**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 ІП-11 Печковський Олександр Костянтинович

(шифр, прізвище, ім'я, по батькові)

Перевірила Халус Олена Андріївна

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc69772242)

[2 ЗаВдання 4](#_Toc69772243)

[3 Виконання 7](#_Toc69772244)

[3.1 Аналіз алгоритму на відповідність властивостям 7](#_Toc69772245)

[3.2 Псевдокод алгоритму 7](#_Toc69772246)

[3.3 Аналіз часової складності 7](#_Toc69772247)

[3.4 Програмна реалізація алгоритму 7](#_Toc69772248)

[3.4.1 Вихідний код 7](#_Toc69772249)

[3.4.2 Приклад роботи 8](#_Toc69772250)

[3.5 Тестування алгоритму 9](#_Toc69772251)

[3.5.1 Часові характеристики оцінювання 9](#_Toc69772252)

[3.5.2 Графіки залежності часових характеристик оцінювання від розмірності масиву 11](#_Toc69772253)

[Висновок 12](#_Toc69772254)

[Критерії оцінювання 13](#_Toc69772255)

# 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 алгоритмів на відповідність властивостям

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

Таблиця 3.1 – Аналіз алгоритмів на відповідність властивостям

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

## Псевдокод алгоритму

**Сортування бульбашкою:**

Функція bubble\_sort(array, size):

start\_time=поточний\_час

compare\_counter=0

swap\_counter=0

Для i в діапазоні(size-1):

Для j в діапазоні(size-i-1):

compare\_counter+=1

Якщо array[j]>array[j+1]:

array[j], array[j+1]=array[j+1], array[j]

swap\_counter+=1

sorting\_time=поточний\_час-start\_time

Виведення compare\_counter, swap\_counter, sorting\_time

**Сортування гребінцем:**

Функція comb\_sort(array2, size):

start\_time=поточний\_час

swap\_counter=0

compare\_counter=0

swaps = Істина

step=size

Поки step > 1 або swaps:

step = max(1, int(step / 1.247))

swaps = Хиба

Для i в діапазоні(size - step):

compare\_counter+=1

Якщо array2[i] > array2[i+step]:

array2[i], array2[i+step] = array2[i+step], array2[i]

swap\_counter+=1

swaps = Істина

sorting\_time=поточний\_час-start\_time

Виведення compare\_counter, swap\_counter, sorting\_time

## Аналіз часової складності

|  |  |  |
| --- | --- | --- |
| **Швидкодія** | **Сортування бульбашкою** | **Сортування гребінцем** |
| Найгірша швидкодія | О(n²) | О(n²) |
| Найкраща швидкодія | О(n) | О(n) |
| Середня швидкодія | О(n²) | O(nlogn) |

## Програмна реалізація алгоритму

### Вихідний код:

import random

from datetime import datetime

import copy

def create\_random\_array(size):

array=list(range(0, size))

random.shuffle(array)

return array

def create\_best\_array(size):

array=list(range(0, size))

return array

def create\_worst\_array(size):

array=[]

while size>0:

array.append(size-1)

size-=1

return array

def bubble\_sort(array, size):

start\_time=datetime.now()

compare\_counter=0

swap\_counter=0

for i in range (size-1):

for j in range (size-i-1):

compare\_counter+=1

if array[j]>array[j+1]:

array[j], array[j+1]=array[j+1], array[j]

swap\_counter+=1

sorting\_time=datetime.now()-start\_time

print("\n Бульбашкою: ")

#print(array)

print(" Кількість порівнянь: "+str(compare\_counter))

print(" Кількість перестановок: "+str(swap\_counter))

print(" Час сортування: "+str(sorting\_time))

def comb\_sort(array2, size):

start\_time=datetime.now()

swap\_counter=0

compare\_counter=0

swaps = True

step=size

while step > 1 or swaps:

step = max(1, int(step / 1.247))

swaps = False

for i in range(size - step):

compare\_counter+=1

if array2[i] > array2[i+step]:

array2[i], array2[i+step] = array2[i+step], array2[i]

swap\_counter+=1

swaps = True

sorting\_time=datetime.now()-start\_time

print("\n Гребінцем: ")

#print(array2)

print(" Кількість порівнянь: "+str(compare\_counter))

print(" Кількість перестановок: "+str(swap\_counter))

print(" Час сортування: "+str(sorting\_time))

size=int(input("Введіть розмір масиву: "))

array=create\_random\_array(size)

array2=copy.deepcopy(array)

print("\nВипадкова послідовність: ")

#print(array)

bubble\_sort(array, size)

comb\_sort(array2, size)

array=create\_best\_array(size)

array2=copy.deepcopy(array)

print("\nУпорядкована послідовність: ")

#print(array)

bubble\_sort(array, size)

comb\_sort(array2, size)

array=create\_worst\_array(size)

array2=copy.deepcopy(array)

print("\nЗворотньо упорядкована послідовність: ")

#print(array)

bubble\_sort(array, size)

comb\_sort(array2, size)

### Приклад роботи

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

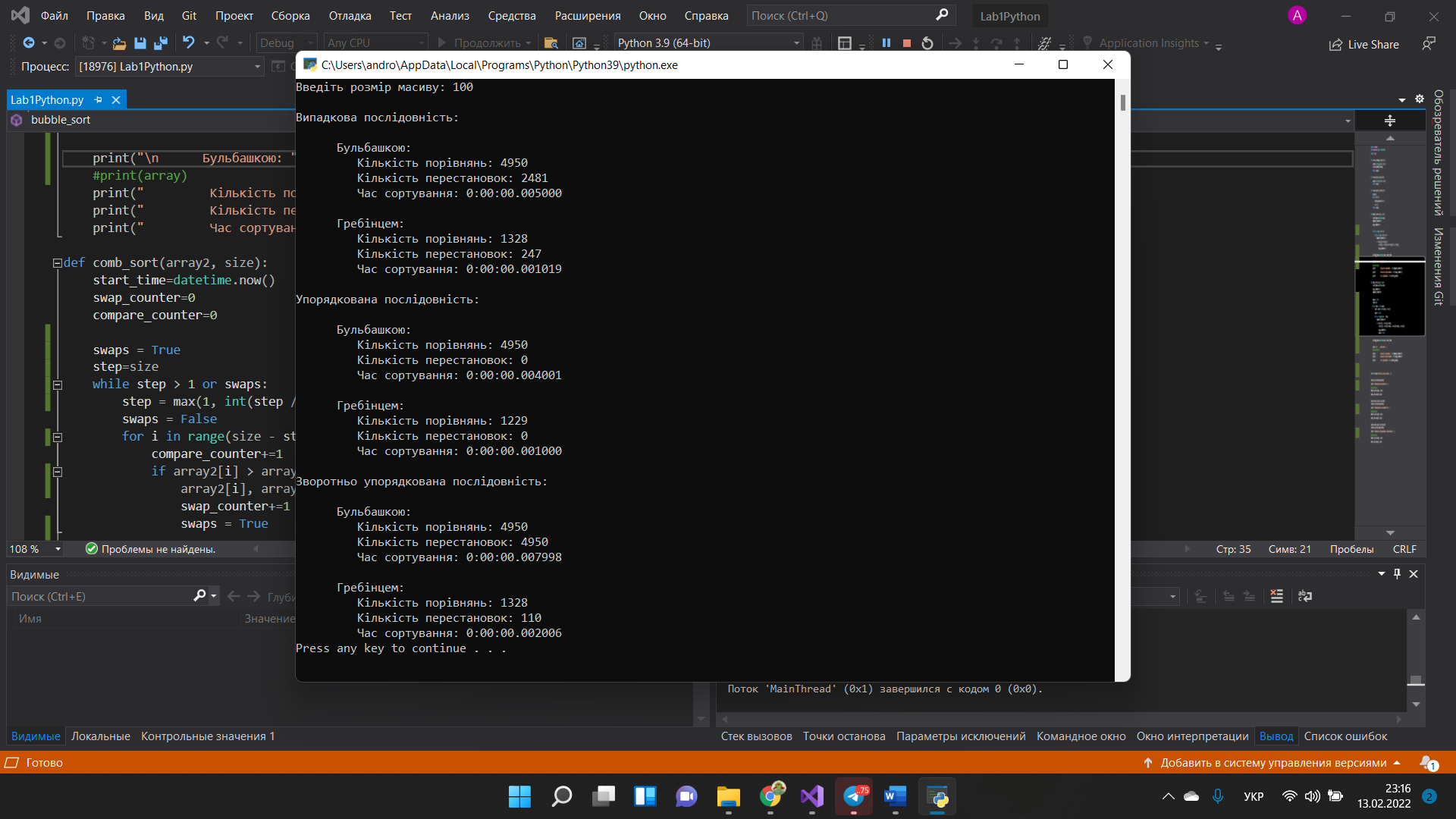


Рисунок 3.1 – Сортування масиву на 100 елементів

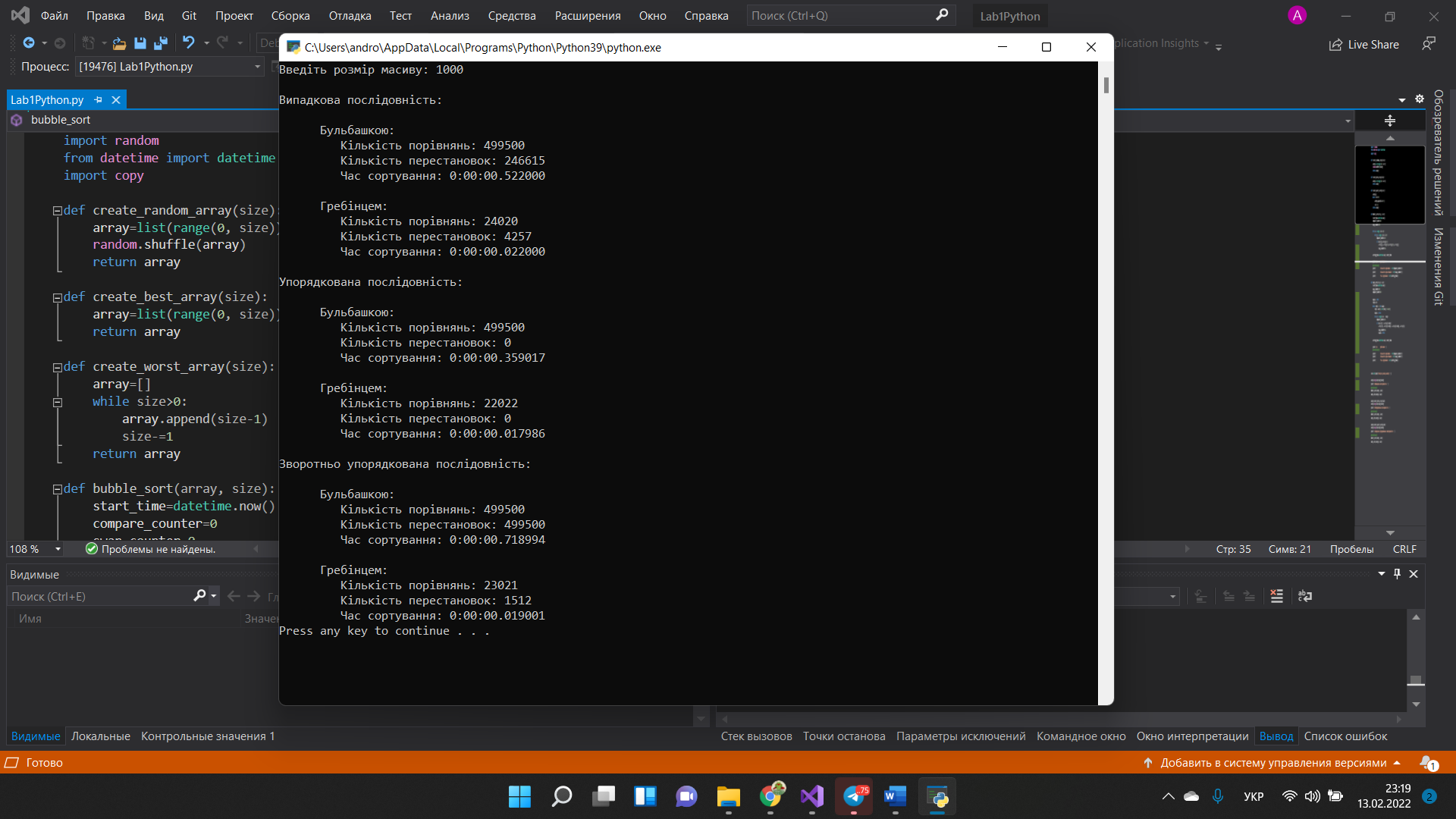


Рисунок 3.2 – Сортування масиву на 1000 елементів

## Тестування алгоритму

### 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 **бульбашкою** для упорядкованої послідовності елементів у масиві

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 0 |
| 100 | 4950 | 0 |
| 1000 | 499500 | 0 |
| 5000 | 12497500 | 0 |
| 10000 | 49995000 | 0 |
| 20000 | 199990000 | 0 |
| 50000 | 1249975000 | 0 |

**Гребінцем:**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 36 | 0 |
| 100 | 1229 | 0 |
| 1000 | 22022 | 0 |
| 5000 | 144832 | 0 |
| 10000 | 329598 | 0 |
| 20000 | 719136 | 0 |
| 50000 | 1997680 | 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 **бульбашкою** 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 |

**Гребінцем:**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 9 |
| 100 | 1328 | 110 |
| 1000 | 23021 | 1512 |
| 5000 | 149831 | 9154 |
| 10000 | 339597 | 19018 |
| 20000 | 739135 | 40730 |
| 50000 | 2047679 | 110332 |

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 **бульбашкою** для випадкової послідовності елементів у масиві.

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 23 |
| 100 | 4950 | 2573 |
| 1000 | 499500 | 246088 |
| 5000 | 12497500 | 6325488 |
| 10000 | 49995000 | 25075788 |
| 20000 | 199990000 | 100499455 |
| 50000 | 1249975000 | 626278565 |

**Гребінцем:**

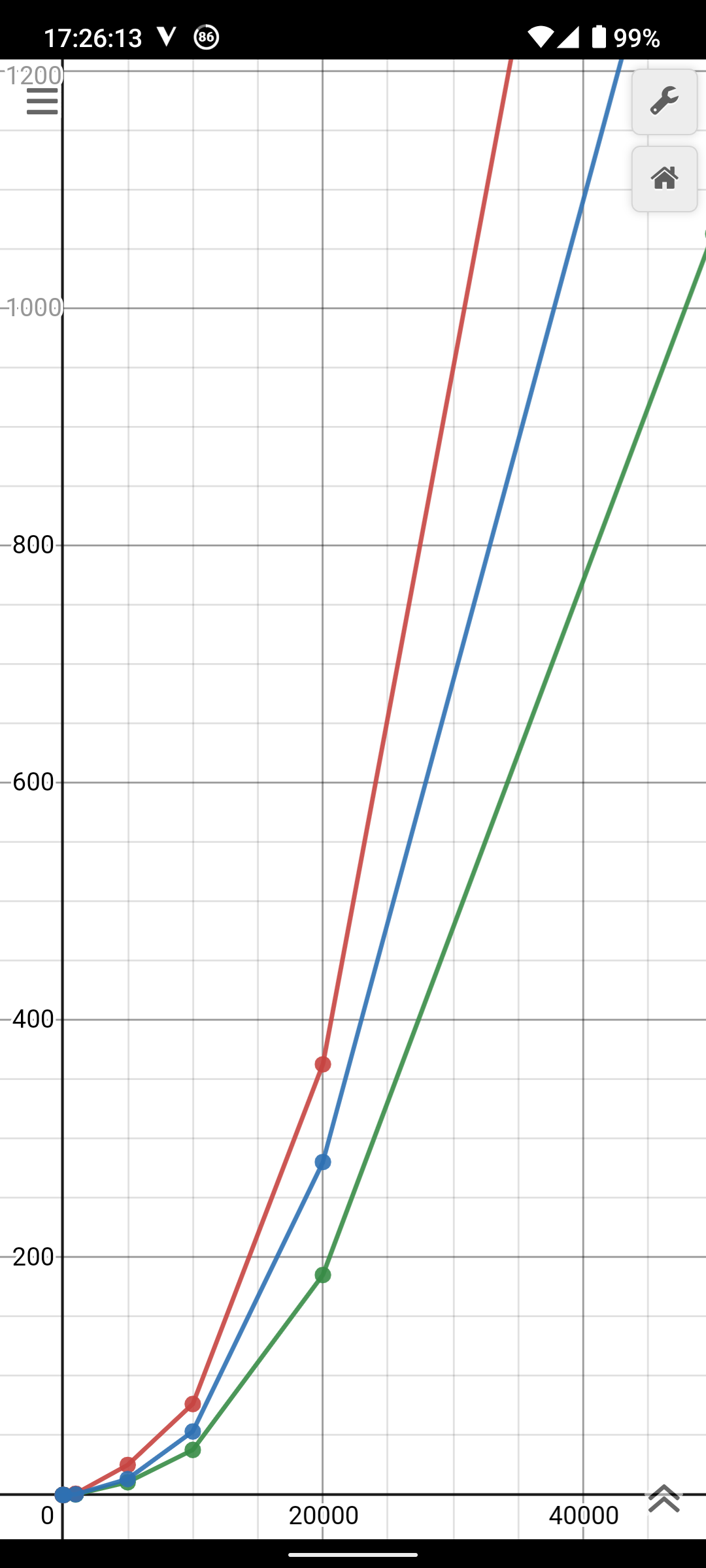
|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 15 |
| 100 | 1328 | 227 |
| 1000 | 23021 | 4326 |
| 5000 | 159829 | 27222 |
| 10000 | 349596 | 60208 |
| 20000 | 759134 | 131239 |
| 50000 | 2097678 | 367227 |

### Графіки залежності часових характеристик оцінювання від розмірності масиву

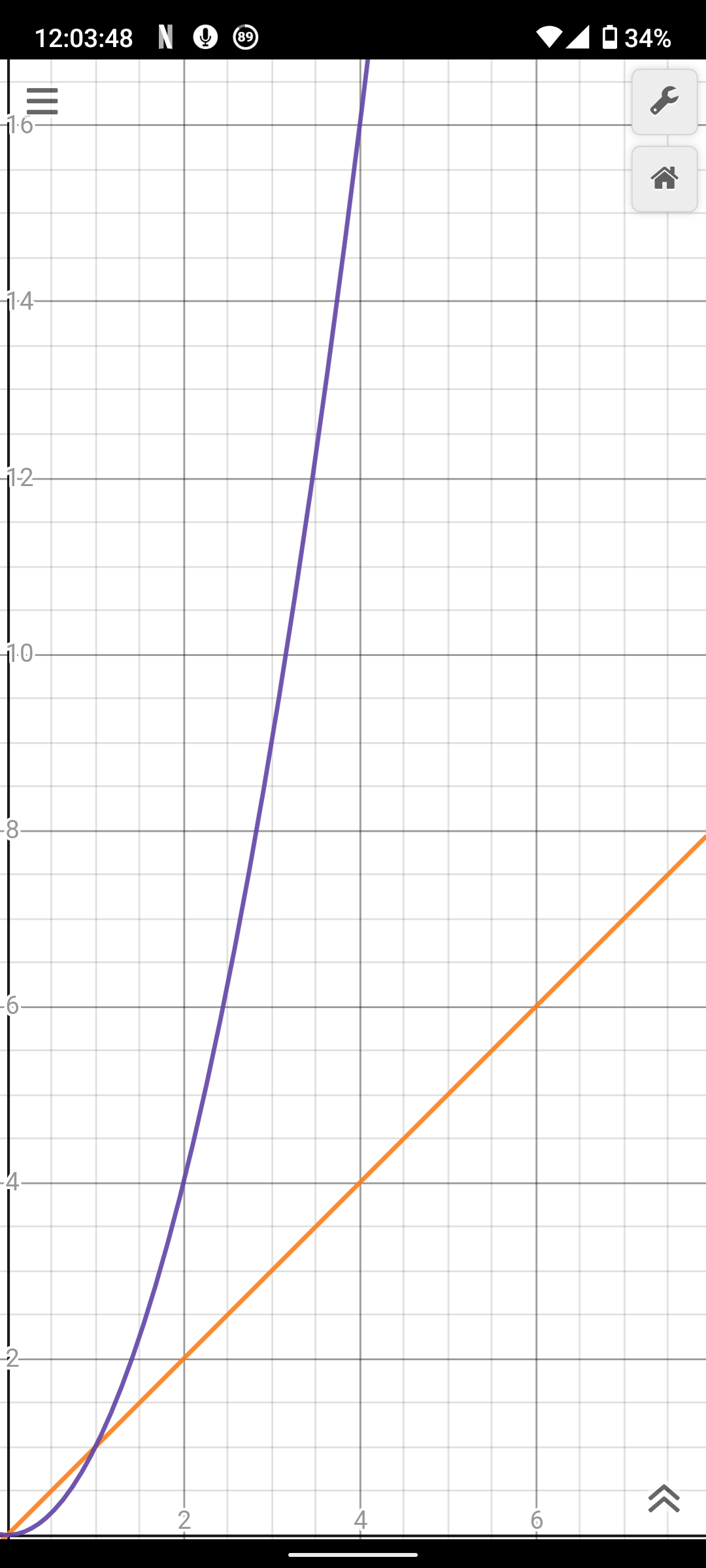
На рисунку 3.3 показані графіки залежності часових характеристик оцінювання від розмірності масиву для випадків, коли масиви містять упорядковану послідовність елементів (зелений графік), коли масиви містять зворотно упорядковану послідовність елементів (червоний графік), коли масиви містять випадкову 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

**Бульбашкою:**

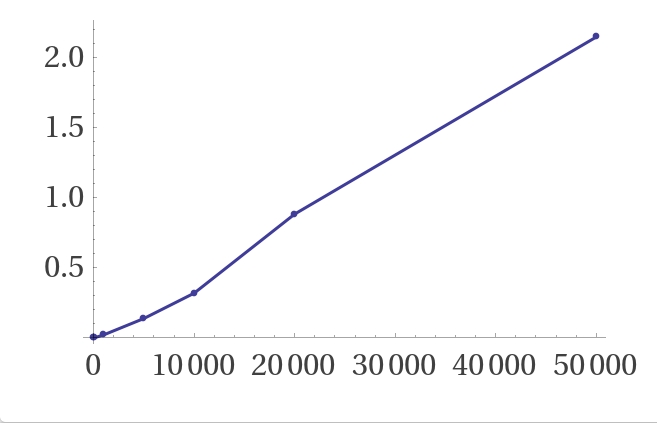
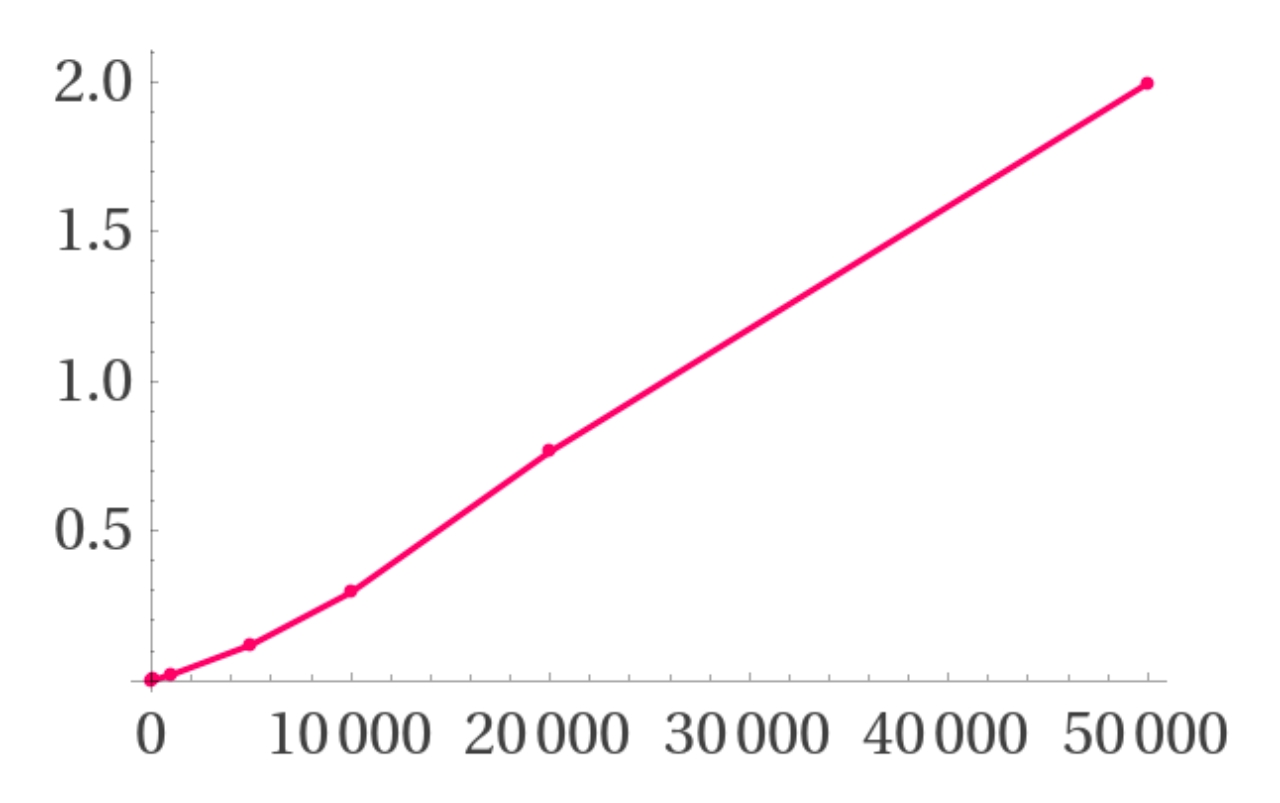
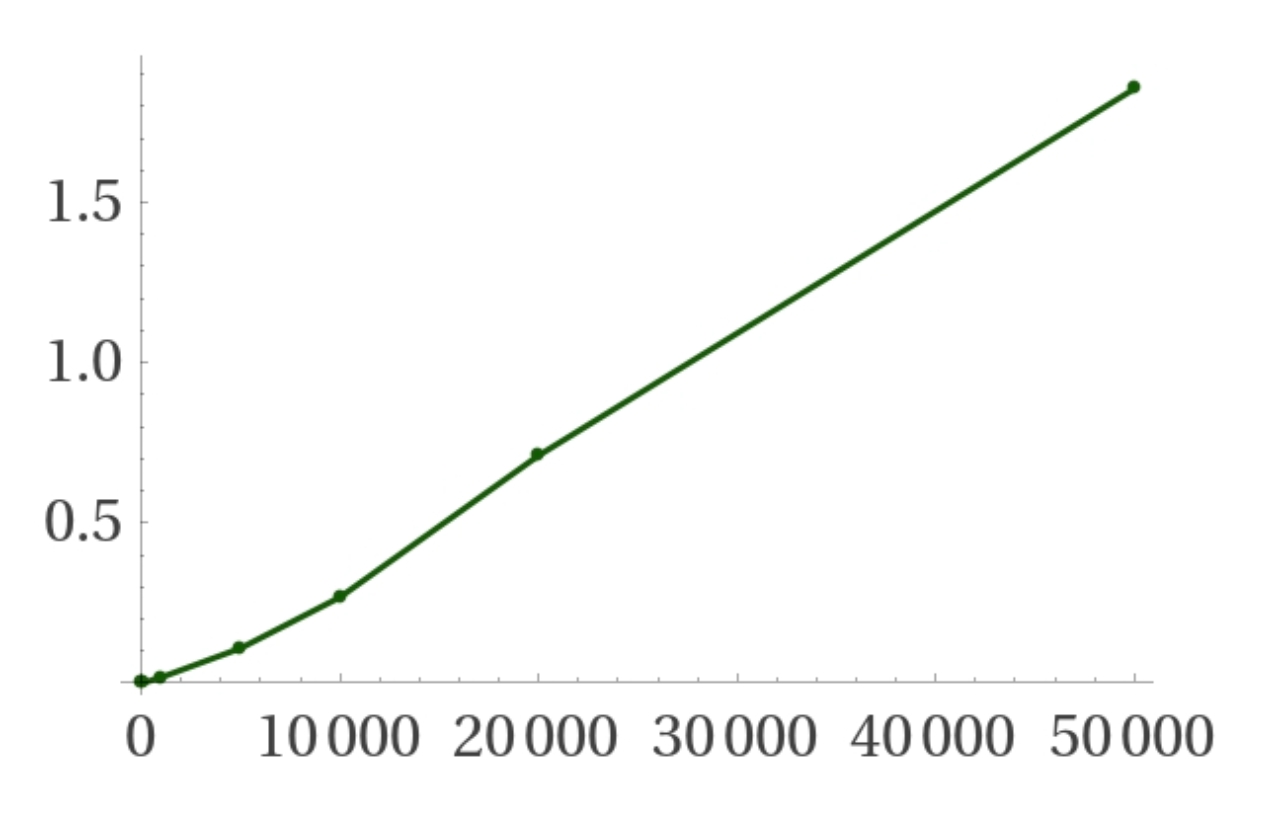
****

x – розмірність масиву, y – час сортування

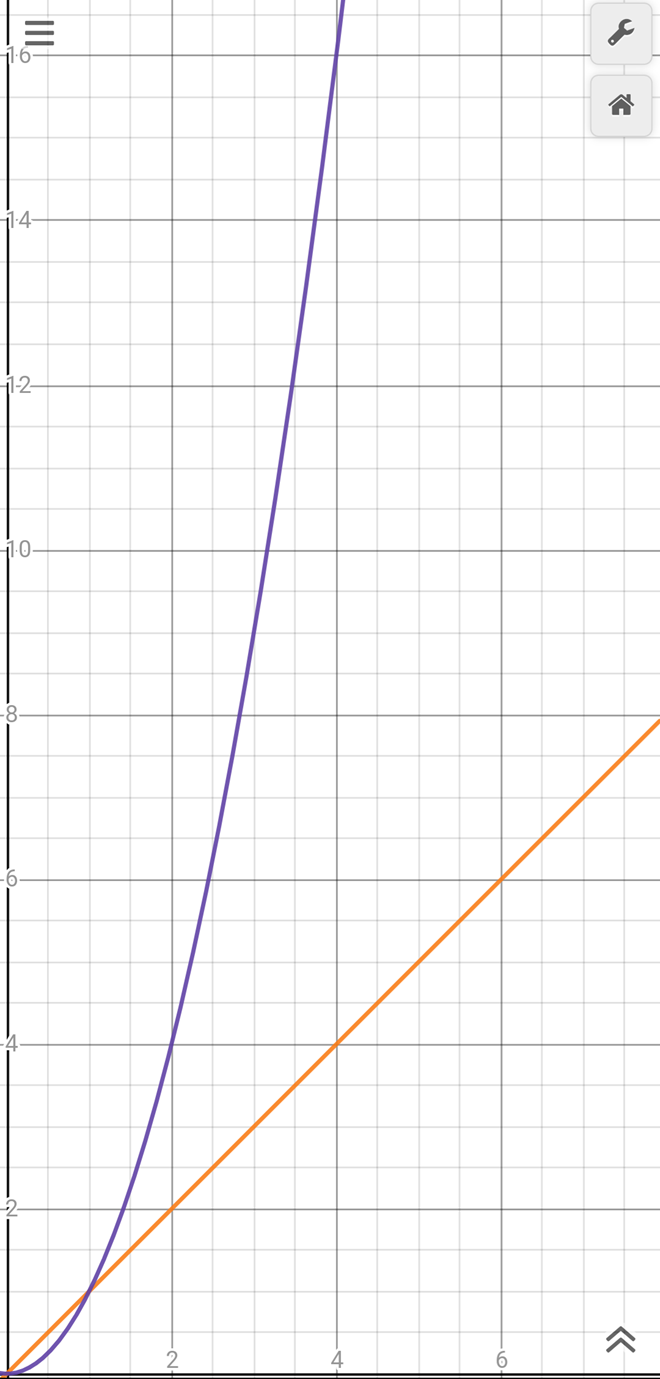


x – розмір вхідних даних, y – кількість операцій

**Гребінцем:**

****

x – розмірність масиву, y – час сортування

****

x – розмір вхідних даних, y – кількість операцій

Висновок

При виконанні даної лабораторної роботи я вивчив основні методи аналізу обчислювальної складності алгоритмів сортування бульбашкою та гребінцем, і оцінив поріг їхньої ефективності.

Я з’ясував, що алгоритм сортування гребінцем значно ефективніший за алгоритм сортування бульбашкою. Особливо це помітно при сортуванні масивів великої розмірності.

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%.