**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-11 Лисенко Андрій Юрійович*

**Перевірила**

(прізвище, ім'я, по батькові)

*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 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 | Так | Так |
| «Природність» поведінки (Adaptability) | Так | ні |
| Базуються на порівняннях | Так | Так |
| Необхідність в додатковій пам'яті (об'єм) | ні | ні |
| Необхідність в знаннях про структури даних | Так | Так |

## Псевдокод алгоритму

a) Bubble sort

size = len(arr)

comps = 0

swaps = 0

is\_sorted = False

while not is\_sorted:

i = 0

is\_sorted = True

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

comparings += 1

if arr[i] > arr[i + 1]:

swaps += 1

arr[i], arr[i + 1] = arr[i + 1], arr[i]

is\_sorted = False

i += 1

return comps, swaps

b) Comb sort

size = len(arr)

is\_sorted = False

gap = size

ratio = 1.24733

comps = 0

swaps = 0

while not is\_sorted and gap > 1:

gap = int(gap / ratio)

if gap < 1: gap = 1

for i in range(size - gap):

comps += 1

if arr[i] > arr[i + gap]:

swaps += 1

arr[i], arr[i + gap] = arr[i + gap], arr[i]

is\_sorted = False

return comps, swaps

## Аналіз часової складності

Обидва алгоритма містять вкладені цикли. Це означає, що найгірший випадок буде мати порядок зростання . У найкращому випадку алгоритм бульбашки матиме складність *Ω(n)* та гребінець - *Ω(nlog(n))*

## Програмна реалізація алгоритму

### Вихідний код

from random import shuffle

from time import time

def create\_array(size):

arr = list(range(1, size + 1))

shuffle(arr)

return arr

def bubble\_sort(arr):

size = len(arr)

comps = 0

swaps = 0

is\_sorted = False

while not is\_sorted:

i = 0

is\_sorted = True

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

comps += 1

if arr[i] > arr[i + 1]:

swaps += 1

arr[i], arr[i + 1] = arr[i + 1], arr[i]

is\_sorted = False

i += 1

return comps, swaps

def comb\_sort(arr):

size = len(arr)

is\_sorted = False

gap = size

ratio = 1.24733

comps = 0

swaps = 0

while not is\_sorted and gap > 1:

gap = int(gap / ratio)

if gap < 1: gap = 1

for i in range(size - gap):

comps += 1

if arr[i] > arr[i + gap]:

swaps += 1

arr[i], arr[i + gap] = arr[i + gap], arr[i]

is\_sorted = False

return comps, swaps

def out\_array(arr):

for i, item in enumerate(arr):

if (i + 1) % 20:

print('{0:>6}'.format(str(item)), end = '')

else:

print('{0:>6}'.format(str(item)))

def main():

size = 100000

arr = create\_array(size)

print('Unsorted array:')

out\_array(arr)

t\_start = time()

comps, swaps = comb\_sort(arr) #bubble\_sort(arr)

t = time() - t\_start

print('\n\nSorted array:')

out\_array(arr)

print(f'\n\nNumber of comparings: {comps}')

print(f'Number of swaps: {swaps}')

print(f'Sorting time: {t} seconds\n')

if \_\_name\_\_ == '\_\_main\_\_':

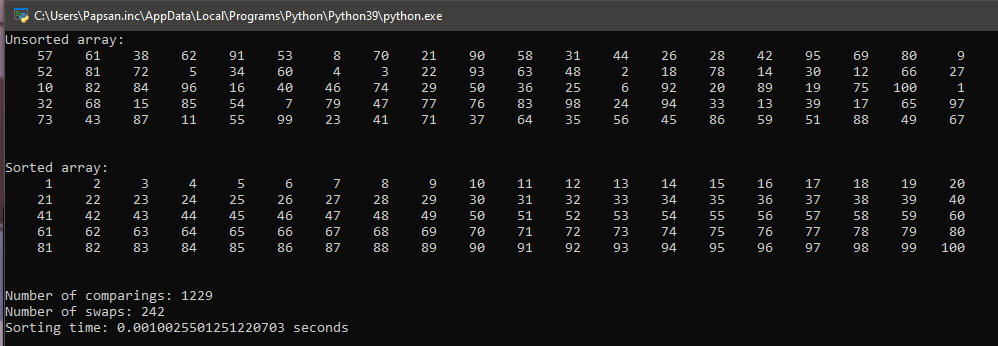
main()

### 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

**comb\_sort:**

****

**bubble\_sort:**

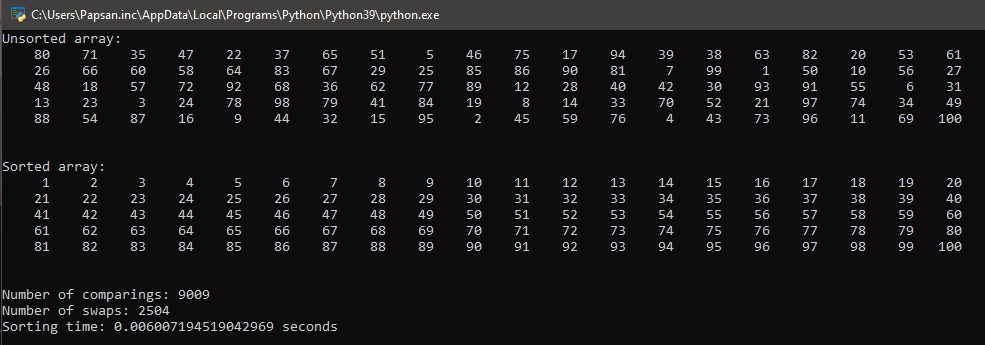
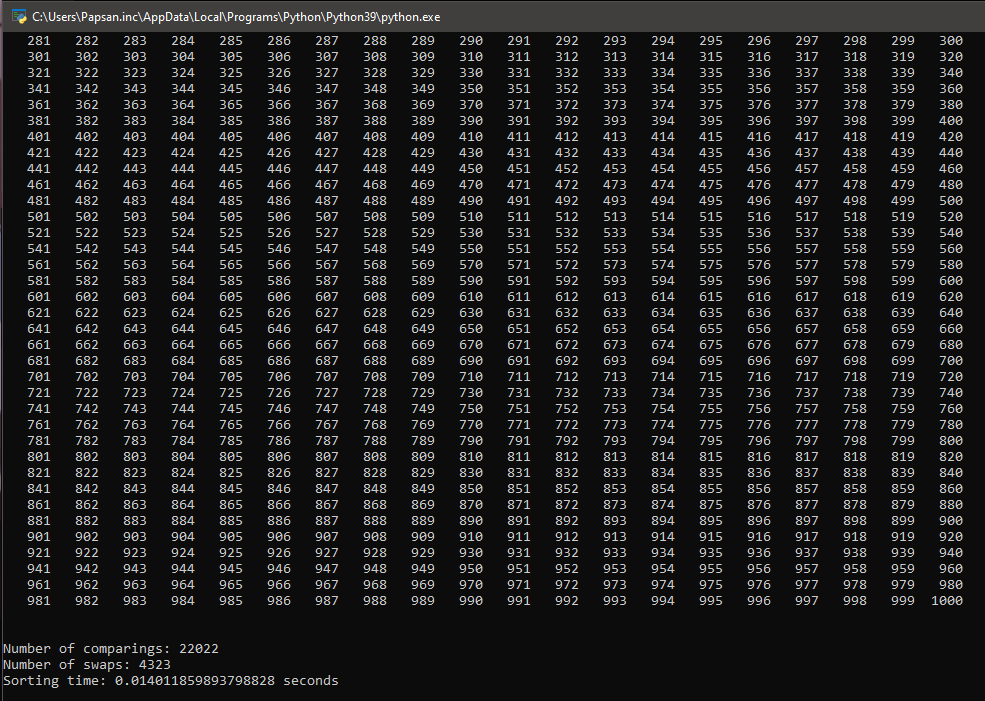
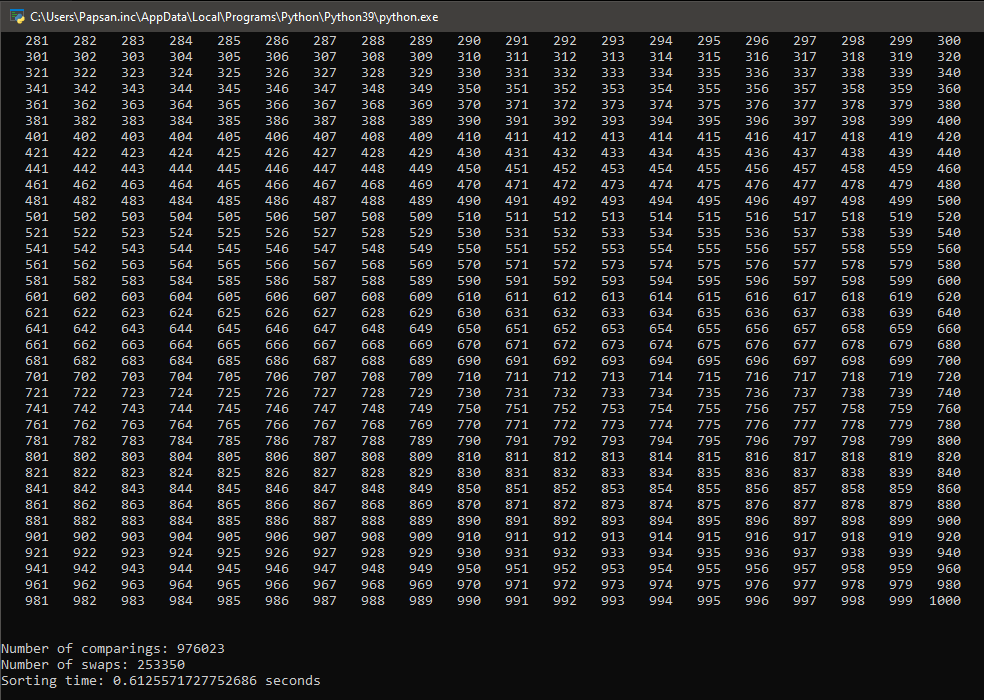


Рисунок 3.2 – Сортування масиву на 1000 елементів

**comb\_sort:**

****

**Bubble\_sort:**

****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 для упорядкованої послідовності елементів у масиві

**Bubble\_sort:**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 9 | 0 |
| 100 | 99 | 0 |
| 1000 | 999 | 0 |
| 5000 | 4 999 | 0 |
| 10000 | 9 999 | 0 |
| 20000 | 19 999 | 0 |
| 50000 | 49 999 | 0 |

**Comb\_sort:**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 36 | 0 |
| 100 | 1 229 | 0 |
| 1000 | 22 022 | 0 |
| 5000 | 144 862 | 0 |
| 10000 | 329 644 | 0 |
| 20000 | 719 241 | 0 |
| 50000 | 1 997 958 | 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 для зворотно упорядкованої послідовності елементів у масиві.

**Bubble\_sort:**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 90 | 45 |
| 100 | 9 900 | 4 950 |
| 1000 | 999 000 | 499 500 |
| 5000 | 24 995 000 | 12 497 500 |
| 10000 | 49 995 000 | 49 995 000 |
| 20000 | 399 980 000 | 199 990 000 |
| 50000 | 2 499 950 000 | 1 249 975 000 |

**Comb\_sort:**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 36 | 9 |
| 100 | 1 229 | 110 |
| 1000 | 22 022 | 1 512 |
| 5000 | 144 862 | 9 016 |
| 10000 | 329 644 | 19 132 |
| 20000 | 719 241 | 40 852 |
| 50000 | 1 997 958 | 109 958 |

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 | Число порівнянь | Число перестановок |
| 10 | 45 | 7 |
| 100 | 9 504 | 2 562 |
| 1000 | 954 045 | 252 052 |
| 5000 | 24 635 072 | 6 209 557 |
| 10000 | 98 660 133 | 24 903 926 |
| 20000 | 397 620 118 | 99 791 188 |
| 50000 | 2 491 700 165 | 623 008 568 |

**Comb\_sort:**

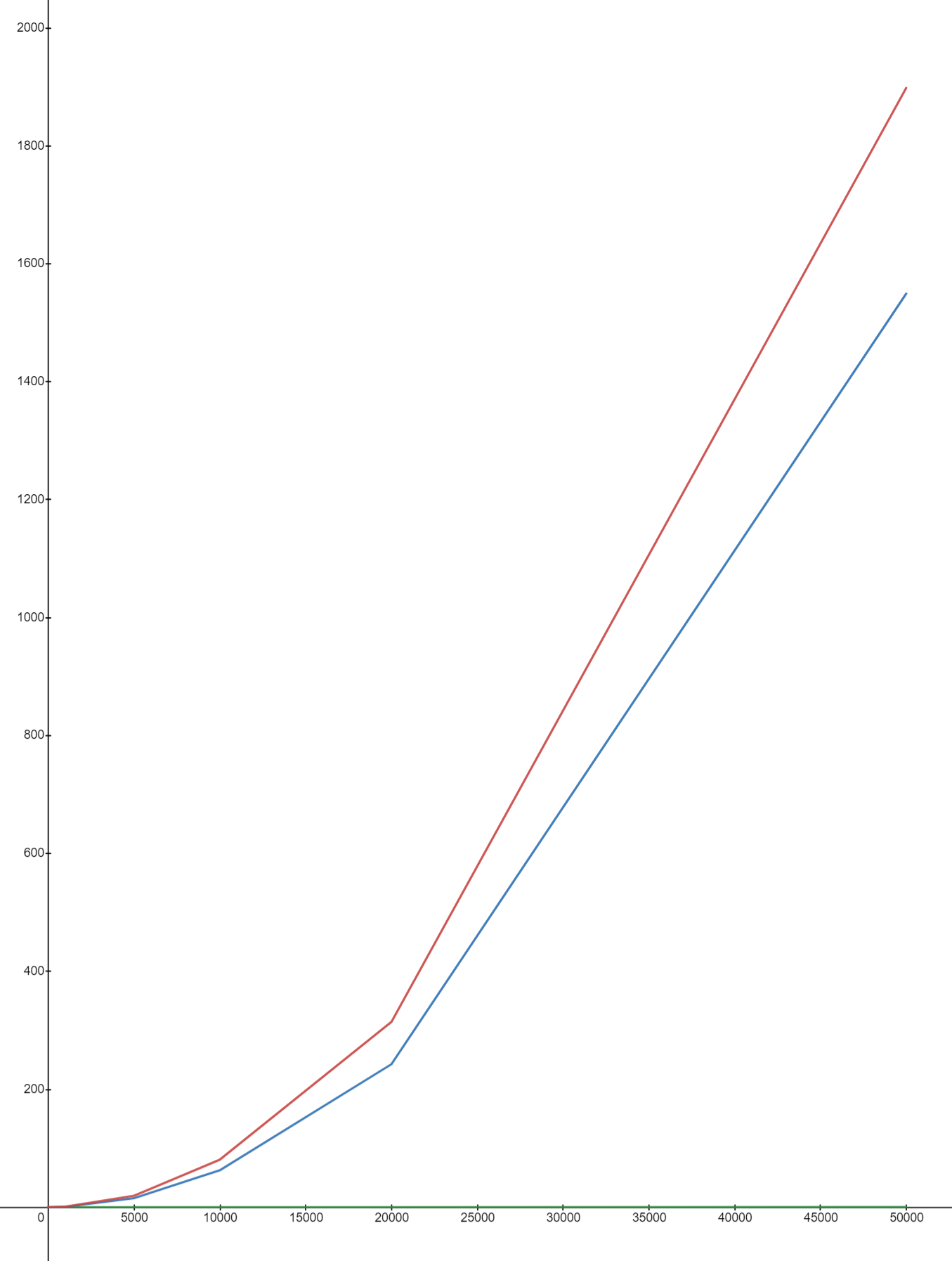
|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 36 | 7 |
| 100 | 1 229 | 222 |
| 1000 | 22 022 | 4 169 |
| 5000 | 144 862 | 27 415 |
| 10000 | 329 644 | 60 440 |
| 20000 | 719 241 | 130 900 |
| 50000 | 1 997 958 | 369 582 |

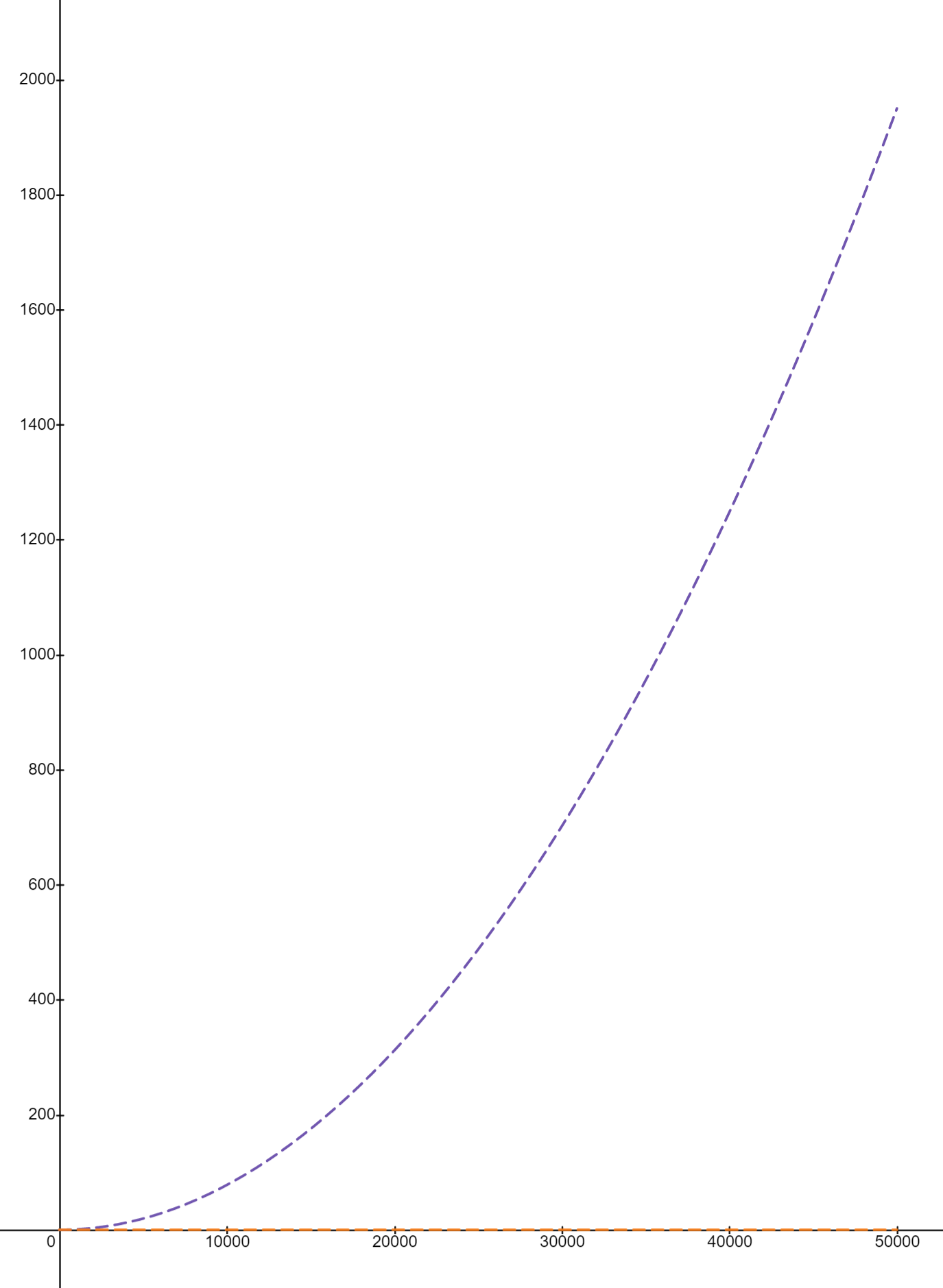
### Графіки залежності часових характеристик оцінювання від розмірності масиву

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.

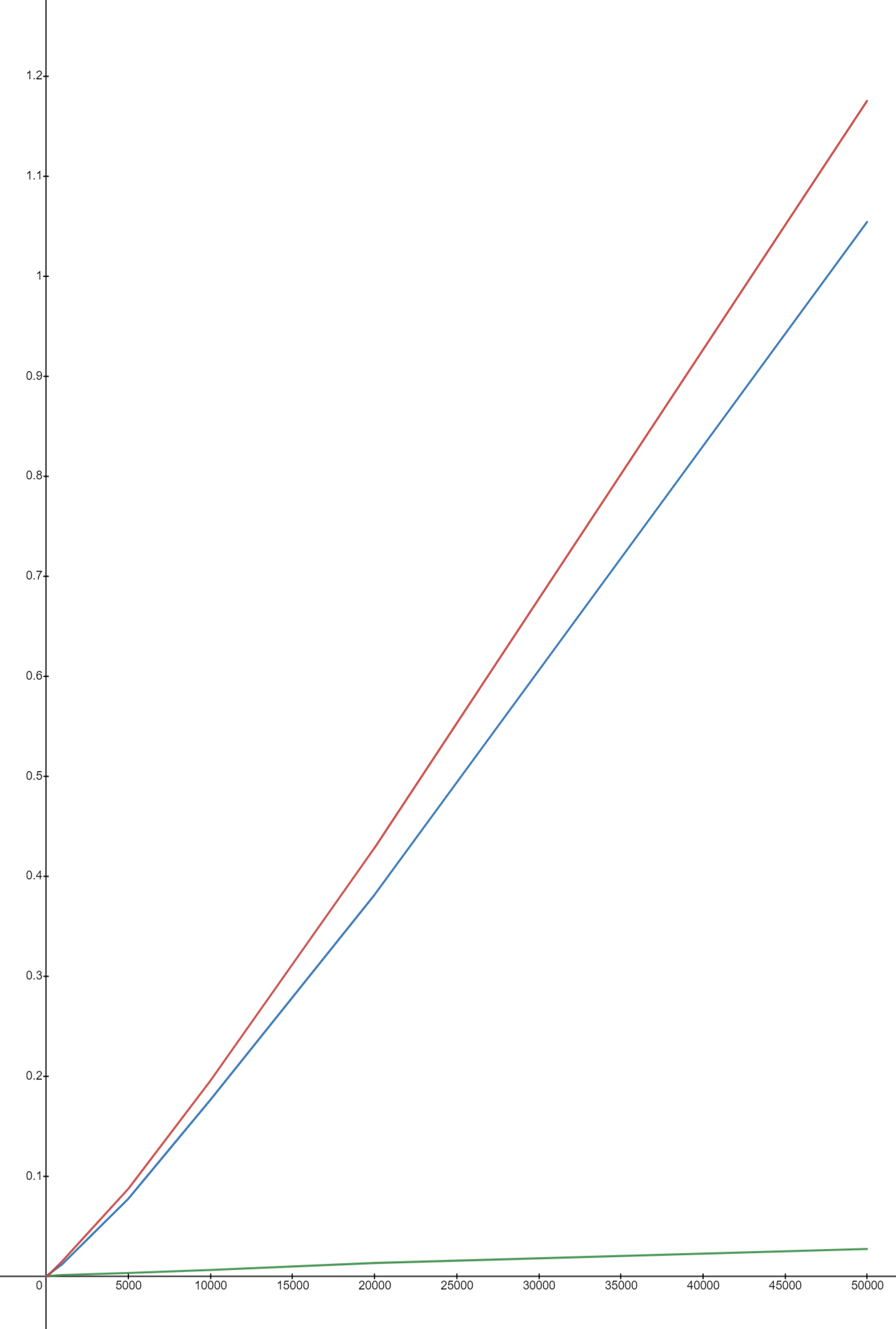
Рисунок 3.3 – Графіки залежності часових характеристик оцінювання

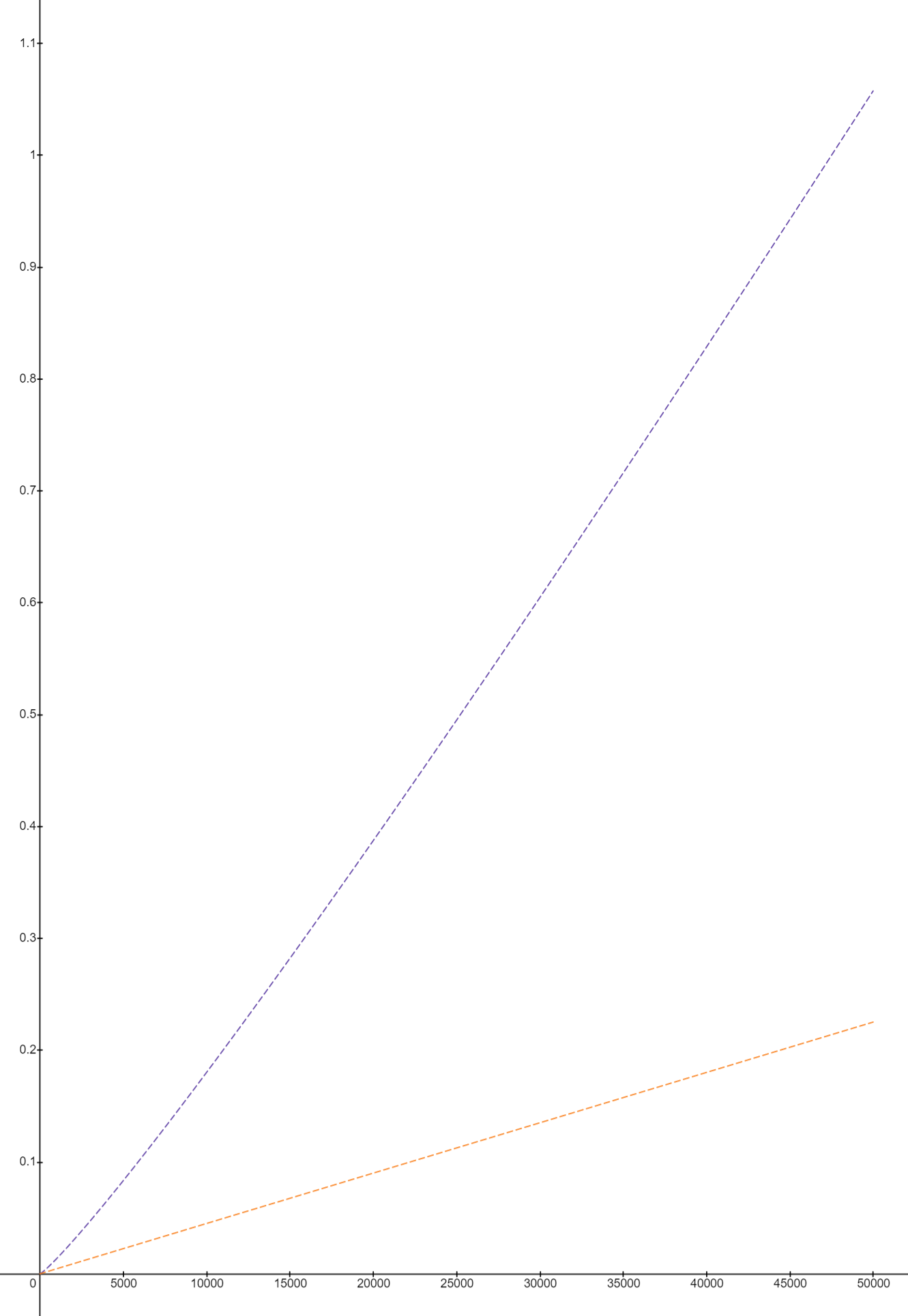
**Bubble\_sort:**

****

****

**Comb\_sort:**

****

****

Висновок

При виконанні даної лабораторної роботи, я отримав практичні навички в аналізі обчислювальної асимптотичної складності алгоритмів сортування, таких, як сортування бульбашкою та гребінцем. Визначив базові властивості цих алгоритмів та використав ці знання під час створення практичних програм. Побудував графіки залежності часу виконання програми від кількості елементів та порівняв їх між двома алгоритмами, а також між очікуванними результатами отриманих під час асимптотичного аналізу. Отже, я вивчив основні методи аналізу обчислювальної складності алгоритмів внутрішнього сортування і оцінив поріг їх ефективності.

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%.