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

Кафедра ІПІ

Звіт

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

„Проектування і аналіз алгоритмів внутрішнього сортування”

Виконав(ла) *ІП-12 Кушнір Ганна Вікторівна* _____
(шифр, прізвище, ім'я, по батькові)

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

Київ 2022

ЗМІСТ

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**	**Сортування гребінцем**
Стійкість	Стійкий	Не стійкий
«Природність» поведінки (Adaptability)	Ні	Так
Базуються на порівняннях	Так	Так
Необхідність в додатковій пам'яті (об'єм)	$\theta(1)$	$\theta(1)$
Необхідність в знаннях про структури даних	Масив	Масив

3.2 Псевдокод алгоритму

3.2.1 Сортування бульбашкою

```
BubbleSort(A):
    для і від 1 до size(A) – 1 включно
        для j від 1 до size(A) – 1 включно
            якщо A[j] > A[j+1]
                то    temp = A[j]
                      A[j] = A[j + 1]
                      A[j + 1] = temp
            кінець якщо
        кінець для
    кінець для
```

3.2.2 Сортування гребінцем

```
CombSort(A):
    step = size(A)
    is_swapped = true
    поки step > 1 або is_swapped == true:
        is_swapped = false
        якщо step / 1.247 > 1
            то    step = округлити(step / 1.247)
            інакше
                step = 1
        все якщо
        для і від 1 до n – step включно
            якщо A[i] > A[i + step]
                то    is_swapped = true
                      temp = A[i]
                      A[i] = A[i + step]
                      A[i + step] = temp
            все якщо
        все для
    все поки
```

3.3 Аналіз часової складності

3.3.1 Сортування бульбашкою

```
BubbleSort(A):                                          // 0
    для і від 1 до size(A) – 1 включно                  // n1 разів
        для j від 1 до size(A) – 1 включно              // n2 разів
            якщо A[j] > A[j+1]                          // c1
                то    temp = A[j]                       // c2
                      A[j] = A[j + 1]                   // c2
                      A[j + 1] = temp                   // c2
            кінець якщо                                 // 0
        кінець для                                      // 0
    кінець для                                          // 0
```

Загальна часова складність (кількість дій):

$T = n_1 * n_2 * c_1 * (3 * c_2) = n_1 * n_2 * c$, де c – константа.

- Найгірший випадок:

  $n_1 = n - 1$, $n_2 = n - 1$

  Тоді $T = O(cn^2) = O(n^2)$

- Найкращий випадок:

  $n_1 = n - 1$, $n_2 = n - 1$

  Тоді $T = \Theta(n^2)$

- Середній випадок:

  $n_1 = n - 1$, $n_2 = n - 1$

  Тоді $T = \Theta(n^2)$

Тобто в кращому, гіршому та середньому випадках складність буде квадратичною.

3.3.2 Сортування гребінцем

```
CombSort(A):                                          // 0
    step = size(A)                                    // c1
    is_swapped = true                                 // c1
    поки step > 1 або is_swapped == true:             // n1 разів
        is_swapped = false                            // c2
        якщо step / 1.247 > 1                         // c3
            то    step = округлити(step / 1.247)      // c4
            інакше                                    // 0
                  step = 1                            // c4
        все якщо                                      // 0
        для і від 1 до n – step включно               // n2 разів
            якщо A[i] > A[i + step]                   // c6
                то    is_swapped = true               // c7
                      temp = A[i]                     // c7
                      A[i] = A[i + step]              // c7
                      A[i + step] = temp              // c7
            все якщо                                  // 0
        все для                                       // 0
    все поки
```

Загальна часова складність (кількість дій):

$T = 2 * c_1 + n_1 * (c_2 + c_3 * c_4 + n_2 * c_6 * 4 * c_7) = n_1 * n_2 * c$, де c – константа.

- Найгірший випадок:

  $T = O(n^2)$

- Найкращий випадок:

  $T = \Theta(nlogn)$

3.4 Програмна реалізація алгоритму

3.4.1 Вихідний код (C++)

3.4.1.1 Сортування бульбашкою

```
void BubbleSort(int arr[], int n)
{
	for (int i = 0; i < n - 1; ++i)
		for (int j = 0; j < n - 1; ++j)
			if (arr[j] > arr[j + 1])
			{
				int temp = arr[j];
				arr[j] = arr[j + 1];
				arr[j + 1] = temp;
			}
}
```

3.4.1.2 Сортування гребінцем

```
void CombSort(long long arr[], int n)
{
	int step = n;
	bool swapped = true;
	while (step > 1 || swapped) {
		swapped = false;
		step = ((float)step/1.247 > 1 ? floor((float)step/1.247) : 1);
		for (int i = 0; i < n - step; i++){
			if (arr[i] > arr[i + step]) {
				swapped = true;
				long temp = arr[i];
				arr[i] = arr[i + step];
				arr[i + step] = temp;
			}
		}
	}
}
```

3.4.2 Приклад роботи

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

3.4.2.1 Сортування бульбашкою

Рисунок 3.1.1 – Сортування масиву на 100 елементів

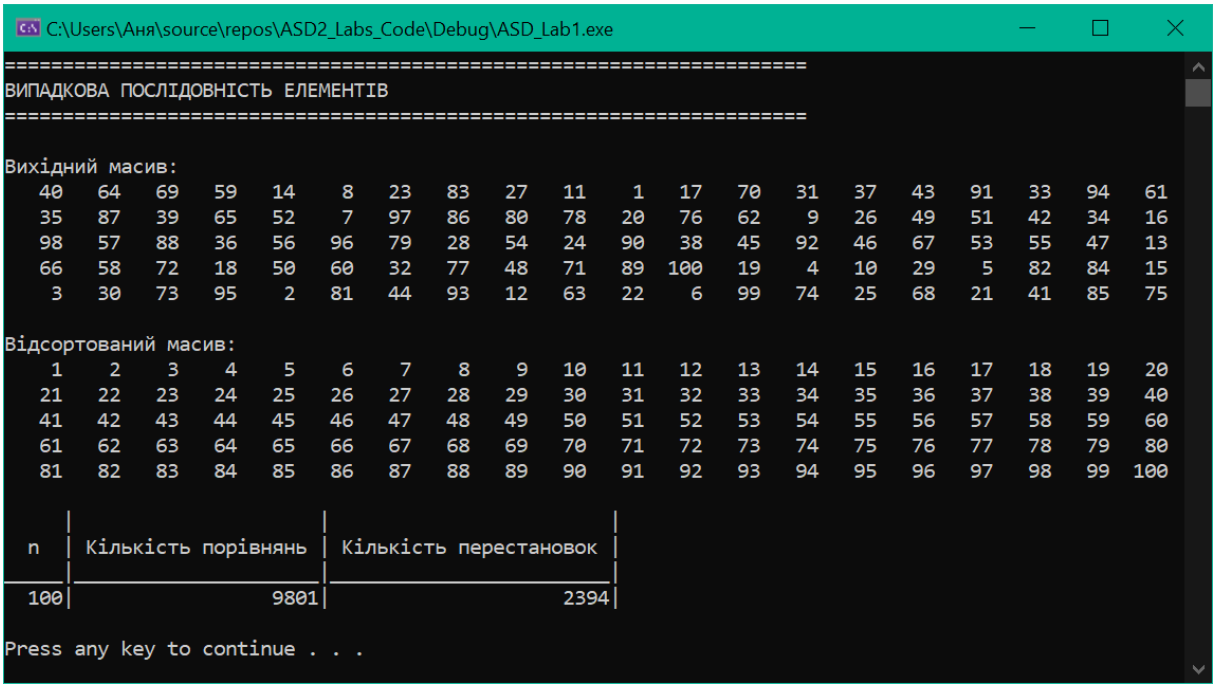
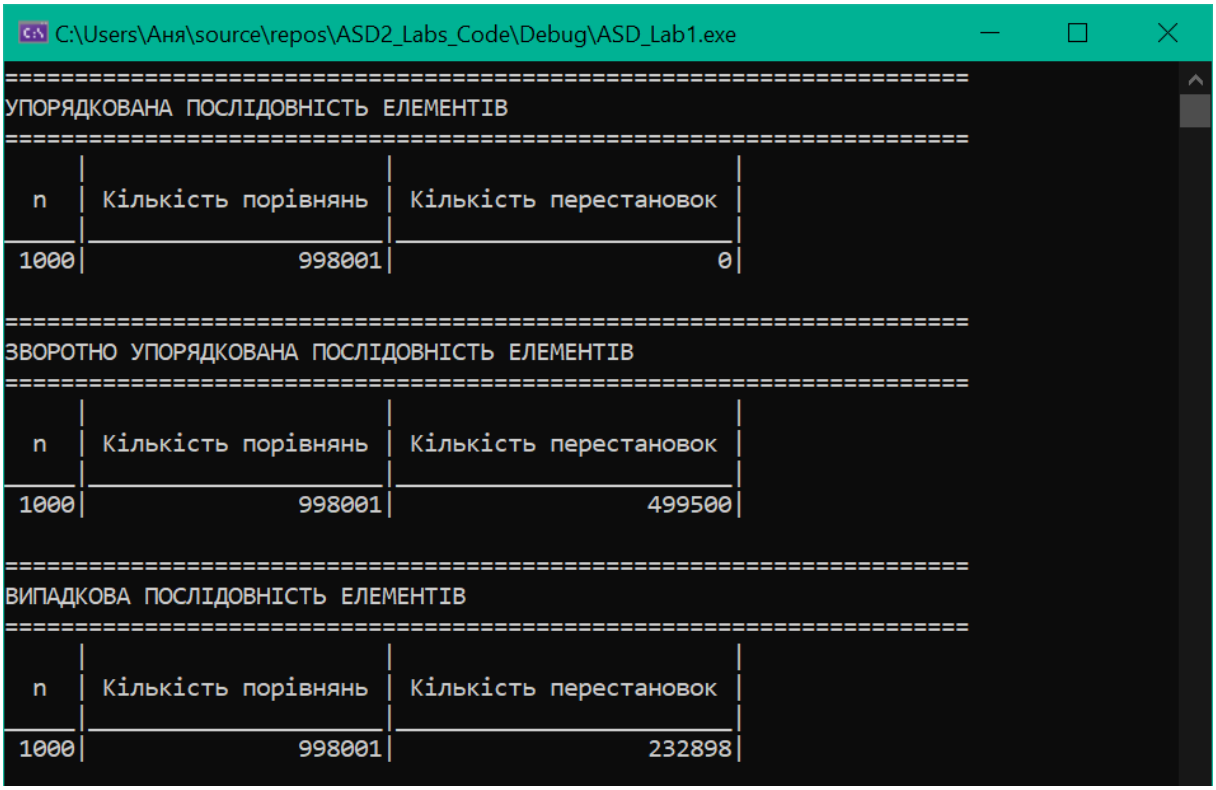


Рисунок 3.2.1 – Сортування масиву на 1000 елементів



3.4.2.2 Сортування гребінцем

Рисунок 3.1.2 – Сортування масиву на 100 елементів

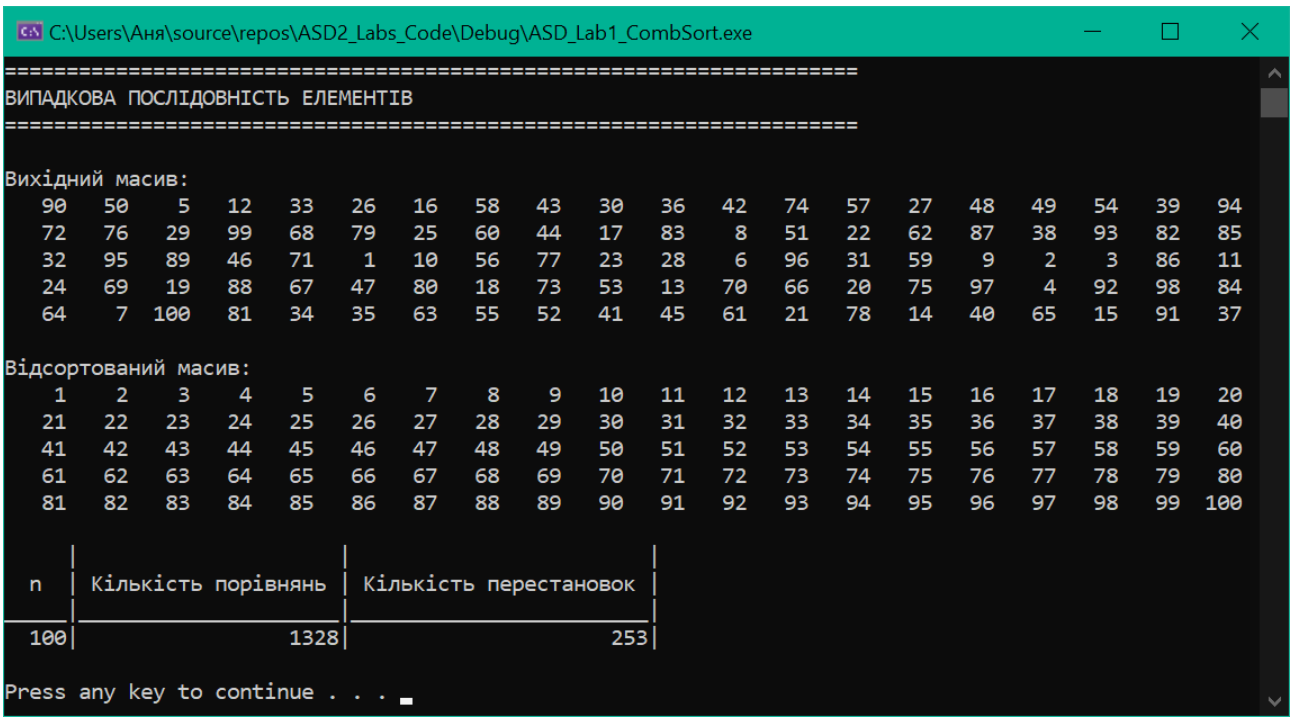
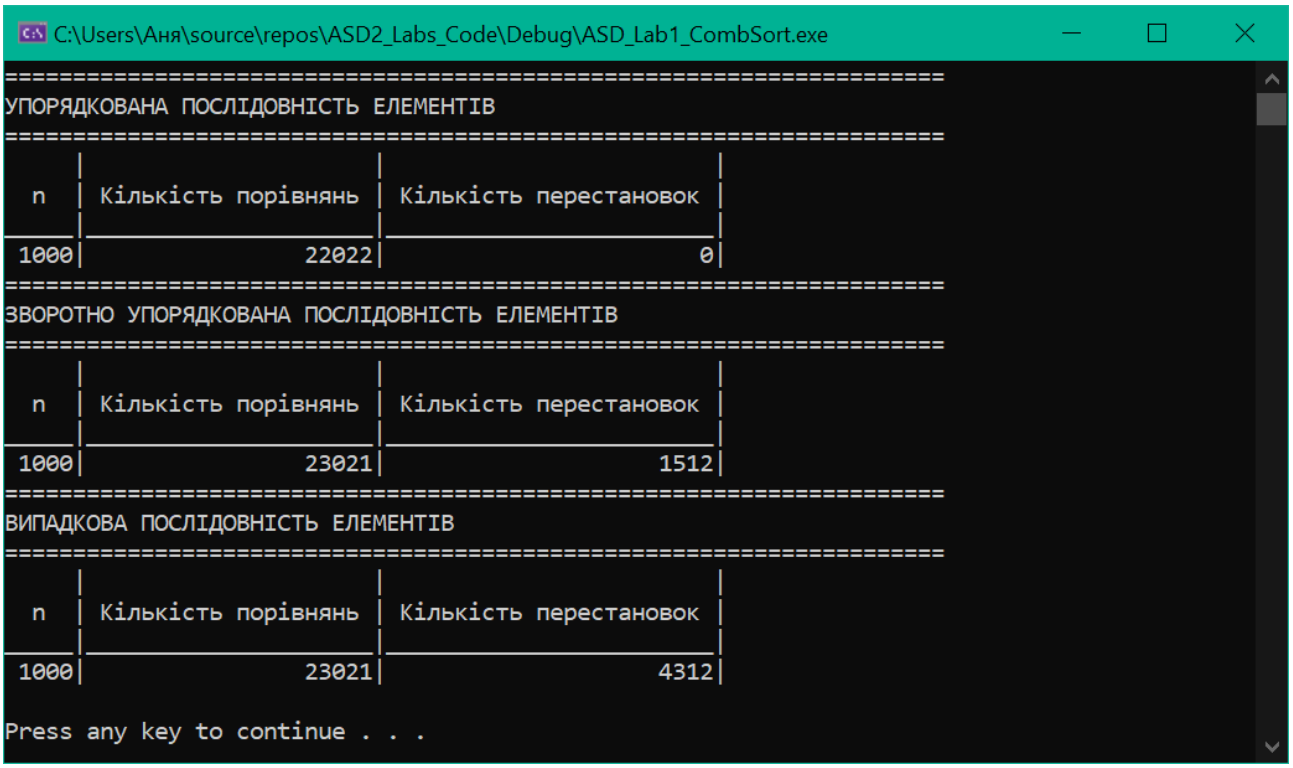


Рисунок 3.2.2 – Сортування масиву на 1000 елементів



3.5 Тестування алгоритму

3.5.1 Часові характеристики оцінювання

3.5.1.1 Сортування бульбашкою

В таблиці 3.2.1 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки для масивів різної розмірності, коли масив містить упорядковану послідовність елементів.

Таблиця 3.2.1 – Характеристики оцінювання алгоритму сортування бульбашки для упорядкованої послідовності елементів у масиві

Розмірність масиву	Число порівнянь	Число перестановок
10	81	0
100	9 801	0
1000	998 001	0
5000	24 990 001	0
10000	99 980 001	0
20000	399 960 001	0
50000	2 499 900 001	0

У таблиці 3.3.1 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки для масивів різної розмірності, коли масиви містять зворотно упорядковану послідовність елементів.

Таблиця 3.3.1 – Характеристики оцінювання алгоритму сортування бульбашки для зворотно упорядкованої послідовності елементів у масиві.

Розмірність масиву	Число порівнянь	Число перестановок
10	81	45
100	9 801	4 950
1000	998 001	499 500
5000	24 990 001	12 497 500
10000	99 980 001	49 995 000
20000	399 960 001	199 990 000
50000	2 499 900 001	1 249 975 000

У таблиці 3.4.1 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки для масивів різної розмірності, масиви містять випадкову послідовність елементів.

Таблиця 3.4.1 – Характеристика оцінювання алгоритму сортування бульбашки для випадкової послідовності елементів у масиві.

Розмірність масиву	Число порівнянь	Число перестановок
10	81	16
100	9 801	2318
1000	998 001	241509
5000	24 990 001	6 031 609
10000	99 980 001	24 147 164
20000	399 960 001	98 409 805
50000	2 499 900 001	671 721 125

3.5.1.2 Сортування гребінцем

В таблиці 3.2.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування гребінцем для масивів різної розмірності, коли масив містить упорядковану послідовність елементів.

Таблиця 3.2.2 – Характеристики оцінювання алгоритму сортування гребінцем для упорядкованої послідовності елементів у масиві

Розмірність масиву	Число порівнянь	Число перестановок
10	36	0
100	1 229	0
1000	22 022	0
5000	144 832	0
10000	329 598	0
20000	719 136	0
50000	1 997 680	0

У таблиці 3.3.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування гребінцем для масивів різної

розмірності, коли масиви містять зворотно упорядковану послідовність елементів.

Таблиця 3.3.2 – Характеристики оцінювання алгоритму сортування гребінцем для зворотно упорядкованої послідовності елементів у масиві.

Розмірність масиву	Число порівнянь	Число перестановок
10	45	9
100	1328	110
1000	23 021	1 512
5000	149 831	9 154
10000	339 597	19 018
20000	739 135	40 730
50000	2 047 679	110 332

У таблиці 3.4.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування гребінцем для масивів різної розмірності, масиви містять випадкову послідовність елементів.

Таблиця 3.4.2 – Характеристика оцінювання алгоритму сортування гребінцем для випадкової послідовності елементів у масиві.

Розмірність масиву	Число порівнянь	Число перестановок
10	45	9
100	1328	231
1000	23021	4 242
5000	154 830	27 312
10000	349 596	59 491
20000	759 134	130 585
50000	2 097 678	368 821

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 (зелений графік), коли масиви містять зворотно упорядковану послідовність елементів (червоний графік), коли масиви містять випадкову послідовність елементів (синій графік), також показані асимптотичні оцінки гіршого (фіолетовий графік) і кращого (жовтий графік) випадків для порівняння.

Рисунок 3.3.1 – Графіки залежності часових характеристик оцінювання для сортування бульбашкою

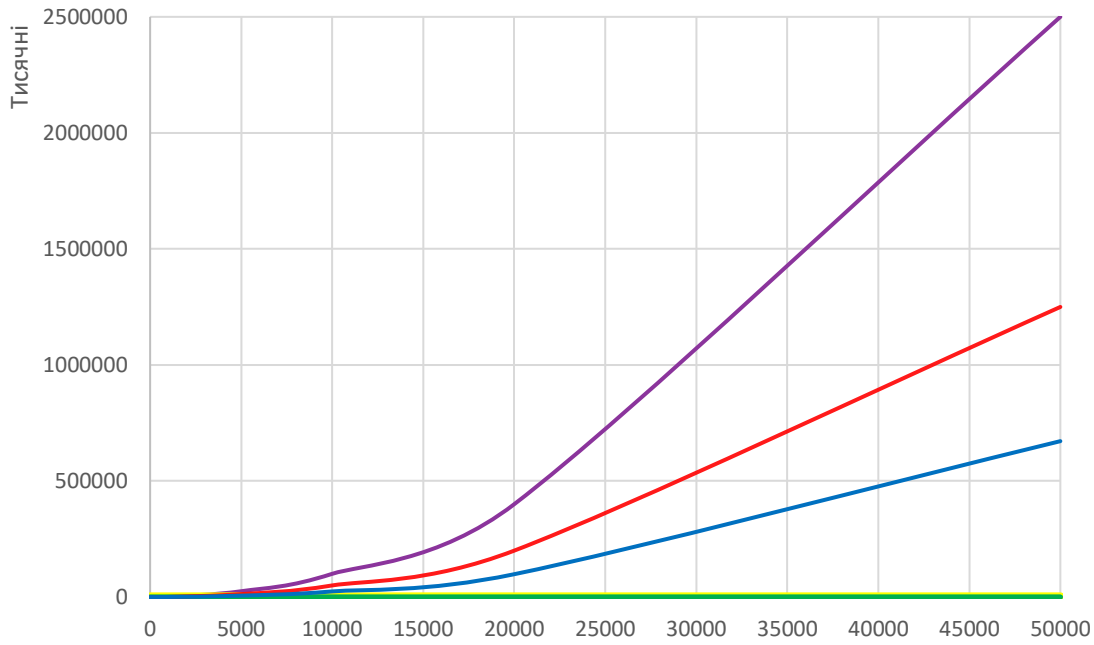
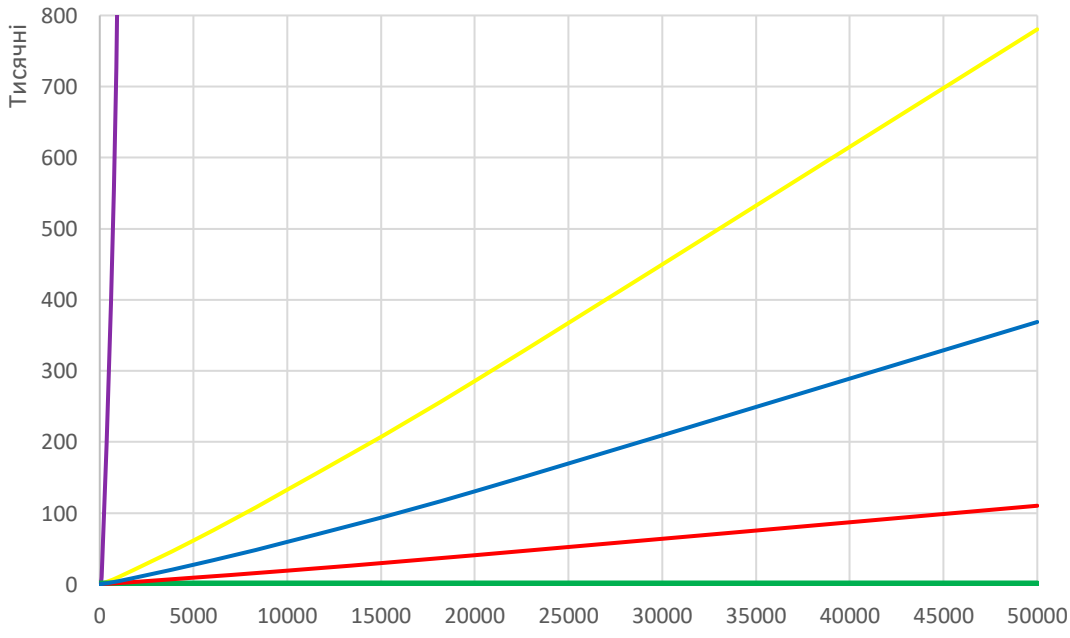


Рисунок 3.3.2 – Графіки залежності часових характеристик оцінювання для сортування гребінцем



ВИСНОВОК

При виконанні даної лабораторної роботи було вивчено основні методи аналізу обчислювальної складності алгоритмів внутрішнього сортування і оцінено поріг їх ефективності.

Було порівняно неоптимізований алгоритм сортування бульбашкою та його оптимізацію – алгоритм сортування гребінцем. У ході порівняння було зроблено висновок, що алгоритм сортування гребінцем набагато ефективніший на великих послідовностях (5000 елементів і більше), ніж його неоптимізована форма (бульбашка).

Кількість порівнянь. У той час, як неоптимізована «бульбашка» здійснює однакову кількість порівнянь на масивах однакової розмірності незалежно від ступеня відсортованості вихідного масиву, «гребінець» здійснює меншу кількість порівнянь на заздалегідь відсортованих масивах, що є ознакою природності його поведінки.

Кількість перестановок. Якщо на вхід подається послідовність з 50000 випадково розташованих елементів, алгоритм сортування бульбашкою сортує його, здійснюючи понад 670 млн перестановок, при цьому алгоритму сортування гребінцем знадобиться близько 360 тис перестановок. Тобто в даному випадку час, витрачений на сортування масиву, є в тисячу разів меншим при сортуванні алгоритмом гребінця.

Маємо, що через дуже велику обчислювальну складність недоцільно використовувати алгоритм сортування бульбашкою при розв’язанні практичних задач у повсякденному житті. Більш раціональним рішенням буде використати його покращення – алгоритм сортування гребінцем.