'яті і не буде працювати швидко, якщо діапазон чисел дуже великий.

Для масивів розміром 100 000 000 потрібен значний обсяг пам'яті (близько 400 МБ для масиву цілих чисел на 32 бітних системах).

Сорти типу Count Sort можуть вимагати додаткової пам'яті для створення проміжних масивів, що може бути проблемою для дуже великих масивів.

Timsort і Quick Sort є оптимізованими і часто використовуються в реальних застосунках для сортування великих масивів.

Merge Sort може бути трохи повільнішим за Quick Sort в середньому, але його поведінка стабільніша в різних ситуаціях.

Тож, алгоритми сортування з часом $O(n \log n)$ (як Timsort, Merge Sort і Quick Sort) можуть працювати з масивами розміром до 100 мільйонів (у моєму випадку до 100 000), але це вимагатиме значного часу для їх виконання, особливо на великих масивах. В той час як Bubble Sort та Insertion Sort для таких розмірів масивів будуть надзвичайно повільними і не підходять для реальних застосувань.