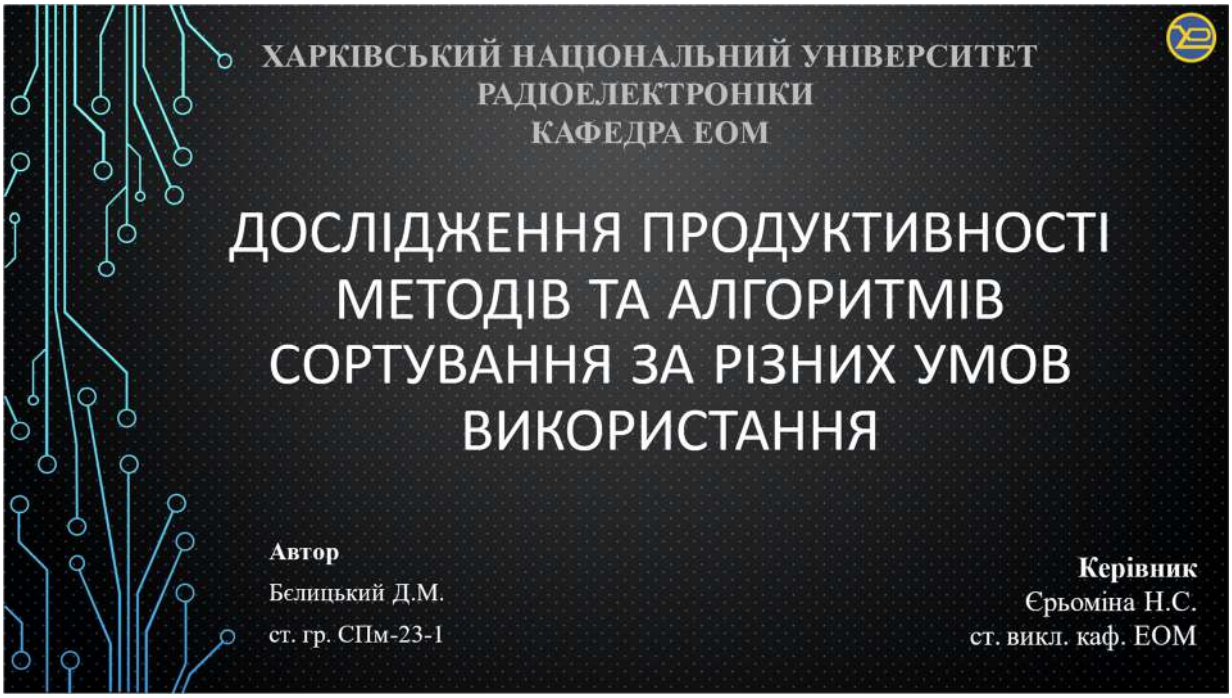
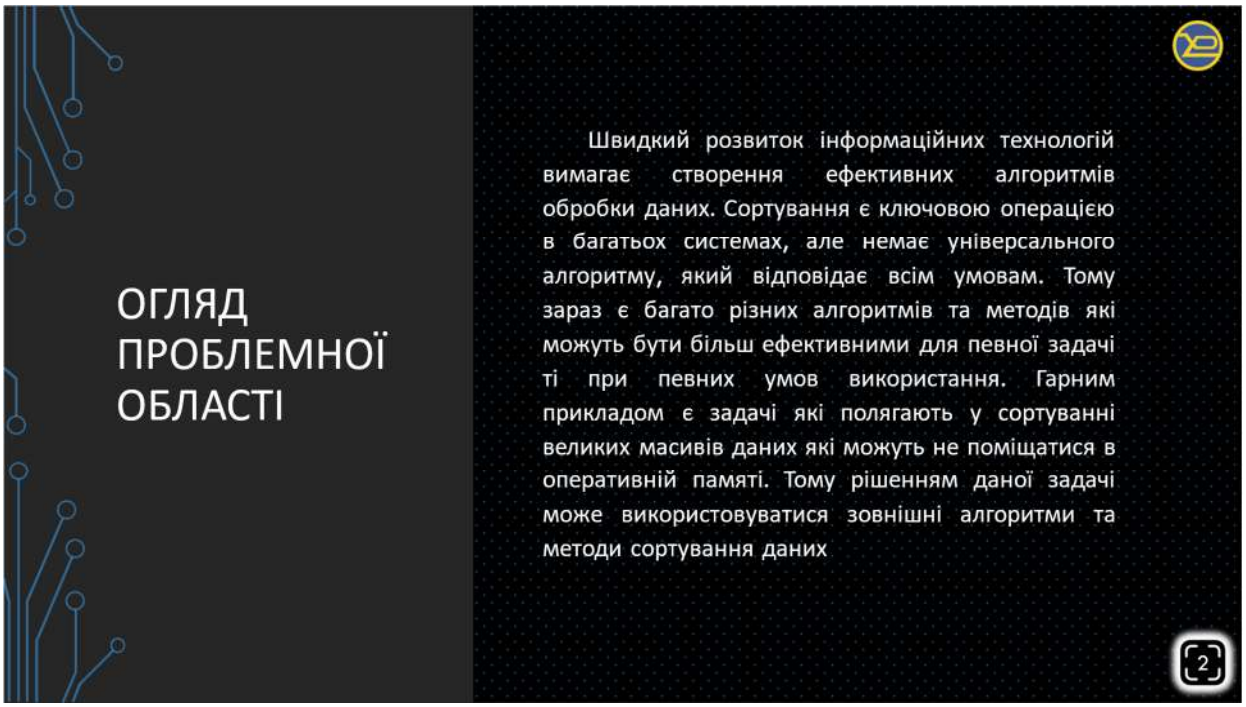
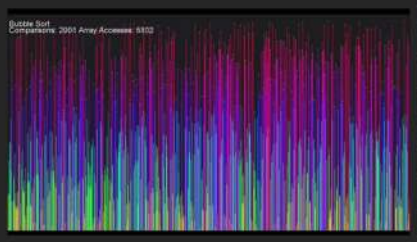
ДОДАТОК А

Графічний матеріал кваліфікаційної роботи


ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ
КАФЕДРА ЕОМ
ДОСЛІДЖЕННЯ ПРОДУКТИВНОСТІ МЕТОДІВ ТА АЛГОРИТМІВ СОРТУВАННЯ ЗА РІЗНИХ УМОВ ВИКОРИСТАННЯ
Автор
Бєлицький Д.М.
ст. гр. СПм-23-1
Керівник
Єрьоміна Н.С.
ст. викл. каф. ЕОМ


ОГЛЯД ПРОБЛЕМНОЇ ОБЛАСТІ
Швидкий розвиток інформаційних технологій вимагає створення ефективних алгоритмів обробки даних. Сортування є ключовою операцією в багатьох системах, але немає універсального алгоритму, який відповідає всім умовам. Тому зараз є багато різних алгоритмів та методів які можуть бути більш ефективними для певної задачі ті при певних умов використання. Гарним прикладом є задачі які полягають у сортуванні великих масивів даних які можуть не поміщатися в оперативній памяті. Тому рішенням даної задачі може використовуватися зовнішні алгоритми та методи сортування даних
2

АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ



Демонстрація сортування бульбашкою

Сортування є фундаментальною операцією у багатьох інформаційних системах. Існує велика кількість алгоритмів, які різняться за ефективністю, складністю реалізації та адаптованістю до різних умов.

Класифікація алгоритмів сортування

1. **За методом реалізації:**

- Порівняльні алгоритми: сортування бульбашкою, швидке сортування (QuickSort), сортування злиттям (MergeSort), купчасте сортування (HeapSort), сортування вибором, вставками.
- Непорівняльні алгоритми: сортування підрахунком (CountingSort), сортування комірками (BucketSort), радікс-сортування (RadixSort).

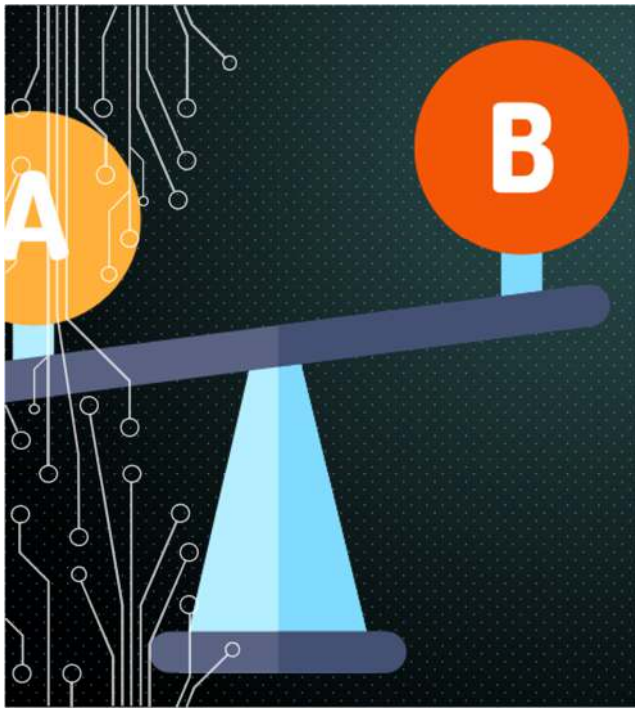
2. **За ефективністю:**

- Прості алгоритми ($O(n^2)$): сортування бульбашкою, вибором, вставками — підходять для невеликих масивів.
- Ефективні алгоритми ($O(n \log n)$): швидке сортування, злиття, купчасте сортування.

3. **За стабільністю:**

- Стабільні: зберігають порядок елементів з однаковими ключами (MergeSort, CountingSort).
- Нестабільні: не гарантують збереження порядку (HeapSort, QuickSort).

3



МЕТА РОБОТИ

Метою даного дослідження є отримання продуктивності обраних алгоритмів та методів сортування в різних умовах використання. Також в ході виконання роботи будуть виявлятися слабкі або сильні сторони кожного з методу або алгоритму сортування. Ця інформація дозволить швидше обирати потрібний метод сортування та відштовхуючись від отриманих результатів обирати найбільш оптимальний

4

ЗАДАЧІ

- Проаналізувати існуючі алгоритми та методи сортування
- Розглянути популярні модифікації
- Обрати умови за якими будуть проводитися тести
- Реалізувати шаблон для тестування алгоритмів та методів
- Провести тестування кожного сортування
- Проаналізувати отримані результати

5

АКТУАЛЬНІСТЬ ОБРАНОЇ ТЕМИ

Кожного дня збільшується кількість даних та досить часто постає задача з сортування даних. Якщо масив даних має малий розмір та не має багато критеріїв що до продуктивності то є багато варіантів для вирішення проблеми з сортуванням. Але якщо для нас є важливим критерієм її продуктивність то наш вибір сильно зменшується. На сьогодні ми не маємо алгоритм або метод який буде швидким, простим, займати мінімум ОП та бути легким для модифікації водночас. Також не слід забувати що сьогодні є тенденція збільшення кількості ядер процесору що може бути вирішальним критерієм при виборі алгоритму чи методу сортування

Percent Change in Average Monthly In-Home Data Usage by Device

2019 VS. 2024 - GIGABYTES RECEIVED

	CONNECTED TV	GAMING CONSOLE	PC/MAC	PHONE	SMART SPEAKER	STREAMING BOX/STICK	TABLET	GRAND TOTAL
JAN	26%	6%	-3%	21%	7%	24%	18%	16%
FEB	22%	12%	-5%	27%	-4%	21%	15%	16%
MAR*	27%	12%	-4%	34%	30%	24%	12%	18%

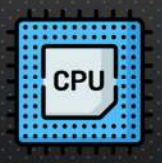
Тенденція збільшення використання даних

6

ЗАПРОПОНОВАНЕ РІШЕННЯ

Для того щоб результати тестування менш залежали від випадкових подій в системі які можуть змінити результат тестування систему було налаштовано так щоб зменшити кількість операцій до мінімуму. Частоти процесору та пам'яті були зафіксовані. Віртуальна пам'ять не були вимкнута тому як її вимкнення хоча і може збільшити продуктивність, але це може впливати на стабільність системи та результату.

1 SSD M2
1 SSD SATA
1 HDD



Ryzen r5 5600x
4.7ghz

4x fury 3600mhz

7

ЗАПРОПОНОВАНЕ РІШЕННЯ

Для того щоб виміряти продуктивність алгоритмів та методів сортування було використано 2 різні мови програмування щоб протестувати їх. Першою було обрано C++ тому що дана мова компілюється одразу в машинний код в той час як обрана наступна мова JavaScript спочатку компілюється в bit code, після чого виконуючись на віртуальній машині вже як машинний код. Даний підхід дозволяє перевірити наскільки сортування видає стабільний результат на різних за типом мовах програмування. Для кожної з мов було обрано інструменти для вимірювання як часу виконання так і кількості пам'яті яка використовується

8

ЗАПРОПОНОВАНЕ РІШЕННЯ

Критеріями для тестування були обрані наступні:

- час виконання з повільною ОП;
- час виконання з швидкою ОП;
- максимальна кількість використаної ОП з частотою 2400;
- максимальна кількість використаної ОП з частотою 3600.

Для алгоритмів які мають непогану та доступну багатопоточну реалізацію або мають зовнішню реалізацію сортування було додатково проведено тестування та аналіз

9

ЗАПРОПОНОВАНЕ РІШЕННЯ

Для вимірювання часу сортування було використано наступний підхід:

- Вимірювання починається в момент перед викликом функції сортування
- Вимірювання закінчується коли функція завершує сортування

Для вимірювання ОП було використано метод критичних точок. Критична точка це місце де в алгоритмі або методі може бути використана найбільша кількість ОП. В цьому місці або місцях буде замірятися використана додатком ОП. В результаті буде отримано максимальне значення використання пам'яті.

Но через даний підхід буде збільшений час виконання через додаткові операції замірювання. Тому кожен алгоритм під час тестування має 2 реалізації а саме просту та з вставленими критичними точками у алгоритмі.

```
function mergeSort(arr, left, right) : void
    trackMaxMemoryUsage();

    if (left >= right) {
        return;
    }
```

Замірювання часу

```
async function getSortTime(arr, method) : Promise<any>
    const startTime : number  = performance.now();

    await method(arr);

    const endTime : number  = performance.now();
```

Замірювання ОП

10

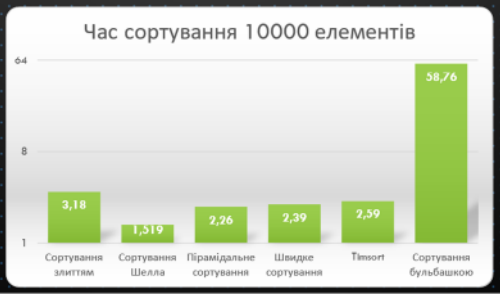
ОТРИМАНІ РЕЗУЛЬТАТИ

В результаті було реалізовано такі методи та алгоритми сортування:

- Сортування злиттям а також зовнішня та багатопоточна реалізація
- Сортування Шелла з послідовністю Циура
- Пірамідальне сортування
- Dual pivot швидке сортування
- Timsort а також зовнішня реалізація
- Сортування бульбашкою

11

ОТРИМАНІ РЕЗУЛЬТАТИ



12

ОТРИМАНІ РЕЗУЛЬТАТИ

Таблиця с загальними результатами

	100ел.	10000ел.	1000000ел.	R-S	Пам'ять	JS-C++
Сортування злиттям	0.0899мс	3.18мс	178.58мс	5.72%	0.34%	3.02%
Сортування Шелла	0.0718мс	1.519мс	110.06мс	15.51%	1.35%	6.89%
Пірамідальне сортування	0.152мс	2.26мс	147.29мс	6.79%	4.23%	48.42%
Швидке сортування	0.152мс	2.39мс	1709.62мс	-1507%	1.94%	33.63%
Timsort	0.0873мс	2.59мс	96.91мс	10.71%	0.42%	90.62%
Сортування бульбашкою	0.286мс	58.76мс	1061170мс	16.25%	5.55%	69.30%

13

АНАЛІЗ РЕЗУЛЬТАТІВ

Загальний вплив сортованих послідовностей дорівнює 10.196%.

Різниця між мовами програмування становила 41.98%.

При зміні частоти оперативної пам'яті з 2400 на 3600 на зменшення затримки таймінгів швидкість виконання збільшилась 2.31%.

Зовнішнє сортування загалом в 9-25 разів повільніша но використовую в 8-9 разів менше оперативної пам'яті.

Використання кількості потоків, що дорівнює логічним ядрам процесора, є оптимальним. Перевищення цієї кількості знижує продуктивність через збільшення накладних витрат. В загалом кожне ядро додає від 20 до 40 відсотків. Якщо використовується додатковий поток на тому самому ядрі то продуктивність збільшується на 5-15%.

Для маленьких масивів кращий метод це сортування Шелла з послідовністю Циура.

Для середніх Шелла з послідовністю Циура або Timsort.

Для великих масивів Timsort.

Самим стабільним алгоритмом сортування за часом виконання було сортування злиттям.

14

ВИСНОВКИ

У даній роботі досліджено продуктивність алгоритмів сортування в різних умовах. Для досягнення мети проведено комплексну підготовку, включаючи аналіз алгоритмів, створення блок-схем і визначення їх сильних та слабких сторін.

Система була налаштована для забезпечення стабільності тестів, щоб уникнути впливу фонових завдань. Обрано шість алгоритмів сортування, для деяких з них реалізовано зовнішнє та паралельне сортування. Створено допоміжні функції для прискорення тестування та уникнення помилок у підрахунках.

Результати показали відмінності в продуктивності алгоритмів від очікуваних теоретичних показників. Дослідження досягло своєї мети, продемонструвавши, як змінюються результати сортування за різних умов. Незважаючи на труднощі з реалізацією на різних мовах програмування, усі проблеми були вирішені оптимальними методами.

15

АПРОБАЦІЯ РЕЗУЛЬТАТІВ

1. Bielytskyi D.M., Yeromina N.S. Ponomarenko O.M. IN SEARCH OF THE PERFECT ALGORITHM: THE INSOLUBLE RIDDLE OF COMPUTER TECHNOLOGY // Проблеми інформатизації : XII міжнародна науково-технічна конференція. - 21-22 листопада 2024. –с.78. doi: https://doi.org/10.32620/PI.24.t2

2. Bielytskyi, D., & Yeromina, N. (2024). Research on the Performance of Sorting Methods and Algorithms under Various Usage Conditions. COMPUTER AND INFORMATION SYSTEMS AND TECHNOLOGIES, 2024, 17–18.

16