

Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний
інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра ІІІ

Звіт

з лабораторної роботи № 1 з дисципліни
«Алгоритми та структури даних 2. Структури даних»

„Проектування і аналіз алгоритмів внутрішнього сортування”

Виконав(ла)

Симотюк Денис Андрійович
(шифр, прізвище, ім'я, по батькові)

ІІІ-13

Перевірив

Сопов Олексій Олександрович
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ.....	5
3.1	АНАЛІЗ АЛГОРИТМУ НА ВІДПОВІДНІСТЬ ВЛАСТИВОСТЯМ.....	5
3.2	ПСЕВДОКОД АЛГОРИТМУ	5
3.3	АНАЛІЗ ЧАСОВОЇ СКЛАДНОСТІ.....	6
3.4	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	6
3.4.1	<i>Вихідний код.....</i>	<i>7</i>
3.4.2	<i>Приклад роботи</i>	<i>11</i>
3.5	ТЕСТУВАННЯ АЛГОРИТМУ	13
3.5.1	<i>Часові характеристики оцінювання.....</i>	<i>13</i>
3.5.2	<i>Графіки залежності часових характеристик оцінювання від розмірності масиву</i>	<i>15</i>
	ВИСНОВОК	ОШИБКА! ЗАКЛАДКА НЕ ОПРЕДЕЛЕНА.
	КРИТЕРІЇ ОЦІНЮВАННЯ	17

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні методи аналізу обчислювальної складності алгоритмів внутрішнього сортування і оцінити поріг їх ефективності.

2 ЗАВДАННЯ

Виконати аналіз алгоритму внутрішнього сортування на відповідність наступним властивостям (таблиця 2.1):

- стійкість;
- «природність» поведінки (Adaptability);
- базуються на порівняннях;
- необхідність додаткової пам'яті (об'єму);
- необхідність в знаннях про структуру даних.

Записати алгоритм внутрішнього сортування за допомогою псевдокоду (чи іншого способу по вибору).

Провести аналіз часової складності в гіршому, кращому і середньому випадках та записати часову складність в асимптотичних оцінках.

Виконати програмну реалізацію алгоритму на будь-якій мові програмування з фіксацією часових характеристик оцінювання (кількість порівнянь, кількість перестановок, глибина рекурсивного поглиблення та інше в залежності від алгоритму).

Провести ряд випробувань алгоритму на масивах різної розмірності (10, 100, 1000, 5000, 10000, 20000, 50000 елементів) і різних наборів вхідних даних (впорядкований масив, зворотно упорядкований масив, масив випадкових чисел) і побудувати графіки залежності часових характеристик оцінювання від розмірності масиву, нанести на графік асимптотичну оцінку гіршого і кращого випадків для порівняння.

Зробити порівняльний аналіз двох алгоритмів.

Зробити узагальнений висновок з лабораторної роботи.

Таблиця 2.1 – Варіанти алгоритмів

№	Алгоритм сортування
1	Сортування бульбашкою
2	Сортування гребінцем («розчіскою»)

3 ВИКОНАННЯ

3.1 Аналіз алгоритму на відповідність властивостям

Аналіз алгоритму **сортування бульбашкою** на відповідність властивостям наведено в таблиці 3.1.

Таблиця 3.1 – Аналіз алгоритму на відповідність властивостям

Властивість	Сортування бульбашкою
Стійкість	+
«Природність» поведінки (Adaptability)	+
Базуються на порівняннях	+
Необхідність в додатковій пам'яті (об'єм)	-
Необхідність в знаннях про структури даних	-

Властивість	Сортування гребінцем
Стійкість	+
«Природність» поведінки (Adaptability)	-
Базуються на порівняннях	+
Необхідність в додатковій пам'яті (об'єм)	-
Необхідність в знаннях про структури даних	-

3.2 Псевдокод алгоритму

Сортування бульбашкою:

повторити для i від 0 до $size - 1$:

повторити для j від 0 до $\text{size} - 1$:

якщо $\text{arr}[j] > \text{arr}[j + 1]$:

$\text{temp} = \text{arr}[j]$

$\text{arr}[j] = \text{arr}[j + 1]$

$\text{arr}[j + 1] = \text{temp}$

все якщо

все повторити

все повторити

Сортування гребінцем:

$\text{step} = \text{size} - 1$

повторити поки $\text{step} \geq 1$:

повторити для i від 0 до $\text{size} - \text{step}$:

якщо $\text{arr}[i] > \text{arr}[i + \text{step}]$:

$\text{temp} = \text{arr}[i]$

$\text{arr}[i] = \text{arr}[i + \text{step}]$

$\text{arr}[i + \text{step}] = \text{temp}$

все якщо

все повторити

$\text{step} /= 1.247$

все повторити

3.3 Аналіз часової складності

	Сортування бульбашкою
найгірший випадок	$O(n^2)$
середній випадок	$O(n^2)$
найкращий випадок	$O(n)$

	Сортування гребінцем
найгірший випадок	$O(n^2)$
середній випадок	$O(n^2)$
найкращий випадок	$O(n)$

3.4 Програмна реалізація алгоритму

3.4.1 Вихідний код

```
#include <iostream>
using namespace std;

void worst_bubble(int size, int* count_swaps, int* count_comps);
void best_bubble(int size, int* count_swaps, int* count_comps);
void output_stats(int count_swaps, int count_comps);
void sort_bubble(int size, int* count_swaps, int* count_comps, int els);
void sort_comb(int size, int* count_swaps, int* count_comps, int els);
void output_arr(int* arr, int size, int els);
void worst_comb(int size, int* count_swaps, int* count_comps);
void best_comb(int size, int* count_swaps, int* count_comps);

int main()
{
    srand(int(time(NULL)));
    int count_swaps = 0, count_comps = 0, size, els;

    cout << "Enter a number of elements: ";
    cin >> size;
    cout << endl;

    cout << "How many elements of array do you want to display: ";
    cin >> els;

    cout << "BUBBLE SORT:" << endl;
    cout << "The worst occasion: \n";
    worst_bubble(size, &count_swaps, &count_comps);
    output_stats(count_swaps, count_comps);
    cout << endl;

    cout << "The best occasion: \n";
    best_bubble(size, &count_swaps, &count_comps);
    output_stats(count_swaps, count_comps);
    cout << endl;

    cout << "Random occasion: \n";
    sort_bubble(size, &count_swaps, &count_comps, els);
    output_stats(count_swaps, count_comps);

    cout << "COMB SORT:" << endl;

    cout << "The worst occasion: \n";
    worst_comb(size, &count_swaps, &count_comps);
    output_stats(count_swaps, count_comps);
}
```

```

    cout << endl;

    cout << "The best occasion: \n";
    best_comb(size, &count_swaps, &count_comps);
    output_stats(count_swaps, count_comps);
    cout << endl;

    cout << "Random occasion: \n";
    sort_comb(size, &count_swaps, &count_comps, els);
    output_stats(count_swaps, count_comps);
}

void worst_bubble(int size, int* count_swaps, int* count_comps)
{
    *count_swaps = 0;
    *count_comps = 0;
    int* arr = new int[size];
    for (int i = 0; i < size; i++)
        arr[i] = size - i;
    for (int i = 0; i < size; i++)
    {
        for (int j = 1; j < size - i; j++)
        {
            *count_comps += 1;
            if (arr[j - 1] > arr[j])
            {
                int temp = arr[j];
                arr[j] = arr[j - 1];
                arr[j - 1] = temp;
                *count_swaps += 1;
            }
        }
    }
}

void best_bubble(int size, int* count_swaps, int* count_comps)
{
    *count_swaps = 0;
    *count_comps = 0;
    int* arr = new int[size];
    for (int i = 0; i < size; i++)
        arr[i] = i;
    for (int i = 0; i < size; i++)
    {
        for (int j = 1; j < size - i; j++)
        {
            *count_comps += 1;
            if (arr[j - 1] > arr[j])
            {
                int temp = arr[j];
                arr[j] = arr[j - 1];
                arr[j - 1] = temp;
                *count_swaps += 1;
            }
        }
    }
}

void output_stats(int count_swaps, int count_comps)
{
    cout << "Number of comparisons: " << count_comps << endl;
    cout << "Number of swaps: " << count_swaps << endl;
}

```



```

void sort_bubble(int size, int *count_swaps, int* count_comps, int els)
{
    *count_swaps = 0;
    *count_comps = 0;
    int* arr = new int[size];

    for (int i = 0; i < size; i++)
        arr[i] = rand();

    cout << "Unsorted array: ";
    output_arr(arr, size, els);
    cout << endl;

    for (int i = 0; i < size; i++)
    {
        for (int j = 1; j < size - i; j++)
        {
            *count_comps += 1;
            if (arr[j - 1] > arr[j])
            {
                int temp = arr[j];
                arr[j] = arr[j - 1];
                arr[j - 1] = temp;
                *count_swaps += 1;
            }
        }
    }

    cout << "Sorted array: ";
    output_arr(arr, size, els);
    cout << endl;
}

void sort_comb(int size, int* count_swaps, int* count_comps, int els)
{
    *count_swaps = 0;
    *count_comps = 0;
    int* arr = new int[size];

    for (int i = 0; i < size; i++)
        arr[i] = rand();

    cout << "Unsorted array: ";
    output_arr(arr, size, els);
    cout << endl;

    int step = size - 1;
    while (step >= 1)
    {
        for (int i = 0; i < size - step; i++)
        {
            *count_comps += 1;
            if (arr[i + step] < arr[i])
            {
                int temp = arr[i];
                arr[i] = arr[i + step];
                arr[i + step] = temp;
                *count_swaps += 1;
            }
        }
        step /= 1.247;
    }

    cout << "Sorted array: ";
    output_arr(arr, size, els);
}

```

```

    cout << endl;
}

void worst_comb(int size, int* count_swaps, int* count_comps)
{
    *count_swaps = 0;
    *count_comps = 0;
    int* arr = new int[size];
    for (int i = 0; i < size; i++)
        arr[i] = size - i;
    int step = size - 1;
    while (step >= 1)
    {
        for (int i = 0; i < size - step; i++)
        {
            *count_comps += 1;
            if (arr[i + step] < arr[i])
            {
                int temp = arr[i];
                arr[i] = arr[i + step];
                arr[i + step] = temp;
                *count_swaps += 1;
            }
        }
        step /= 1.247;
    }
}

void best_comb(int size, int* count_swaps, int* count_comps)
{
    *count_swaps = 0;
    *count_comps = 0;
    int* arr = new int[size];
    for (int i = 0; i < size; i++)
        arr[i] = i;
    int step = size - 1;
    while (step >= 1)
    {
        for (int i = 0; i < size - step; i++)
        {
            *count_comps += 1;
            if (arr[i + step] < arr[i])
            {
                int temp = arr[i];
                arr[i] = arr[i + step];
                arr[i + step] = temp;
                *count_swaps += 1;
            }
        }
        step /= 1.247;
    }
}

void output_arr(int* arr, int size, int els)
{
    for (int i = 0; i < els; i++)
        cout << arr[i] << " ";
    cout << endl;
}

```

3.4.2 Приклад роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми сортування масивів на 100 і 1000 елементів відповідно.

```
Консоль отладки Microsoft Visual Studio
Enter a number of elements: 100

How many elements of array do you want to display: 10
BUBBLE SORT:
The worst occasion:
Number of comparisons: 4950
Number of swaps: 4950

The best occasion:
Number of comparisons: 4950
Number of swaps: 0

Random occasion:
Unsorted array: 7035 849 19083 2015 10572 16759 10643 11668 16894 25389

Sorted array: 849 981 1798 2015 2341 3018 3125 3532 3614 3854

Number of comparisons: 4950
Number of swaps: 2644
COMB SORT:
The worst occasion:
Number of comparisons: 1233
Number of swaps: 106

The best occasion:
Number of comparisons: 1233
Number of swaps: 0

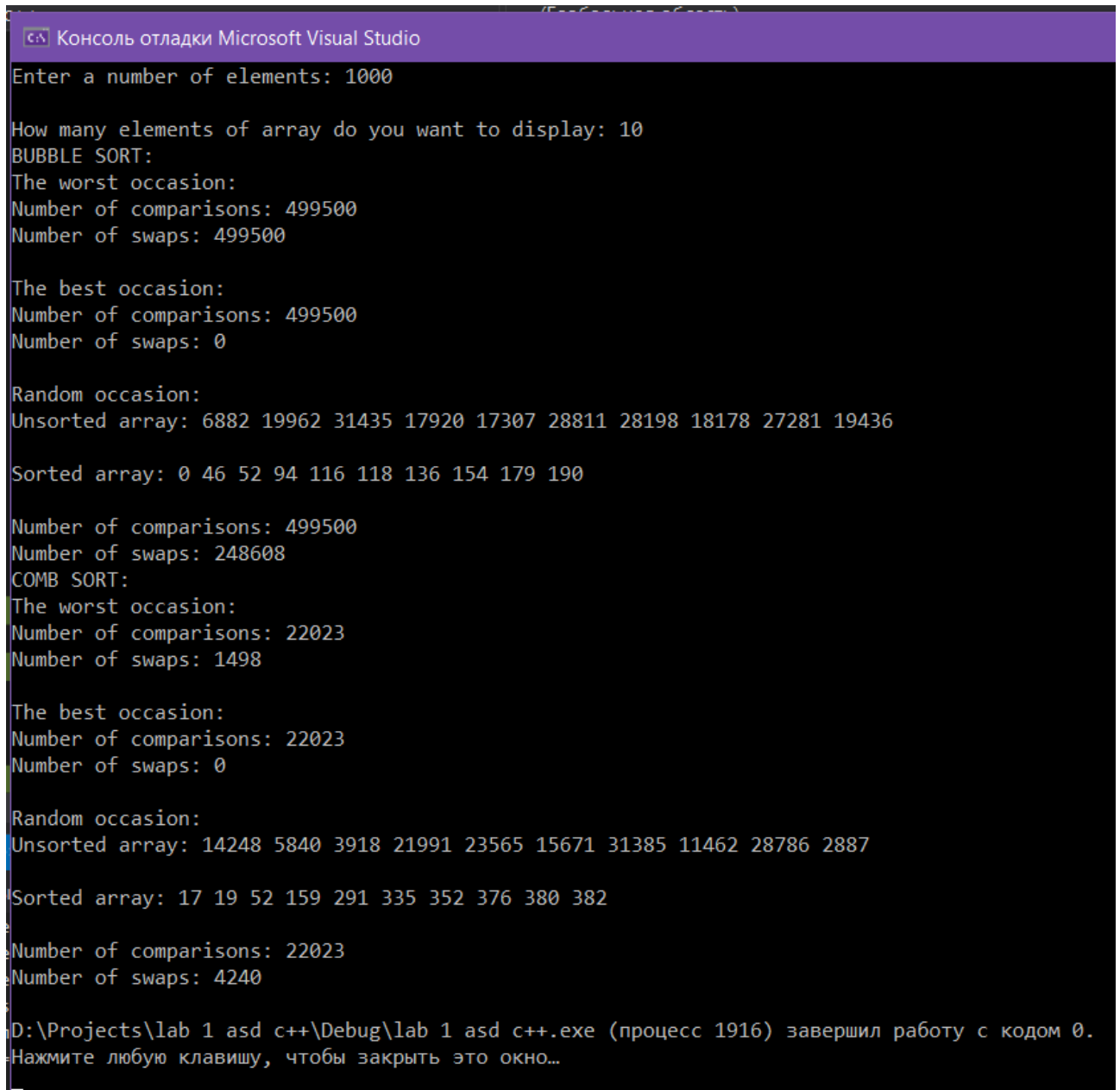
Random occasion:
Unsorted array: 32082 14698 23467 26496 18855 14831 28111 10552 593 25531

Sorted array: 10 593 1285 1982 2917 3284 3340 3408 3696 5169

Number of comparisons: 1233
Number of swaps: 275

D:\Projects\lab 1 asd c++\Debug\lab 1 asd c++.exe (процесс 15040) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно...
```

Рисунок 3.1 – Сортвання масиву на 100 елементів



```
Консоль отладки Microsoft Visual Studio
Enter a number of elements: 1000

How many elements of array do you want to display: 10
BUBBLE SORT:
The worst occasion:
Number of comparisons: 499500
Number of swaps: 499500

The best occasion:
Number of comparisons: 499500
Number of swaps: 0

Random occasion:
Unsorted array: 6882 19962 31435 17920 17307 28811 28198 18178 27281 19436

Sorted array: 0 46 52 94 116 118 136 154 179 190

Number of comparisons: 499500
Number of swaps: 248608
COMB SORT:
The worst occasion:
Number of comparisons: 22023
Number of swaps: 1498

The best occasion:
Number of comparisons: 22023
Number of swaps: 0

Random occasion:
Unsorted array: 14248 5840 3918 21991 23565 15671 31385 11462 28786 2887

Sorted array: 17 19 52 159 291 335 352 376 380 382

Number of comparisons: 22023
Number of swaps: 4240

D:\Projects\lab 1 asd c++\Debug\lab 1 asd c++.exe (процесс 1916) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно...
```

Рисунок 3.2 – Сортвання масиву на 1000 елементів

3.5 Тестування алгоритму

3.5.1 Часові характеристики оцінювання

В таблиці 3.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки для масивів різної розмірності, коли масив містить упорядковану послідовність елементів.

Таблиця 3.2 – Характеристики оцінювання алгоритму сортування бульбашки для упорядкованої послідовності елементів у масиві.

	Бульбашка		Гребінець	
Розмірність масиву	Число порівнянь	Число перестановок	Число порівнянь	Число перестановок
10	45	0	39	0
100	4950	0	1233	0
1000	499500	0	22023	0
5000	12497500	0	144834	0
10000	49995000	0	329606	0
20000	199990000	0	719141	0
50000	1249975000	0	1997682	0

В таблиці 3.3 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки для масивів різної розмірності, коли масиви містять зворотно упорядковану послідовність елементів.

Таблиця 3.3 – Характеристики оцінювання алгоритму сортування бульбашки для зворотно упорядкованої послідовності елементів у масиві.

Розмірність масиву	Бульбашка		Гребінець	
	Число порівнянь	Число перестановок	Число порівнянь	Число перестановок
10	45	45	39	5
100	4950	4950	1233	106
1000	499500	499500	22023	1498
5000	12497500	12497500	144834	9050
10000	49995000	49995000	329606	18994
20000	199990000	199990000	719141	40668
50000	1249975000	1249975000	1997682	110406

У таблиці 3.4 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки для масивів різної розмірності, масиви містять випадкову послідовність елементів.

Таблиця 3.4 – Характеристика оцінювання алгоритму сортування бульбашки для випадкової послідовності елементів у масиві.

Розмірність масиву	Бульбашка		Гребінець	
	Число порівнянь	Число перестановок	Число порівнянь	Число перестановок
10	45	26	39	9
100	4950	2495	1233	260
1000	499500	248265	22023	4205
5000	12497500	6362152	144834	27105
10000	49995000	24840165	329606	59346
20000	199990000	99596605	719141	126444
50000	1249975000	620197639	1997682	345401

3.5.2 Графіки залежності часових характеристик оцінювання від розмірності масиву

На рисунку 3.3 показані графіки залежності часових характеристик оцінювання від розмірності масиву для випадків, коли масиви містять упорядковану послідовність елементів (червоний графік), коли масиви містять зворотно упорядковану послідовність елементів (фіолетовий графік), коли масиви містять випадкову послідовність елементів (зелений графік), також показані асимптотичні оцінки гіршого (чорний графік) і кращого (синій графік) випадків для порівняння.

Сортування бульбашкою:

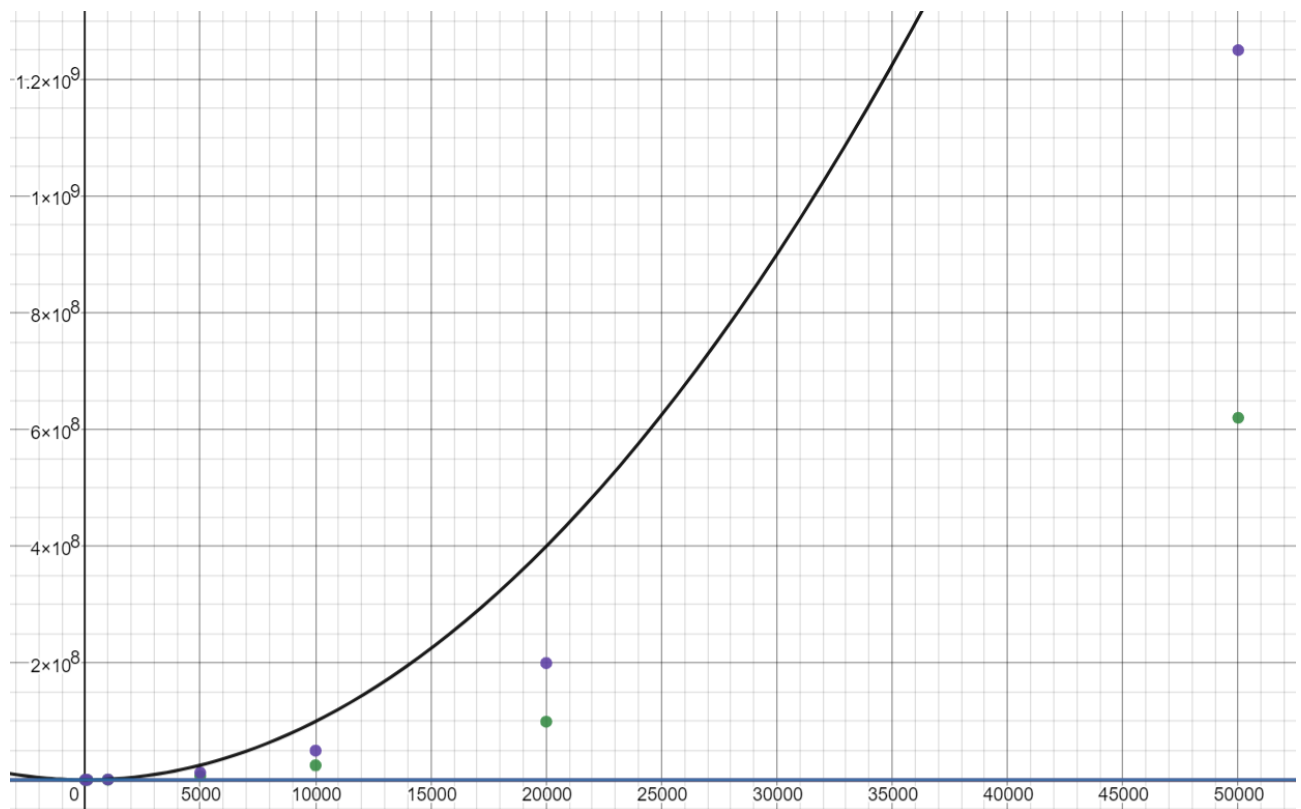


рис 3.3.1

Сортування гребінцем:

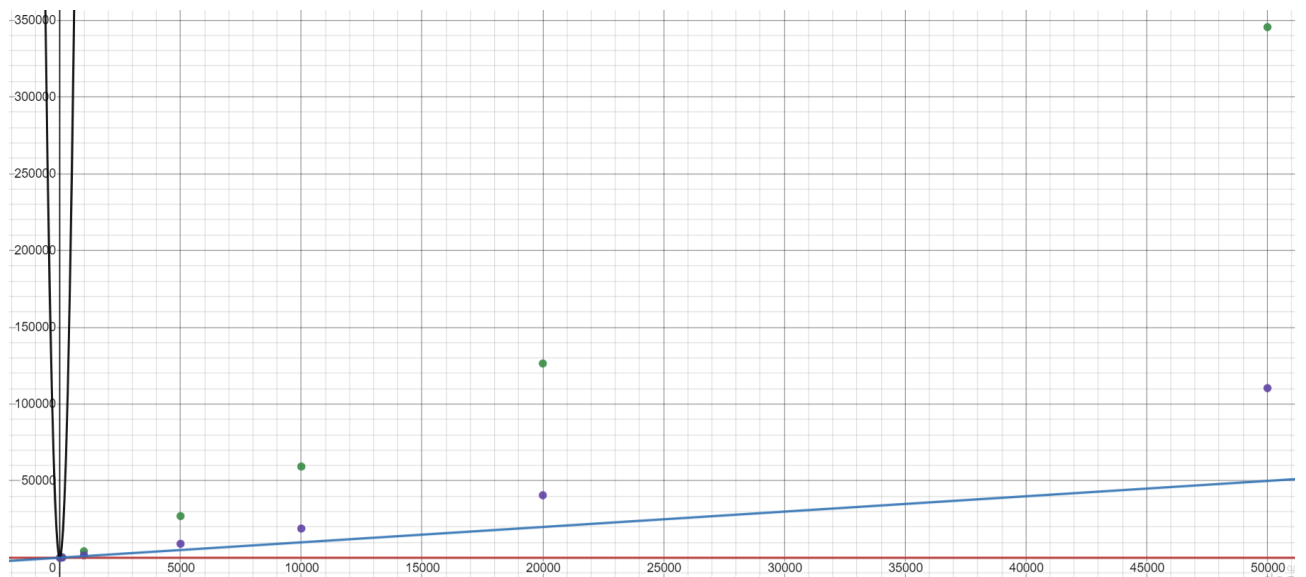


рис 3.3.2

Висновок

При виконанні даної лабораторної роботи я дослідив алгоритми сортування бульбашкою та гребінцем, використав дані алгоритми для сортування масиву під час виконання програми, порівняв їх. В результаті зіставлення результатів роботи можна підсумувати, що сортування гребінцем в усіх випадках набагато ефективніше, так як по суті створювалось як модифікація сортування бульбашкою.

КРИТЕРІЇ ОЦІНЮВАННЯ

У випадку здачі лабораторної роботи до 21.02.2022 включно максимальний бал дорівнює – 5. Після 21.02.2022 – 28.02.2022 максимальний бал дорівнює – 2,5. Після 28.02.2022 робота не приймається

Критерії оцінювання у відсотках від максимального балу:

- аналіз алгоритму на відповідність властивостям – 10%;
- псевдокод алгоритму – 15%;
- аналіз часової складності – 25%;
- програмна реалізація алгоритму – 25%;
- тестування алгоритму – 20%;
- висновок – 5%.