

**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)**

ІІІ-13 Вдовиченко Станіслав Юрійович  
(шифр, прізвище, ім'я, по батькові)

**Перевірив**

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

Київ 2022

## ЗМІСТ

<b>1</b>	<b>МЕТА ЛАБОРАТОРНОЇ РОБОТИ .....</b>	<b>3</b>
<b>2</b>	<b>ЗАВДАННЯ .....</b>	<b>4</b>
<b>3</b>	<b>ВИКОНАННЯ.....</b>	<b>5</b>
3.1	АНАЛІЗ АЛГОРИТМУ НА ВІДПОВІДНІСТЬ ВЛАСТИВОСТЯМ .....	5
3.2	ПСЕВДОКОД АЛГОРИТМУ .....	6
3.3	АНАЛІЗ ЧАСОВОЇ СКЛАДНОСТІ.....	7
3.4	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ .....	8
3.4.1	<i>Вихідний код.....</i>	<i>8</i>
3.4.2	<i>Приклад роботи .....</i>	<i>9</i>
3.5	ТЕСТУВАННЯ АЛГОРИТМУ .....	11
3.5.1	<i>Часові характеристики оцінювання.....</i>	<i>11</i>
3.5.2	<i>Графіки залежності часових характеристик оцінювання від розмірності масиву .....</i>	<i>13</i>
	<b>ВИСНОВОК .....</b>	<b>17</b>
	<b>КРИТЕРІЇ ОЦІНЮВАННЯ .....</b>	<b>18</b>

## 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 та гребінцем на відповідність властивостям наведено в таблиці 3.1.

Таблиця 3.1 – Аналіз алгоритму на відповідність властивостям

Властивість	Сортування бульбашкою	Сортування гребінцем
Стійкість	+	-
«Природність» поведінки (Adaptability)	-	-
Базуються на порівняннях	+	+
Необхідність в додатковій пам'яті (об'єм)	$O(1)$	$O(1)$
Необхідність в знаннях про структури даних	+	+

### 3.2 Псевдокод алгоритму

Сортування бульбашкою:

bubbleSort(array):

    n = length(array)

**do**

        swapped = false

**for** i = 1 to n – 1 **do**

**if** array[i] > array[i+1] **then**

                temp = array[i]

                array[i] = array[i+1]

                array[i+1] = temp

                swapped = true

**end if**

**end for**

**while** swapped == true

Сортування гребінцем:

combSort(array):

gap = length(array)

shrink = 1.3

sorted = false

**while** sorted == false **do**

gap = floor(gap/shrink)

**if** gap <= 1 **then**

gap = 1

sorted = true

**end if**

i = 0

**while** i + gap < length(array) **do**

**if** array[i] > array[i+gap] **then**

temp = array[i]

array[i] = array[i+gap]

array[i+gap] = temp

sorted = false

**end if**

i = i + 1

**end while**

**end while**

### 3.3 Аналіз часової складності

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

## 3.4 Програмна реалізація алгоритму

### 3.4.1 Вихідний код

main.py

```
from module import *

array = createArray()
comparisons = 0
swaps = 0
print(f"Generated array: {array}")
mode = input("Enter 'b' if you want to use bubble sort\n"+"Enter 'c' if you want to use comb sort\n")
if mode == 'b':
    comparisons, swaps = bubbleSort(array)
elif mode == 'c':
    comparisons, swaps = combSort(array)
print(f"Sorted array: {array}")
print(f"Number of comparisons: {comparisons}\n" + f"Number of swaps: {swaps}")
```

module.py

```
from random import randint

def createArray():
    array = []
    num = int(input("Enter number of elements in array: "))
    key = input("Enter 'a' to generate an average array,\n" + "'b' to generate the best,\n" + "'w' to generate the worst.\n")
    if key == 'a':
        array = [randint(0, num) for i in range(num)]
    elif key == 'w':
        array = [num - 1 - i for i in range(num)]
    elif key == 'b':
        array = [i for i in range(num)]
    return array

def bubbleSort(array):
    n = len(array)
    comps = 0
    swaps = 0
    swapped = True
    while swapped:
        swapped = False
        for i in range(n - 1):
            comps += 1
            if array[i] > array[i + 1]:
                swaps += 1
                array[i], array[i + 1] = array[i + 1], array[i]
                swapped = True
    return comps, swaps

def combSort(array):
    size = len(array)
    gap = size
```



```

shrink = 1.3
is_sorted = False
comps = 0
swaps = 0
while not is_sorted:
    gap = int(gap//shrink)
    if gap <= 1:
        gap = 1
        is_sorted = True
    for i in range(size - gap):
        comps += 1
        if array[i] > array[i+gap]:
            swaps += 1
            array[i], array[i+gap] = array[i+gap], array[i]
        is_sorted = False
return comps, swaps

```

### 3.4.2 Приклад роботи

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

Сортування бульбашкою.

```

Run: C:\Users\Stas\PycharmProjects\ASD1\venv\Scripts\python.exe C:/Users/Stas/PycharmProjects/ASD1/main.py
Enter number of elements in array: 100
Enter 'a' to generate an average array,
'b' to generate the best,
'w' to generate the worst.
Generated array: [68, 85, 8, 3, 10, 7, 12, 86, 68, 24, 17, 100, 37, 21, 94, 45, 73, 12, 3, 7, 11, 100, 11, 66, 2, 54, 89, 98, 9, 69, 29, 3, 50, 57, 31, 8, 29, 83, 3, 95, 3, 40, 92, 87, 83, 58, 54, 29]
Enter 'b' if you want to use bubble sort
Enter 'c' if you want to use comb sort
Sorted array: [1, 2, 3, 3, 3, 3, 3, 6, 7, 7, 8, 8, 8, 8, 9, 10, 10, 10, 11, 11, 12, 12, 12, 15, 15, 17, 17, 17, 21, 23, 23, 24, 24, 24, 26, 29, 29, 29, 31, 34, 37, 37, 40, 40, 41, 42, 42, 44, 45, 45]
Number of comparisons: 7722
Number of swaps: 2225
Process finished with exit code 0

```

Сортування гребінцем.

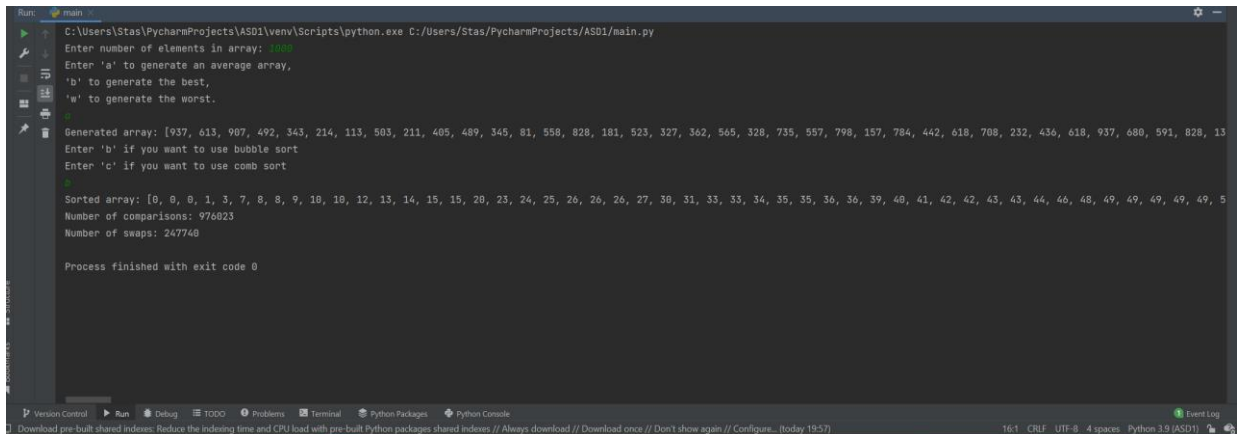
```

Run: C:\Users\Stas\PycharmProjects\ASD1\venv\Scripts\python.exe C:/Users/Stas/PycharmProjects/ASD1/main.py
Enter number of elements in array: 1000
Enter 'a' to generate an average array,
'b' to generate the best,
'w' to generate the worst.
Generated array: [57, 48, 84, 75, 10, 53, 58, 2, 60, 45, 43, 77, 50, 37, 0, 47, 59, 79, 95, 86, 55, 15, 49, 56, 30, 5, 10, 84, 95, 76, 80, 38, 63, 97, 96, 66, 46, 82, 37, 9, 20, 47, 80, 100, 40, 61]
Enter 'b' if you want to use bubble sort
Enter 'c' if you want to use comb sort
Sorted array: [0, 0, 1, 2, 2, 5, 5, 6, 7, 9, 10, 10, 12, 13, 14, 14, 15, 15, 15, 15, 17, 17, 18, 20, 22, 27, 27, 27, 30, 32, 35, 37, 37, 38, 39, 40, 43, 43, 45, 46, 47, 47, 48, 48, 49, 49, 49, 50, 50]
Number of comparisons: 1201
Number of swaps: 249
Process finished with exit code 0

```

## Рисунок 3.1 – Сортуння масиву на 100 елементів

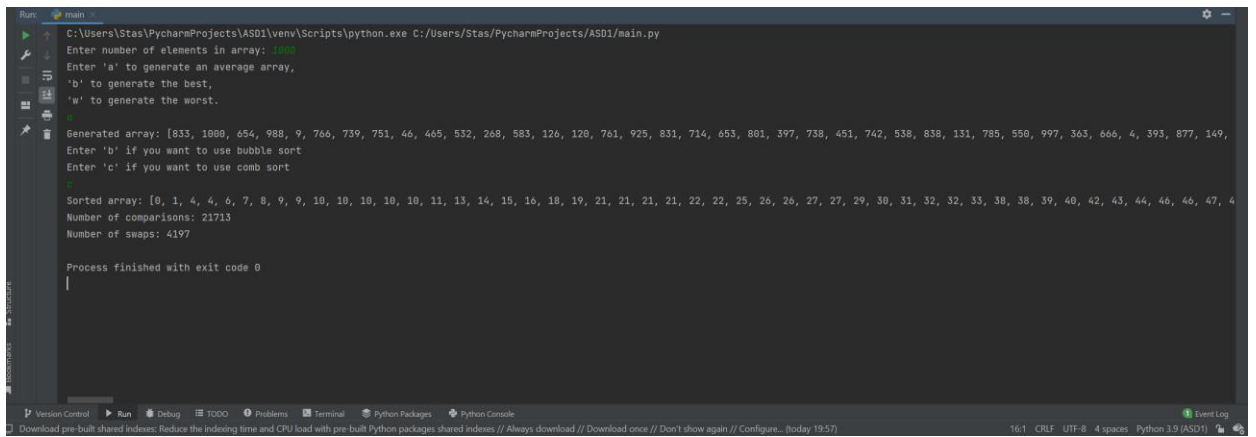
Сортуння бульбашкою.



```
Run: main
C:\Users\Stas\PycharmProjects\ASD1\venv\Scripts\python.exe C:/Users/Stas/PycharmProjects/ASD1/main.py
Enter number of elements in array: 1000
Enter 'a' to generate an average array,
'b' to generate the best,
'w' to generate the worst.
a
Generated array: [937, 613, 907, 492, 343, 214, 113, 503, 211, 405, 489, 345, 81, 558, 828, 181, 523, 327, 362, 565, 328, 735, 557, 798, 157, 784, 442, 618, 788, 232, 436, 618, 937, 680, 591, 828, 13, 5]
Enter 'b' if you want to use bubble sort
Enter 'c' if you want to use comb sort
b
Sorted array: [0, 0, 0, 0, 1, 3, 7, 8, 8, 9, 10, 10, 12, 13, 14, 15, 15, 20, 23, 24, 25, 26, 26, 26, 27, 30, 31, 33, 33, 34, 35, 35, 36, 36, 39, 40, 41, 42, 42, 43, 43, 44, 46, 48, 49, 49, 49, 49, 5]
Number of comparisons: 976023
Number of swaps: 247748

Process finished with exit code 0
```

Сортуння гребінцем.



```
Run: main
C:\Users\Stas\PycharmProjects\ASD1\venv\Scripts\python.exe C:/Users/Stas/PycharmProjects/ASD1/main.py
Enter number of elements in array: 1000
Enter 'a' to generate an average array,
'b' to generate the best,
'w' to generate the worst.
a
Generated array: [833, 1080, 654, 988, 9, 766, 739, 751, 46, 465, 532, 268, 583, 126, 120, 761, 925, 831, 714, 653, 801, 397, 738, 451, 742, 538, 838, 131, 785, 550, 997, 363, 666, 4, 393, 877, 149, 4]
Enter 'b' if you want to use bubble sort
Enter 'c' if you want to use comb sort
c
Sorted array: [0, 1, 4, 4, 6, 7, 8, 9, 9, 10, 10, 10, 10, 10, 11, 13, 14, 15, 16, 18, 19, 21, 21, 21, 21, 22, 22, 25, 26, 26, 27, 27, 29, 30, 31, 32, 32, 33, 38, 38, 39, 40, 42, 43, 44, 46, 46, 47, 4]
Number of comparisons: 21713
Number of swaps: 4197

Process finished with exit code 0
```

## Рисунок 3.2 – Сортуння 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 та гребінця для масивів різної розмірності, коли масив містить упорядковану послідовність елементів.

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

#### Сортування бульбашкою.

Розмірність масиву	Число порівнянь	Число перестановок
10	9	0
100	99	0
1000	999	0
5000	4999	0
10000	9999	0
20000	19999	0
50000	49999	0

#### Сортування гребінцем.

Розмірність масиву	Число порівнянь	Число перестановок
10	32	0
100	1003	0
1000	18716	0
5000	123395	0
10000	276745	0
20000	593412	0
50000	1683424	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.

#### Сортування бульбашкою.

Розмірність масиву	Число порівнянь	Число перестановок
10	90	45
100	9900	4950
1000	999000	499500
5000	24995000	12497500
10000	99990000	49995000
20000	399980000	199990000
50000	2499950000	1249975000

#### Сортування гребінцем.

Розмірність масиву	Число порівнянь	Число перестановок
10	41	7
100	1102	122
1000	19715	1572
5000	128394	9552
10000	286744	20050
20000	613411	43762
50000	1733423	115908

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.

### Сортування бульбашкою.

Розмірність масиву	Число порівнянь	Число перестановок
10	81	22
100	8712	2489
1000	950049	249354
5000	24600079	6228803
10000	97690230	24906876
20000	398280085	99584564
50000	2486800263	626251325

### Сортування гребінцем.

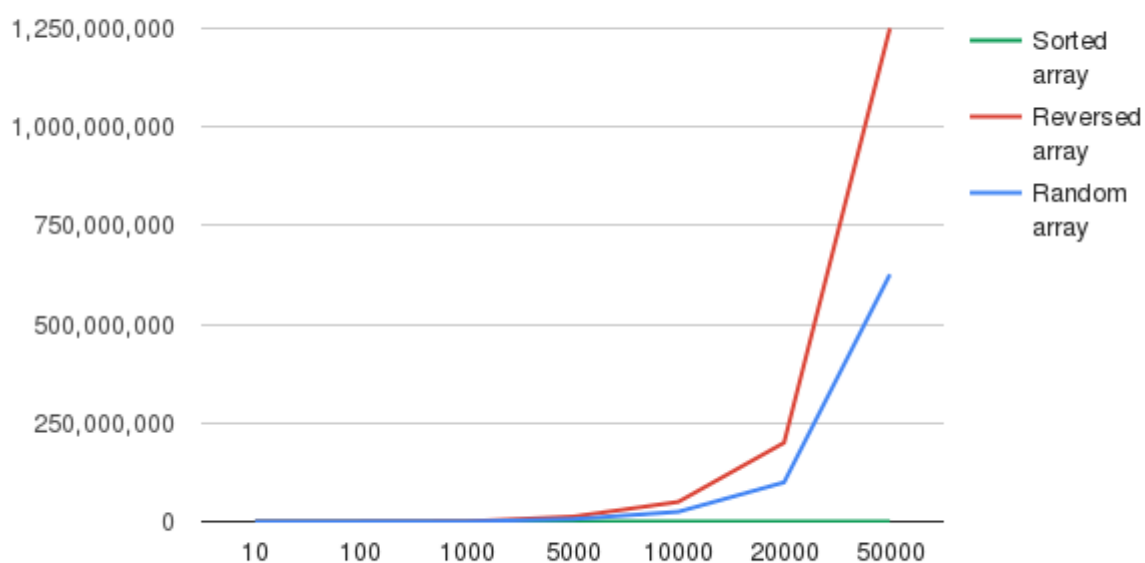
Розмірність масиву	Число порівнянь	Число перестановок
10	41	8
100	1201	213
1000	20714	4114
5000	143391	27331
10000	336739	60944
20000	713406	134016
50000	2033417	368167

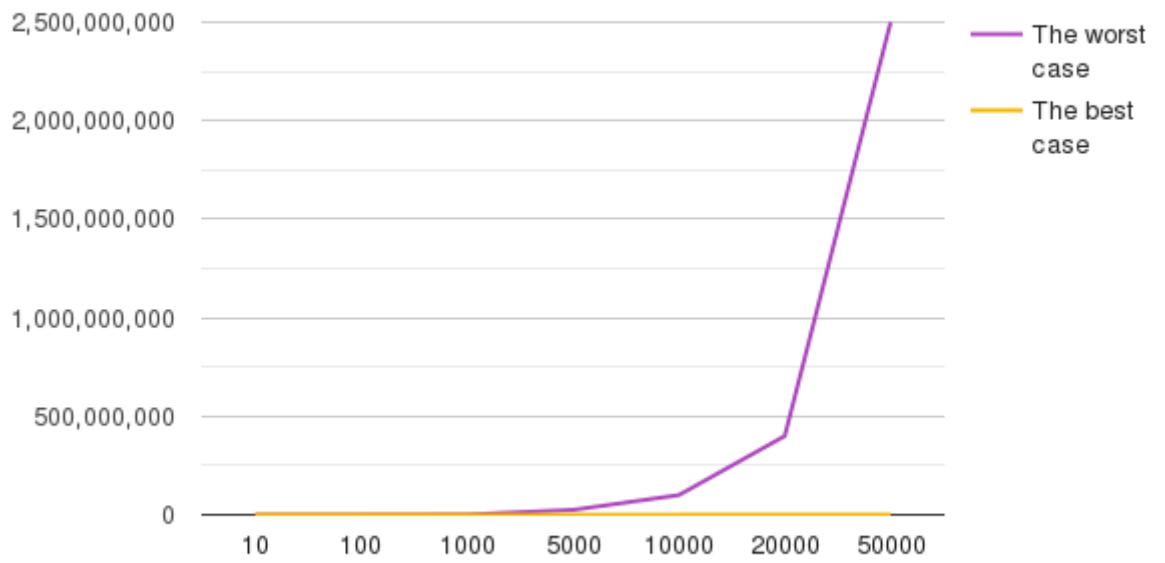
3.5.2 Графіки залежності часових характеристик оцінювання від розмірності масиву

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

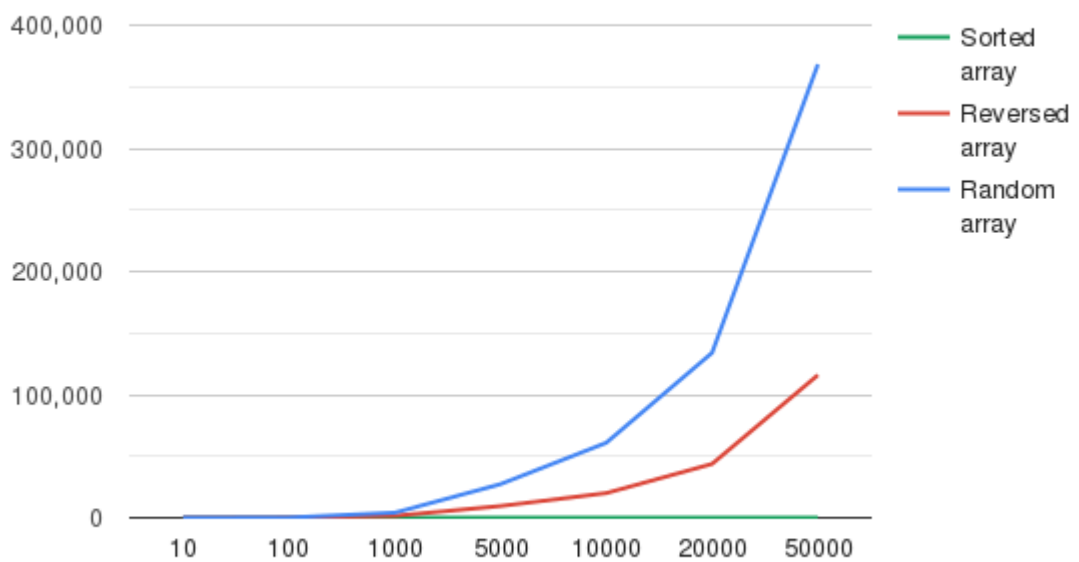
асимптотичні оцінки гіршого (фіолетовий графік) і кращого (жовтий графік) випадків для порівняння.

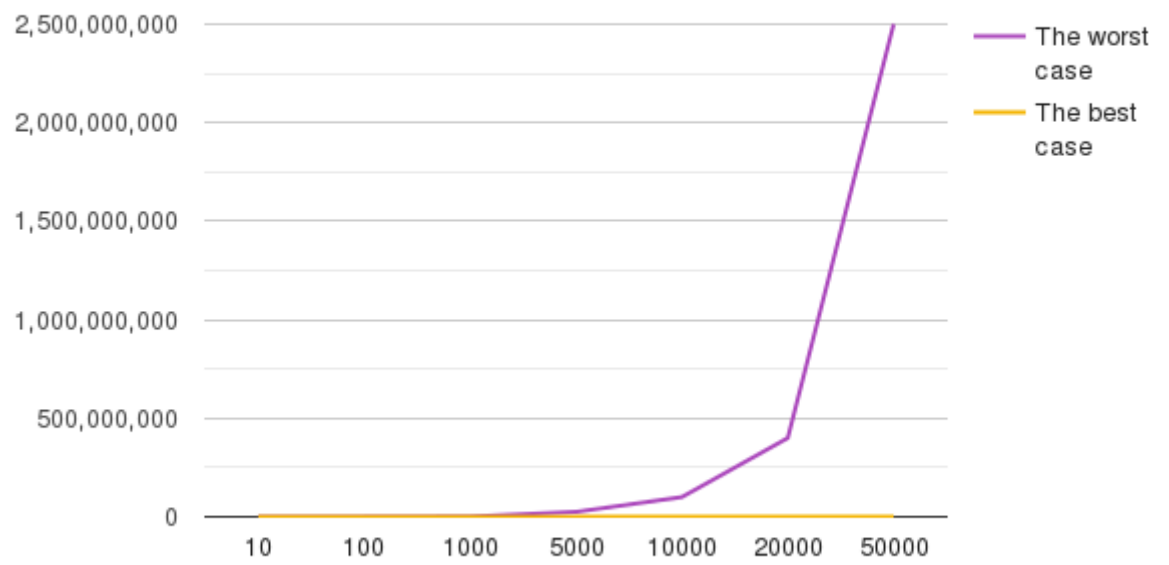
Рисунок 3.3 – Графіки залежності часових характеристик оцінювання  
Сортування бульбашкою





### Сортування гребінцем







## 9 ВИСНОВОК

При виконанні даної лабораторної роботи я вивчив основні методи аналізу обчислювальної складності алгоритмів внутрішнього сортування і оцінив поріг їх ефективності. Дослідив та порівняв алгоритми сортування бульбашкою та гребінцем. Протестував їх на масивах різної розмірності, порівняв часові характеристики, побудував графіки залежності часових характеристик оцінювання від розмірності масиву.

## 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%.