**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)**

(шифр, прізвище, ім'я, по батькові)

*ІП-13 Шиманська Ганна Артурівна*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов Олексій Олександрович*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc69772242)

[2 ЗаВдання 4](#_Toc69772243)

[3 Виконання 5](#_Toc69772244)

[3.1 Аналіз алгоритму на відповідність властивостям 5](#_Toc69772245)

[3.2 Псевдокод алгоритму 5](#_Toc69772246)

[3.3 Аналіз часової складності 6](#_Toc69772247)

[3.4 Програмна реалізація алгоритму 6](#_Toc69772248)

[3.4.1 Вихідний код 7](#_Toc69772249)

[3.4.2 Приклад роботи 8](#_Toc69772250)

[3.5 Тестування алгоритму 12](#_Toc69772251)

[3.5.1 Часові характеристики оцінювання 12](#_Toc69772252)

[3.5.2 Графіки залежності часових характеристик оцінювання від розмірності масиву 15](#_Toc69772253)

[Висновок 17](#_Toc69772254)

# Мета лабораторної роботи

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

# 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») |

# Виконання

## Аналіз алгоритму на відповідність властивостям

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

Таблиця 3.1 – Аналіз алгоритму на відповідність властивостям

|  |  |  |
| --- | --- | --- |
| **Властивість** | **Сортування бульбашкою** | **Сортування гребінцем** |
| Стійкість | Стійкий | Cтійкий |
| «Природність» поведінки (Adaptability) | Відсутня | Відсутня |
| Базуються на порівняннях | Так | Так |
| Необхідність в додатковій пам'яті (об'єм) | - | - |
| Необхідність в знаннях про структури даних | Наявна | Наявна |

## Псевдокод алгоритму

**Функція Swap(int[] array, int step, int upperBound)**

**для** j **від** 0 **до** array.Length – i **повторити**

**якщо** array[j + step] < array[j]

**то**

(array[j], array[j + step]) = (array[j + step], array[j]);

**все якщо**

**все повторити**

**все функція**

1. **Сортування бульбашкою**

**Функція DoSort(int[] array)**

**для** i **від** 1 **до** array.Length **повторити**

Swap(array, 1, i);

**все повторити**

**все функція**

1. **Сортування гребінцем**

**Функція GetUpdatedGap(int gap)**

gap = (int)(gap / 1.2473)

**якщо** gap < 1

**то**

gap = 1

**все якщо**

**повернути** gap

**Функція DoSort(int[] array)**

gap = array.Length

**поки** gap != 1

gap = GetUpdatedGap(gap)

**S**wap(array, gap, gap);

**все поки**

**все функція**

## Аналіз часової складності

|  |  |  |
| --- | --- | --- |
|  | **Сортування бульбашкою** | **Сортування гребінцем** |
| найгірший випадок | O(n2) | O(n2) |
| середній випадок | Θ(n2) | Θ(n2) |
| найкращий випадок | * Ω(n2) | Ω(n log(n)) |

## Програмна реалізація алгоритму

### Вихідний код

using System;  
using System.Diagnostics;  
  
namespace ArrayOperations  
{  
 public class ArrayGenerator  
 {  
 public int[] GetArray(int length, bool? sorted = null)  
 {  
 int[] array = new int[length];  
 Random rand = new Random();  
 for (int i = 0; i < length; i++)  
 {  
 if (sorted is null)  
 array[i] = rand.Next(-5000, 10001);  
 else if (sorted is true)  
 {  
 array[i] = i;  
 }   
 else  
 {  
 array[i] = length - i;  
 }  
 }  
  
 return array;  
 }  
 }  
 public class BubbleSort  
 {  
 private int \_comparisonAmount;  
 private int \_swapAmount;  
   
 protected void Swap(int[] array, int step, int upperBound)  
 {  
 for (int j = 0; j < array.Length - upperBound; j++)  
 {  
 \_comparisonAmount++;  
 if (array[j + step] < array[j])  
 {  
 (array[j], array[j + step]) = (array[j + step], array[j]);  
 \_swapAmount++;  
 }  
 }  
 }  
  
 public void DisplayCharacteristics()  
 {  
 Console.WriteLine("The amount of comparisons: " + \_comparisonAmount);  
 Console.WriteLine("The amount of swaps: " + \_swapAmount);  
 }  
   
 public void DisplayArray(int[] array)  
 {  
 Console.WriteLine($"\n{this.GetType().Name}");  
 for (int i = 0; i < array.Length; i++)  
 {  
 Console.Write("{0,7}", array[i]);  
 }  
 Console.WriteLine();  
 }  
  
 public double GetSortingTime(int[] array)  
 {  
 Stopwatch sw = Stopwatch.StartNew();  
 DoSort(array);  
 sw.Stop();  
 return sw.Elapsed.TotalMilliseconds;  
 }  
   
 protected virtual void DoSort(int[] array)  
 {  
 for (int i = 1; i <= array.Length - 1; i++)  
 {  
 Swap(array, 1, i);  
 }  
 }  
 }  
  
 public class CombSort : BubbleSort  
 {  
 private int GetUpdatedGap(int gap)  
 {  
 gap = (int)(gap / 1.2473);  
 if (gap < 1) gap = 1;  
 return gap;  
 }  
  
 protected override void DoSort(int[] array)  
 {  
 int gap = array.Length;  
 while (gap != 1)  
 {  
 gap = GetUpdatedGap(gap);  
 Swap(array, gap, gap);  
 }  
 }  
 }  
}

### Приклад роботи

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

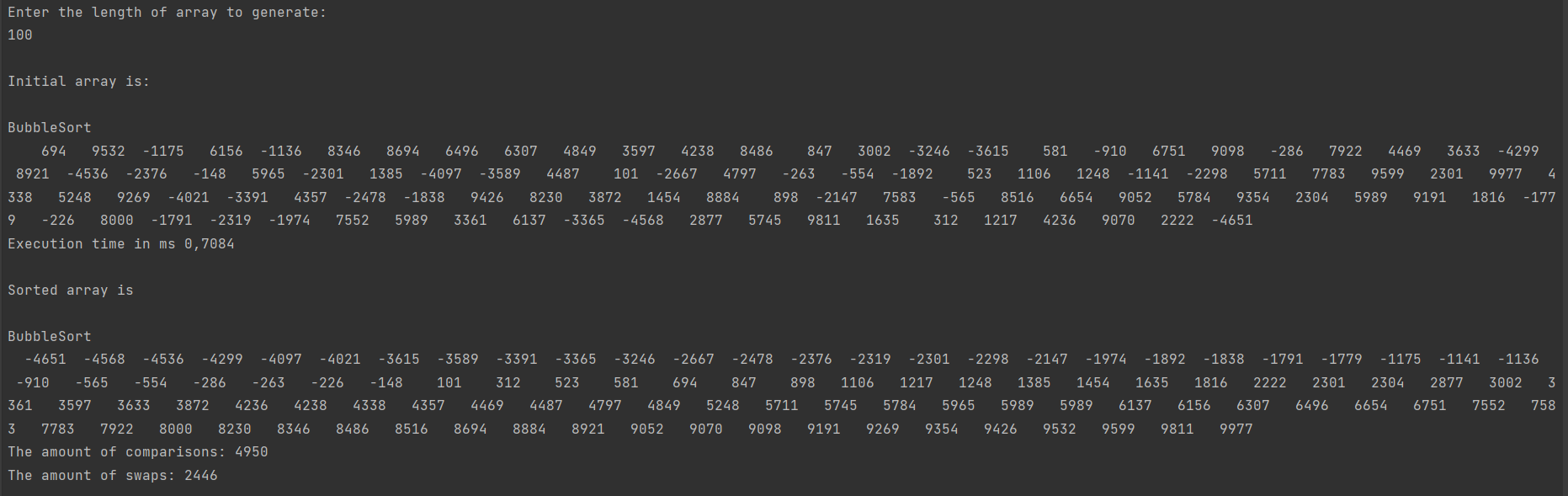


Рисунок 3.1.1 – Сортування масиву на 100 елементів (Bubble Sort)

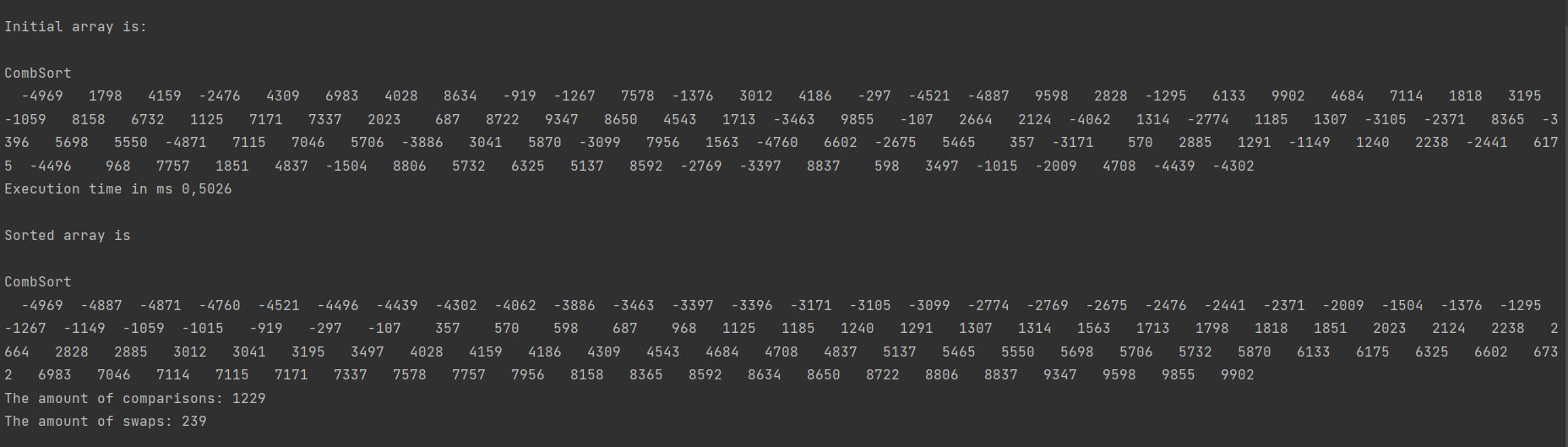
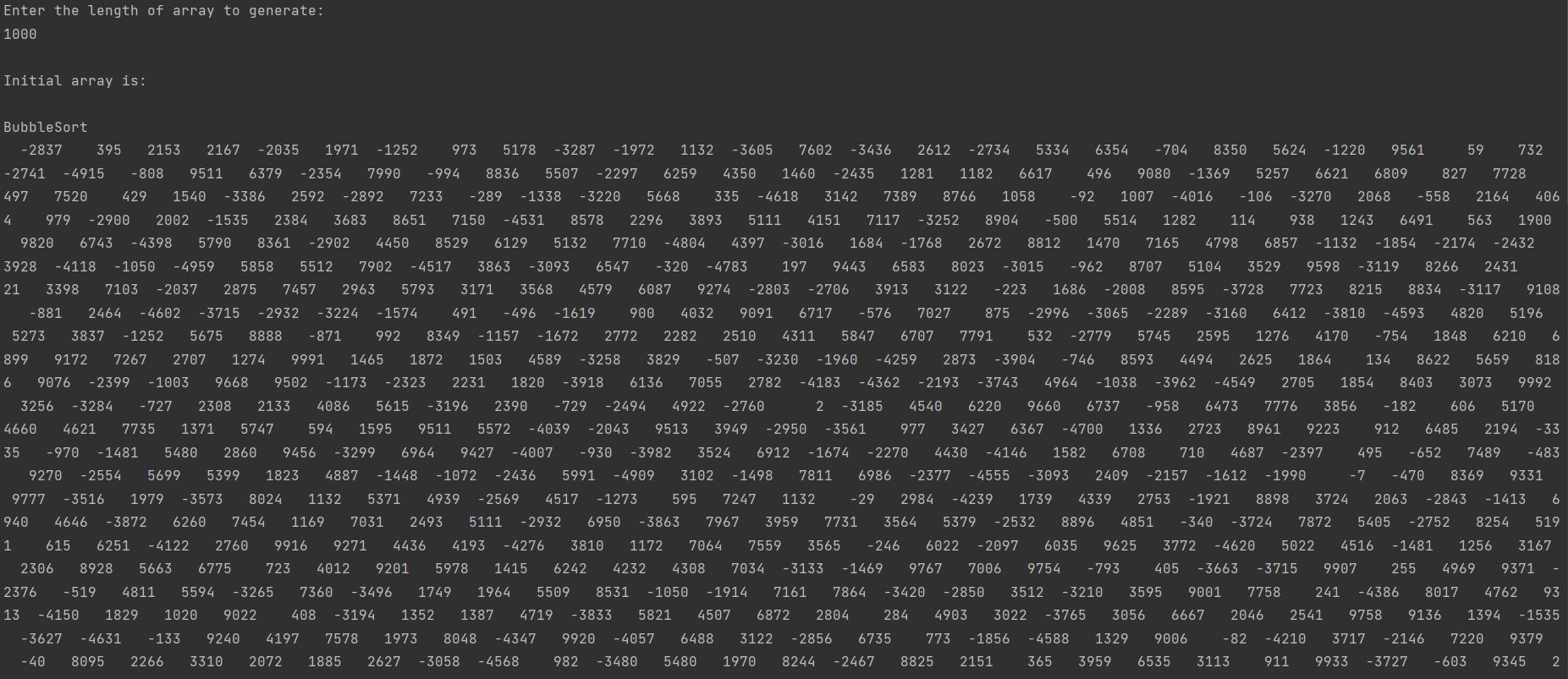
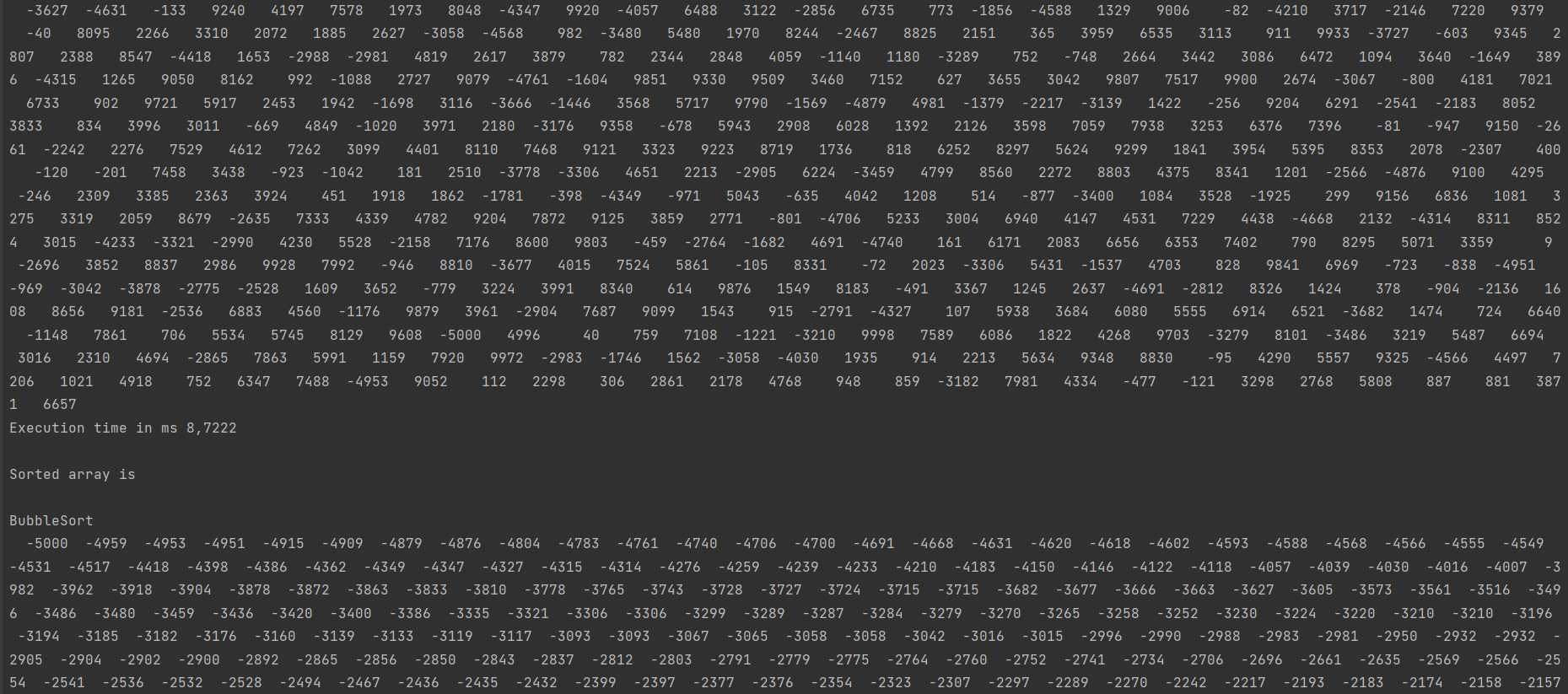


Рисунок 3.1.2 – Сортування масиву на 100 елементів (Comb Sort)





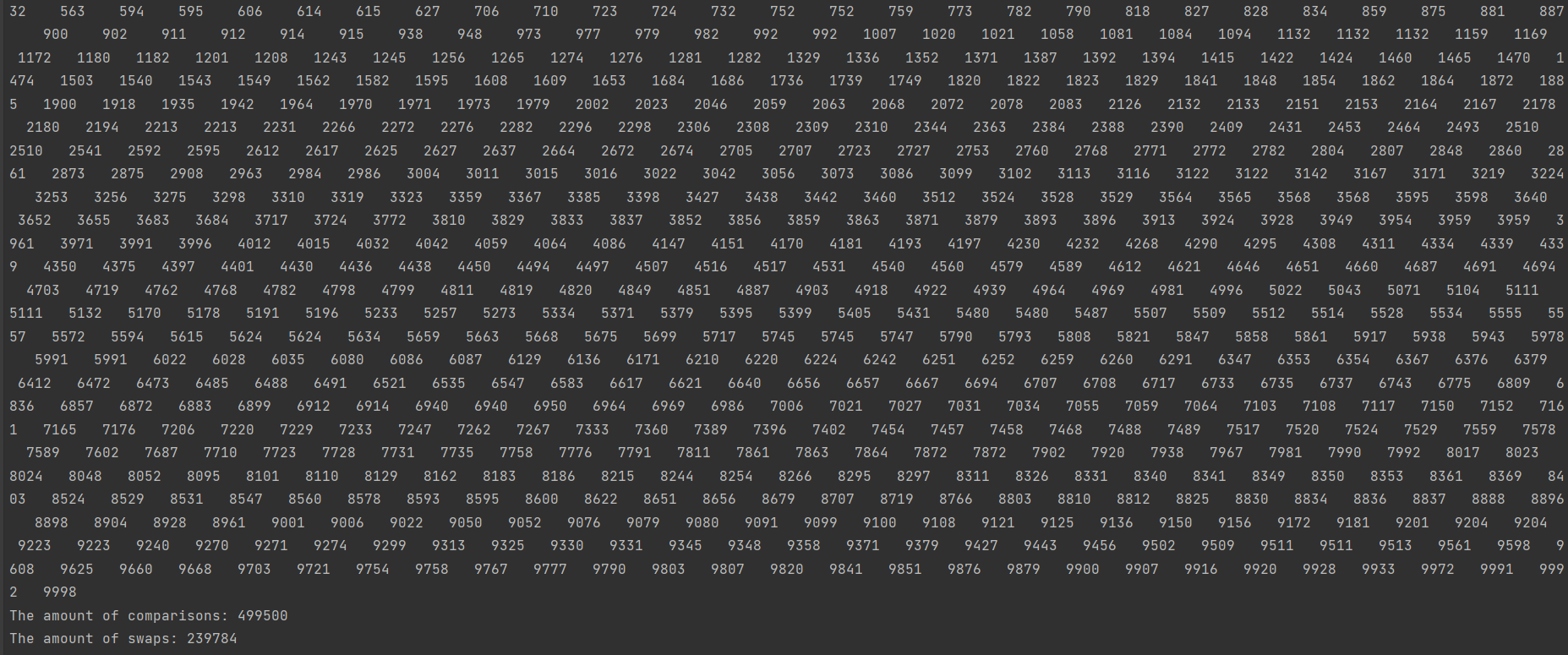
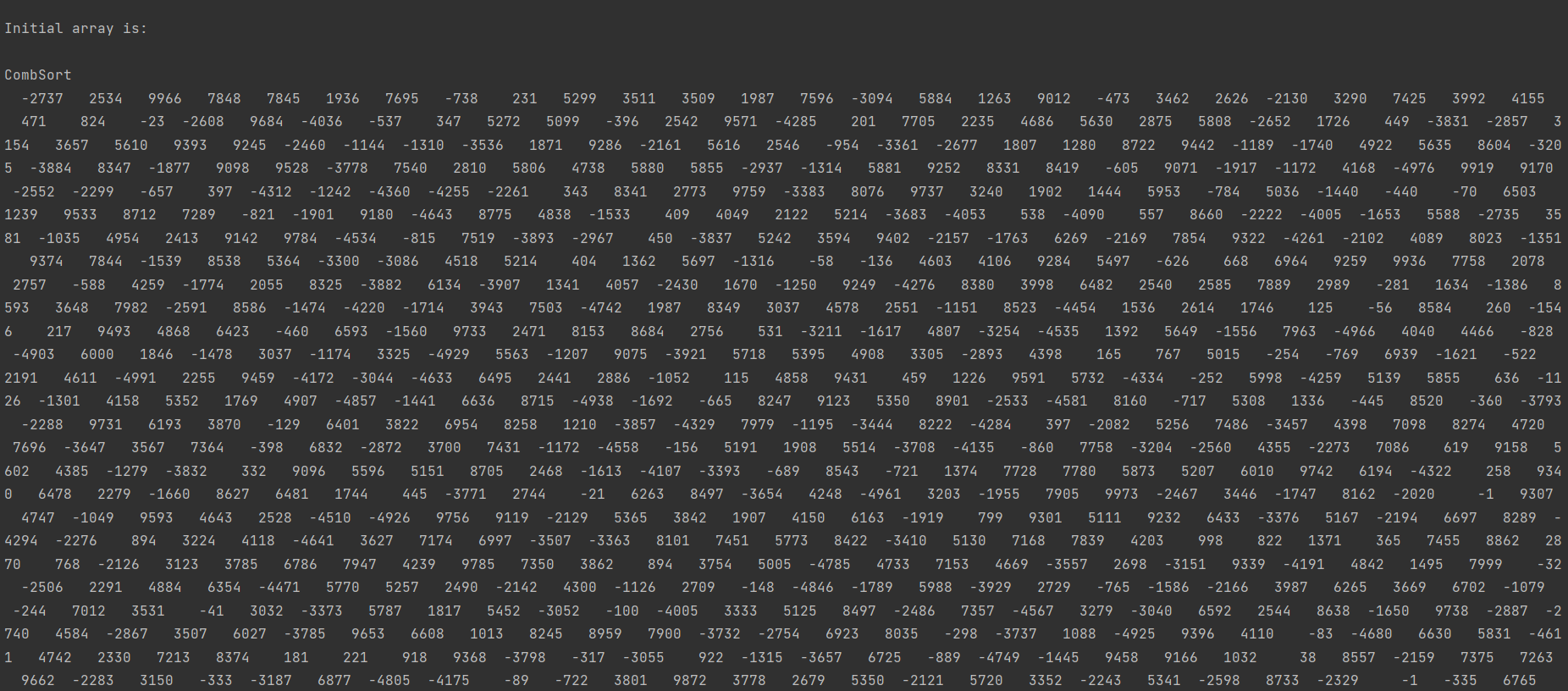
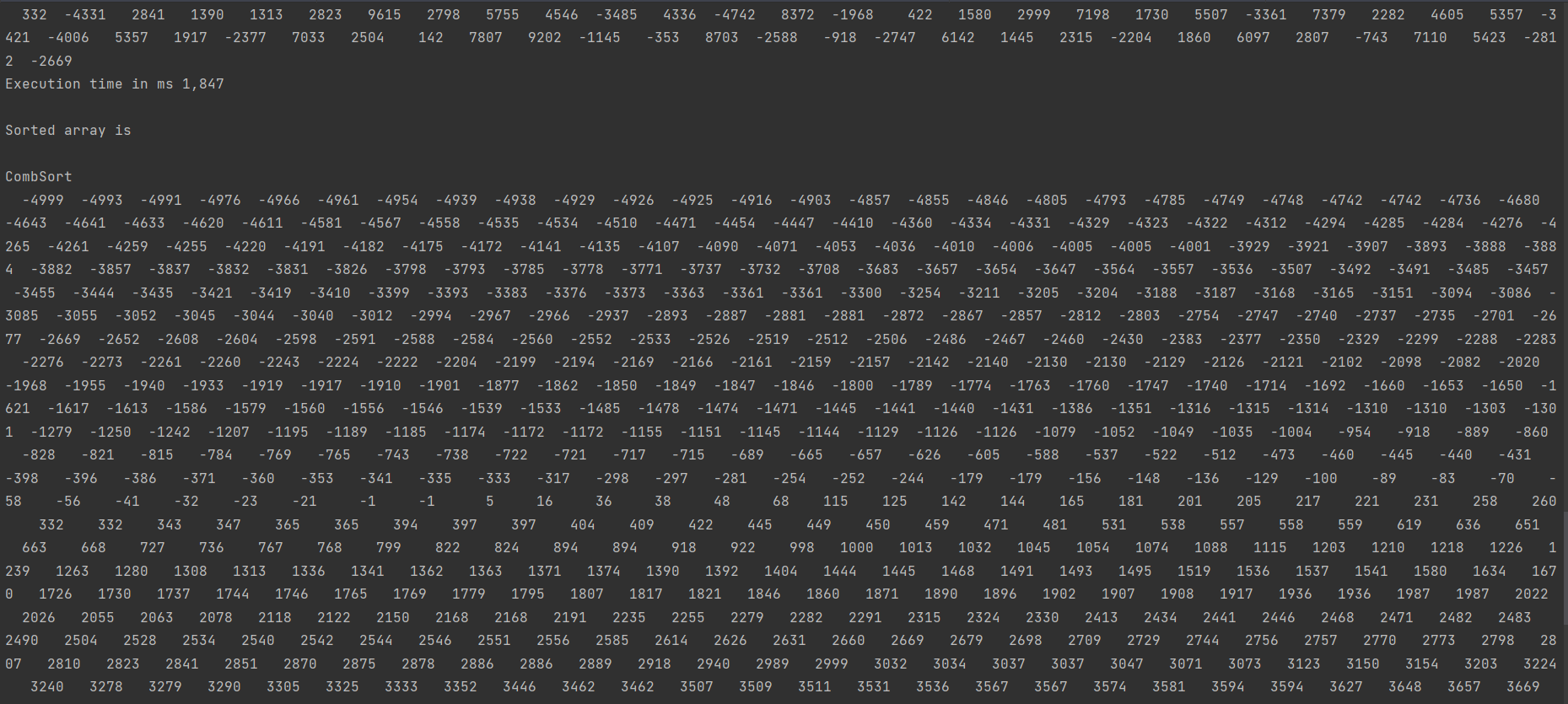


Рисунок 3.2.1 – Сортування масиву на 1000 елементів(Bubble Sort)





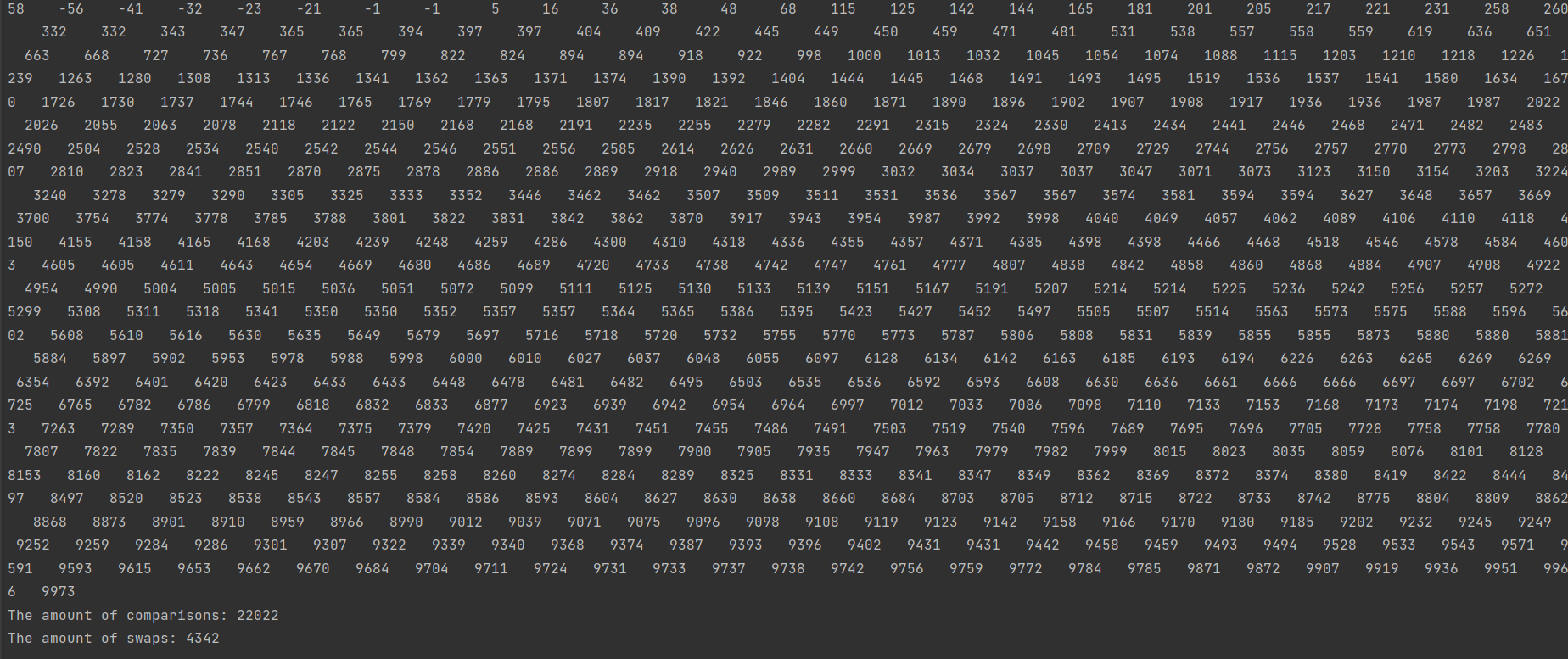


Рисунок 3.2.1 – Сортування масиву на 1000 елементів(Comb Sort)

## Тестування алгоритму

### Часові характеристики оцінювання

В таблицях 3.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки і гребінцем 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 (Bubble Sort) | Число перестановок (Bubble Sort) |
| 10 | 45 | 0 |
| 100 | 4950 | 0 |
| 1000 | 499500 | 0 |
| 5000 | 12497500 | 0 |
| 10000 | 49995000 | 0 |
| 20000 | 199990000 | 0 |
| 50000 | 1249975000 | 0 |

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

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь (Comb Sort) | Число перестановок (Comb Sort) |
| 10 | 36 | 0 |
| 100 | 1229 | 0 |
| 1000 | 22022 | 0 |
| 5000 | 144862 | 0 |
| 10000 | 329644 | 0 |
| 20000 | 719230 | 0 |
| 50000 | 1997920 | 0 |

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

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 45 |
| 100 | 4950 | 4950 |
| 1000 | 499500 | 499500 |
| 5000 | 12497500 | 12497500 |
| 10000 | 49995000 | 49995000 |
| 20000 | 199990000 | 199990000 |
| 50000 | 1249975000 | 1249975000 |

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

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 36 | 9 |
| 100 | 1229 | 110 |
| 1000 | 22022 | 1512 |
| 5000 | 144862 | 9016 |
| 10000 | 329644 | 19132 |
| 20000 | 719230 | 40720 |
| 50000 | 1997920 | 109522 |

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

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

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 14 |
| 100 | 4950 | 2458 |
| 1000 | 499500 | 249247 |
| 5000 | 12497500 | 6272819 |
| 10000 | 49995000 | 25125273 |
| 20000 | 199990000 | 100220872 |
| 50000 | 1249975000 | 625486366 |

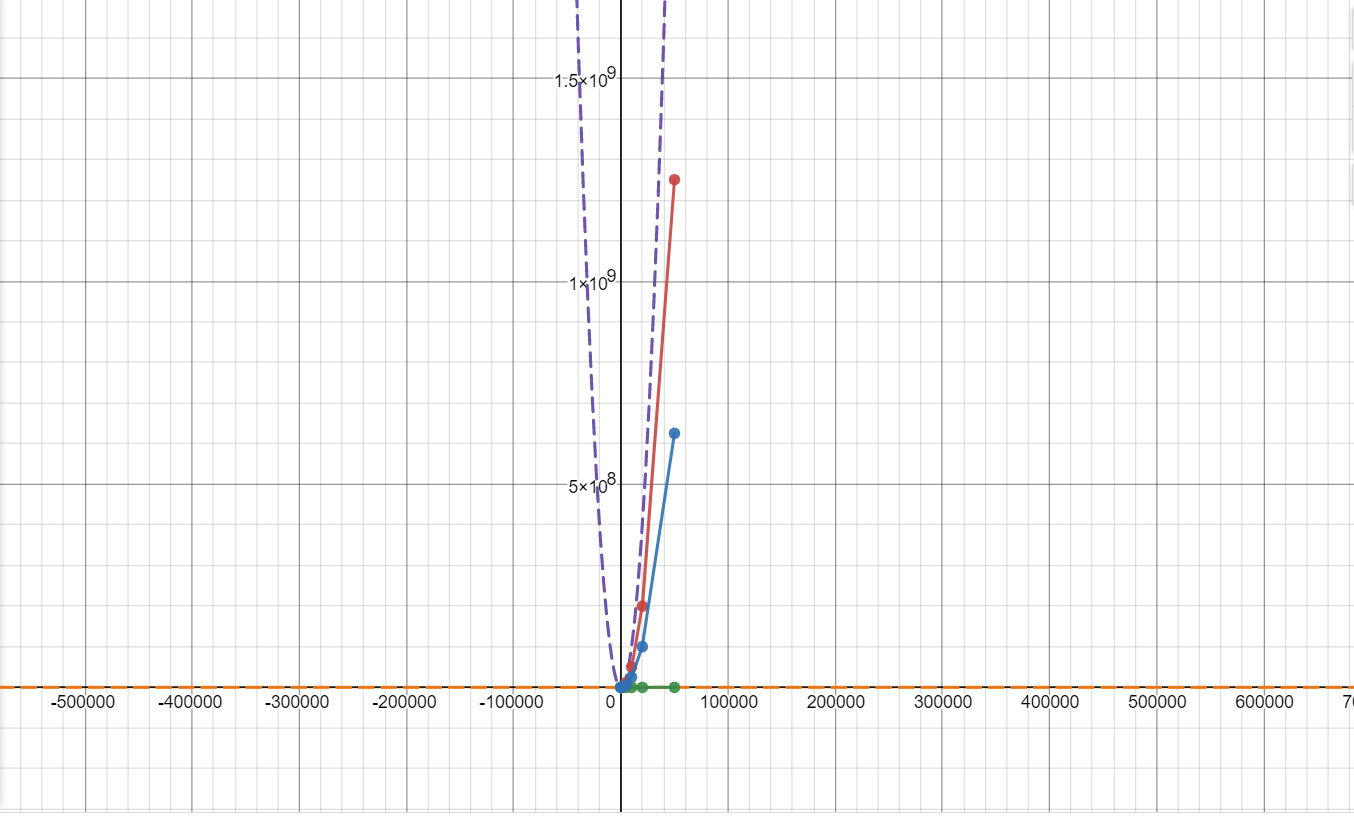
Таблиця 3.4.1 – Характеристика оцінювання алгоритму сортування гребінцем для випадкової послідовності елементів у масиві.

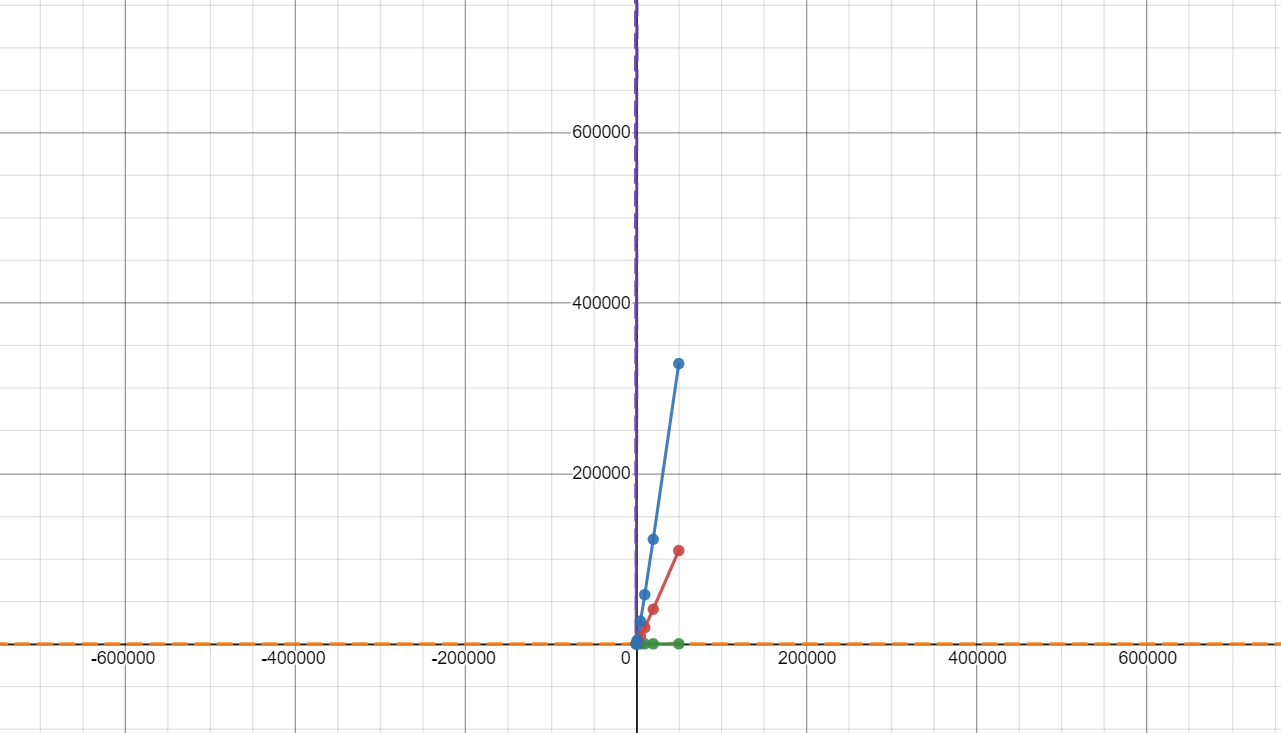
|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 36 | 11 |
| 100 | 1229 | 237 |
| 1000 | 22022 | 4189 |
| 5000 | 144862 | 26799 |
| 10000 | 329644 | 57804 |
| 20000 | 719230 | 122814 |
| 50000 | 1997920 | 329078 |

### 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 – Графіки залежності часових характеристик оцінювання





Висновок

При виконанні даної лабораторної роботи я дослідила алгоритми сортування бульбашкою та гребінцем. Я провела аналіз алгоритмів на відповідність властивостям, проаналізувала часову складність алгоритмів та порівняла часові характеристики.