**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-12 Сімчук Андрій Володимирович исоИввввВВВВВОВолоВолодимирович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов Олексій Олександрович*

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 | Стійкий | Не стійкий |
| «Природність» поведінки (Adaptability) | Ні | Так |
| Базуються на порівняннях | Так | Так |
| Необхідність в додатковій пам'яті (об'єм) | *θ*(1) | *θ*(1) |
| Необхідність в знаннях про структури даних | Знання про масиви | Знання про масиви |

## Псевдокод алгоритму

3.2.1 Сортування бульбашкою

**BubbleSort(array):**

n = size(array)

**повторити**

**для** і **від** 0 **до** n

**повторити**

**для** j **від** 0 **до** n

**якщо** array[j] > array[j + 1]:

**то**

temp = array[j]

array[j] = array[j + 1]

array[j + 1] = temp

**все якщо**

**все повторити**

**все повторити**

3.2.2 Сортування гребінцем

**CombSort(array):**

n = size(array)

step = n

swapped = true

**поки** step > 1 **або** swapped == true

step = step / 1.3

**якщо** step < 1

**то**

step = 1

**все якщо**

swapped = false

**повторити**

**для** i **від** 0 **до** n - step

**якщо** array[i] > array[i + step]

**то**

temp = array[і]

array[і] = array[і + step]

array[і + step] = temp

swapped = true

**все якщо**

**все повторити**

**все поки**

## Аналіз часової складності

3.3.1 Сортування бульбашкою

Найгірший випадок: O (n2)

Середній випадок: Ω (n2)

Найкращий випадок: Θ (n2)

3.3.2 Сортування гребінцем

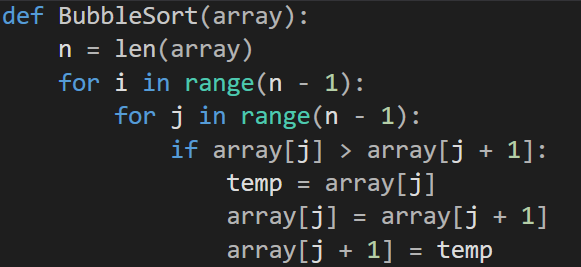
Найгірший випадок: O (n2)

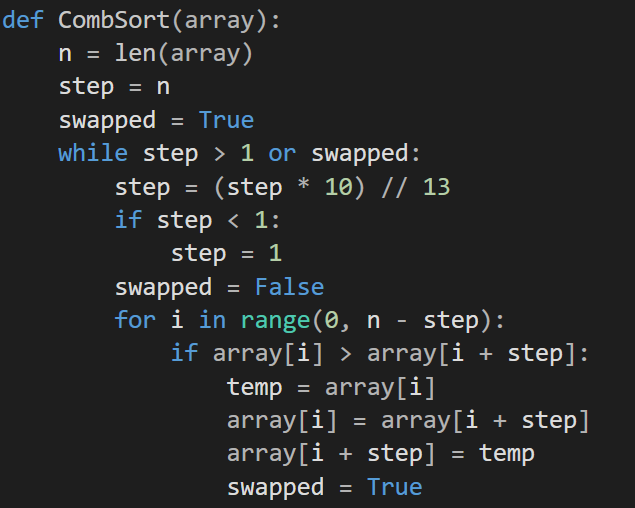
Середній випадок: Ω (n2)

Найкращий випадок: Θ (nlogn)

## Програмна реалізація алгоритму

### Вихідний код

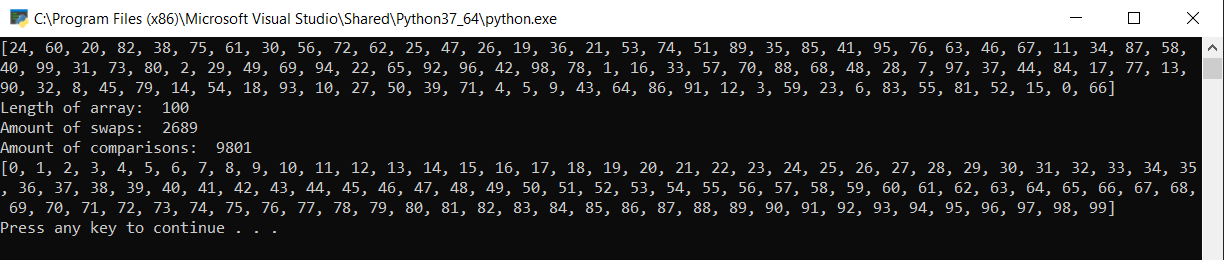
3.4.1.1 Сортування бульбашкою

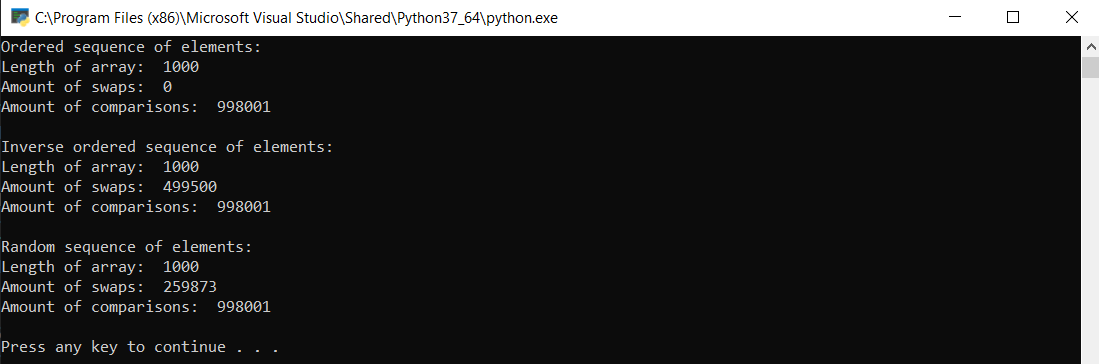
3.4.1.2 Сортування гребінцем

### Приклад роботи

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

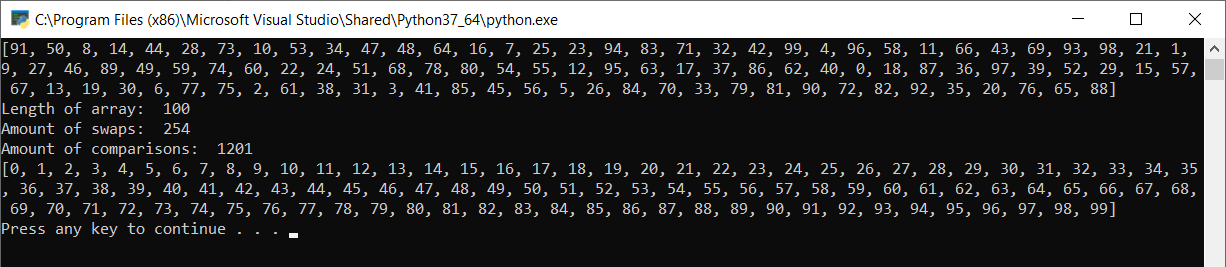
3.4.2.1 Сортування бульбашкою

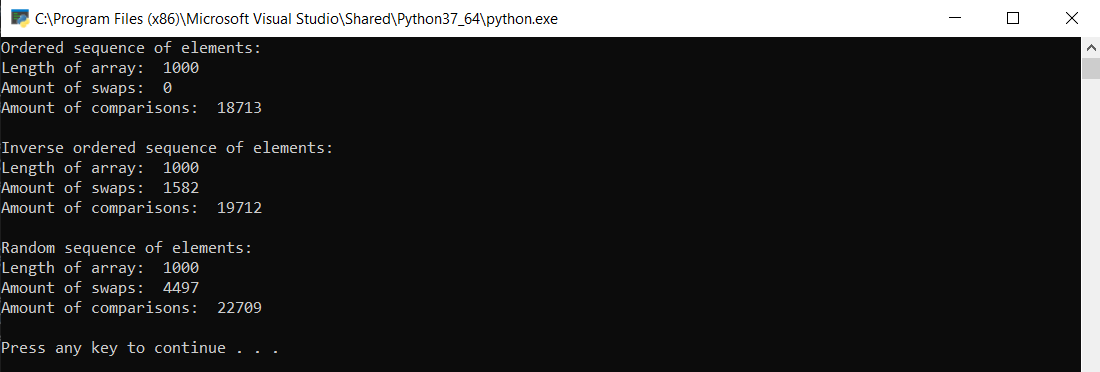
Рисунок 3.1.1 – Сортування масиву на 100 елементів

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

## Тестування алгоритму

### 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 |

В 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 | 17 |
| 100 | 9 801 | 2 132 |
| 1000 | 998 001 | 255 602 |
| 5000 | 24 990 001 | 6 300 270 |
| 10000 | 99 980 001 | 25 053 983 |
| 20000 | 399 960 001 | 99 352 414 |
| 50000 | 2 499 900 001 | 625 605 438 |

3.5.1.2 Сортування гребінцем

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

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

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 32 | 0 |
| 100 | 1 003 | 0 |
| 1000 | 18 713 | 0 |
| 5000 | 123 386 | 0 |
| 10000 | 276 739 | 0 |
| 20000 | 613 402 | 0 |
| 50000 | 1 683 412 | 0 |

В таблиці 3.3.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки для масивів різної розмірності, коли масиви містять зворотно упорядковану послідовність елементів.

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

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 41 | 7 |
| 100 | 1 102 | 122 |
| 1000 | 19 712 | 1 582 |
| 5000 | 128 385 | 9 572 |
| 10000 | 286 738 | 20 078 |
| 20000 | 633 401 | 42 634 |
| 50000 | 1 733 411 | 116 838 |

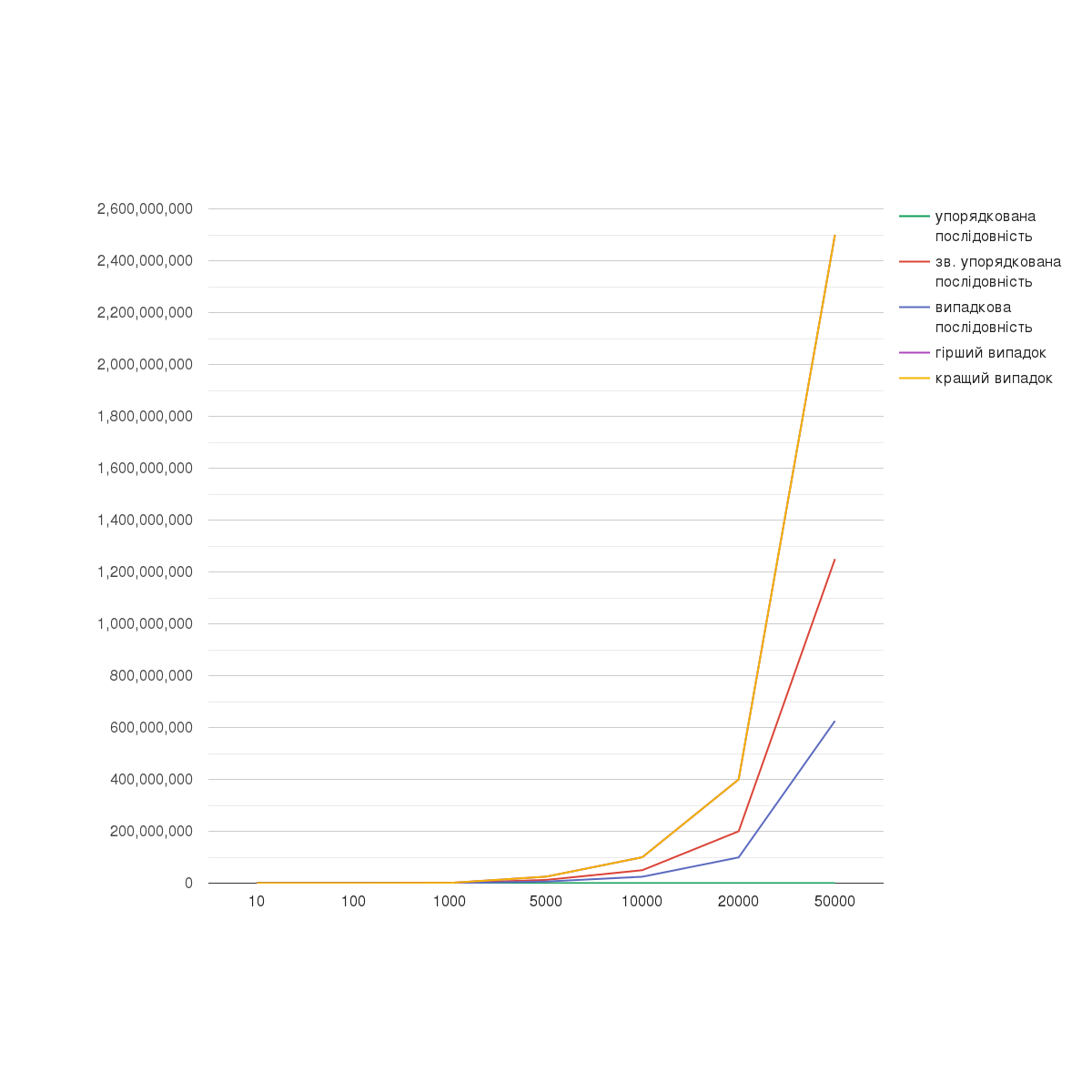
У таблиці 3.4.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки для масивів різної розмірності, масиви містять випадкову послідовність елементів.

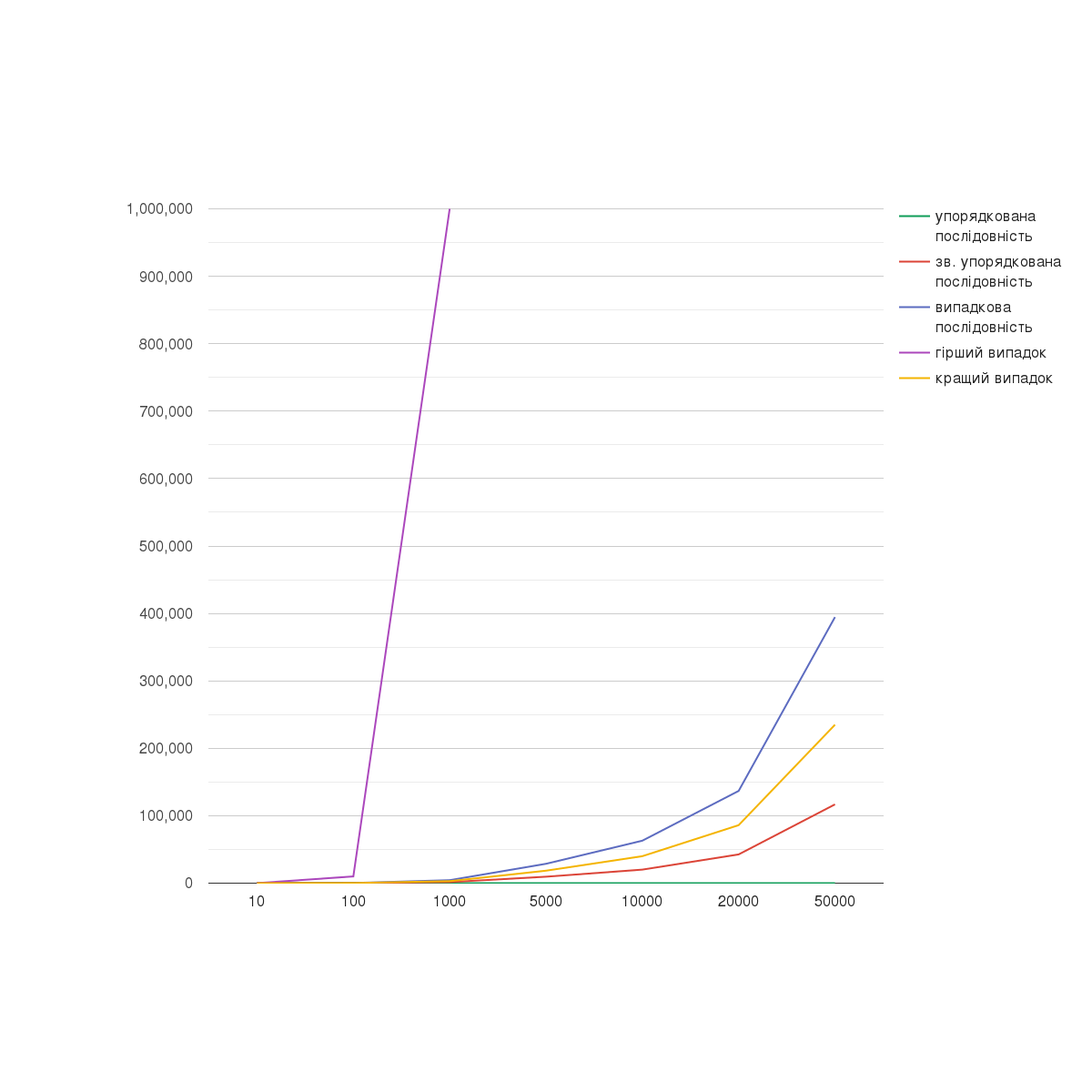
Таблиця 3.4.2 – Характеристика оцінювання алгоритму сортування бульбашки для випадкової послідовності елементів у масиві.

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 41 | 7 |
| 100 | 1 201 | 238 |
| 1000 | 22 709 | 4 493 |
| 5000 | 148 381 | 29 116 |
| 10000 | 326 734 | 63 621 |
| 20000 | 693 398 | 136 595 |
| 50000 | 2 283 400 | 396 144 |

### 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, записати алгоритми внутрішніх сортувань за допомогою псевдокоду, проведено аналіз часової складності в гіршому, кращому і середньому випадках та записано часову складність в асимптотичних оцінках.

Було виконано програмну реалізацію алгоритмів на Python з фіксацією часових характеристик оцінювання, проведено ряд випробувань алгоритму та досліджено формули для кількості операцій у оптимізованому  та звичайному алгоритмі сортування «бульбашкою», порівняно його з сортуванням «гребінцем» і побудовано відповідні графіки.

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.

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%.