**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 Васильєв Єгор Костянтинович*

**Перевірив**

(прізвище, ім'я, по батькові)

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

## Псевдокод алгоритму

**Бульбашка**

n = length(A)

tmp

**повторити** (i=0, n-1) **раз**

**повторити** (j=0, n-1) **раз**

**якщо** A[j]>A[j+1]

**то**

tmp = A[j]

A[j] = A[j+1]

A[j+1] = tmp

**кінець якщо**

**кінець повторити**

**кінець повторити**

**Гребінець**

**new\_gap:**

gap=(gap\*10)//13

**якщо** gap<1

**то**

**повернути 1**

**повернути** gap

**comb\_sort:**

tmp

n = length(A)

gap=n

swapped=True

**поки** gap>1 **або** swapped==True

gap = new\_gap (gap)

swapped = False

**повторити** (i=0, n-gap) **раз**

**якщо** A[i]>A[i+gap]

**то**

tmp = A[i]

A[i] = A[i+gap]

A[i+gap] = tmp

swapped = True

**кінець якщо**

**кінець повторити**

## Аналіз часової складності

**Бульбашка**

Найгірший випадок: O ()

Середній випадок: Ω ()

Найкращий випадок: Θ ()

**Гребінець**

Найгірший випадок: O ()

Середній випадок: Ω ()

Найкращий випадок: Θ (*nlogn*)

## Програмна реалізація алгоритму

### Вихідний код

**Бульбашка**

def bubble\_sort(arr):  
 n = len(arr)  
 for i in range(n - 1):  
 for j in range(0, n - 1):  
 if arr[j] > arr[j + 1]:  
 arr[j], arr[j + 1] = arr[j + 1], arr[j]

**Гребінець**

def get\_next\_gap(gap):  
 gap = (gap \* 10) // 13  
 if gap < 1:  
 return 1  
 return gap  
  
  
def comb\_sort(arr):  
 n = len(arr)  
 gap = n  
 swapped = True  
 while gap > 1 or swapped:  
 gap = get\_next\_gap(gap)  
 swapped = False  
 for i in range(0, n - gap):  
 if arr[i] > arr[i + gap]:  
 arr[i], arr[i + gap] = arr[i + gap], arr[i]  
 swapped = True

### Приклад роботи

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

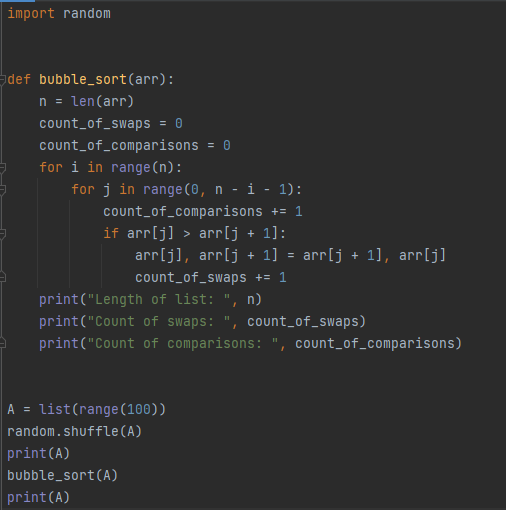
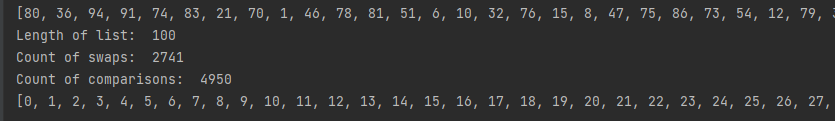
**Бульбашка**

Рисунок 3.1 – Сортування масиву на 100 елементів

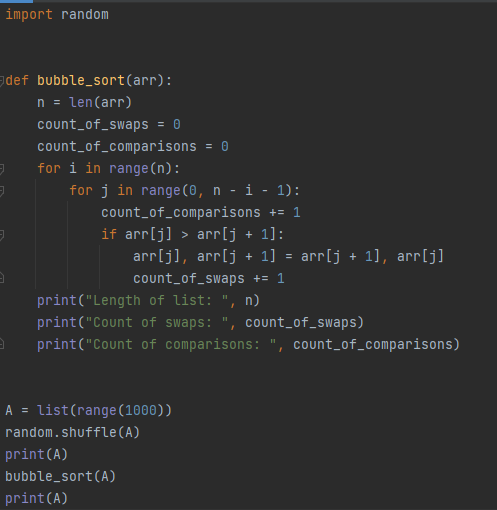
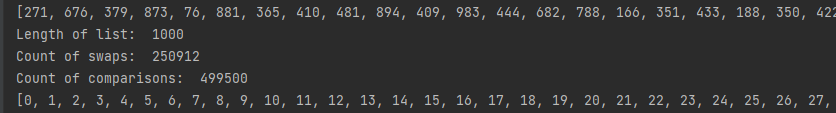


Рисунок 3.1 – Сортування масиву на 1000 елементів

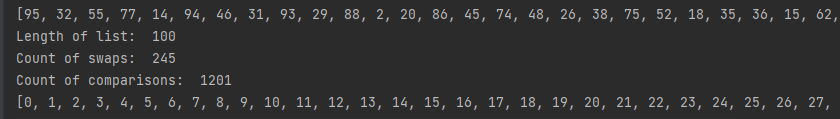
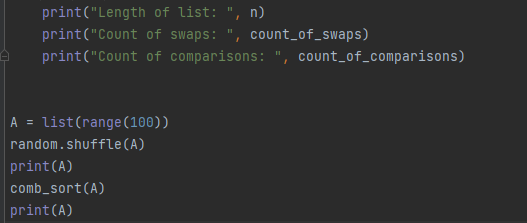
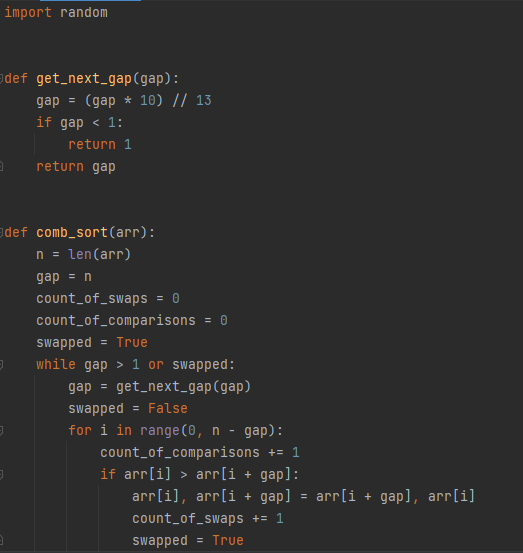
**Гребінець**

Рисунок 3.1 – Сортування масиву на 100 елементів

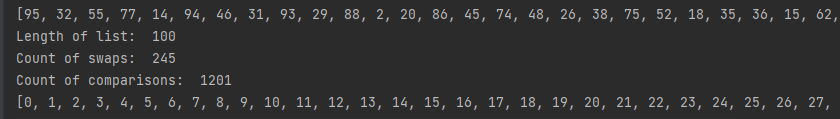
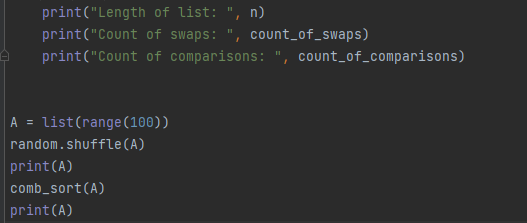
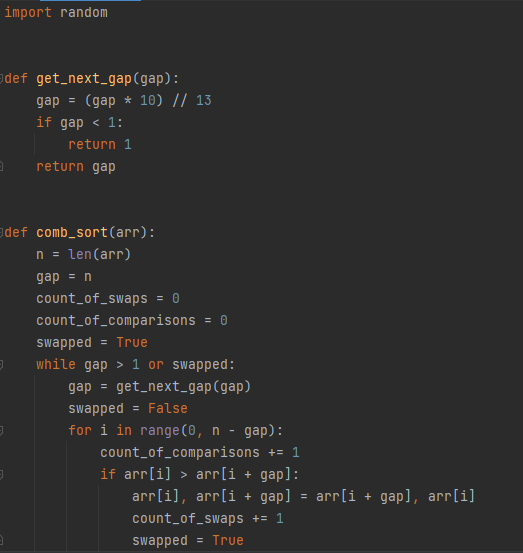
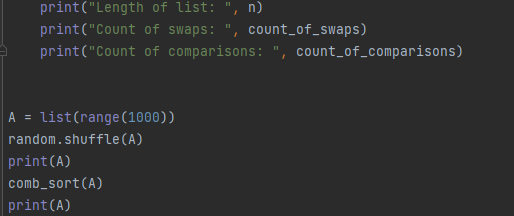
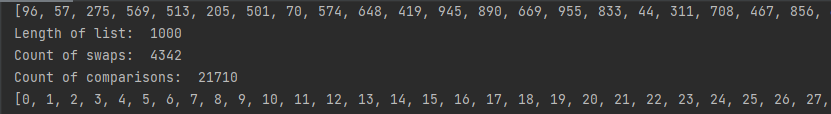


Рисунок 3.1 – Сортування масиву на 1000 елементів

## 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

для упорядкованої послідовності елементів у масиві

**Бульбашка**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь у коді (оптимізований/звичайний) | Число перестановок у коді |
| 10 | 45 (81) | 0 |
| 100 | 4950 (9801) | 0 |
| 1000 | 499500 (998001) | 0 |
| 5000 | 12497500 (24990001) | 0 |
| 10000 | 49995000 (99980001) | 0 |
| 20000 | 199990000 (399960001) | 0 |
| 50000 | 1249975000 (2499900001) | 0 |

**Гребінець**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь у коді | Число перестановок у коді |
| 10 | 32 | 0 |
| 100 | 1003 | 0 |
| 1000 | 18713 | 0 |
| 5000 | 123386 | 0 |
| 10000 | 276739 | 0 |
| 20000 | 613402 | 0 |
| 50000 | 1683412 | 0 |

В таблиці 3.3 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування 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.

**Бульбашка**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь у коді (оптимізований/звичайний) | Число перестановок у коді |
| 10 | 45 (81) | 45 |
| 100 | 4950 (9801) | 4950 |
| 1000 | 499500 (998001) | 499500 |
| 5000 | 12497500 (24990001) | 12497500 |
| 10000 | 49995000 (99980001) | 49995000 |
| 20000 | 199990000 (399960001) | 199990000 |
| 50000 | 1249975000 (2499900001) | 1249975000 |

**Гребінець**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь у коді | Число перестановок у коді |
| 10 | 41 | 7 |
| 100 | 1102 | 122 |
| 1000 | 19712 | 1582 |
| 5000 | 128385 | 9572 |
| 10000 | 286738 | 20078 |
| 20000 | 633401 | 42634 |
| 50000 | 1733411 | 116838 |

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

Таблиця 3.4 – Характеристика оцінювання алгоритму сортування для випадкової послідовності елементів у масиві.

**Бульбашка**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь у коді (оптимізований/звичайний) | Число перестановок у коді |
| 10 | 45 (81) | 19 |
| 100 | 4950 (9801) | 2426 |
| 1000 | 499500 (998001) | 267112 |
| 5000 | 12497500 (24990001) | 6241329 |
| 10000 | 49995000 (99980001) | 24740358 |
| 20000 | 199990000 (399960001) | 99352414 |
| 50000 | 1249975000 (2499900001) | 625605438 |

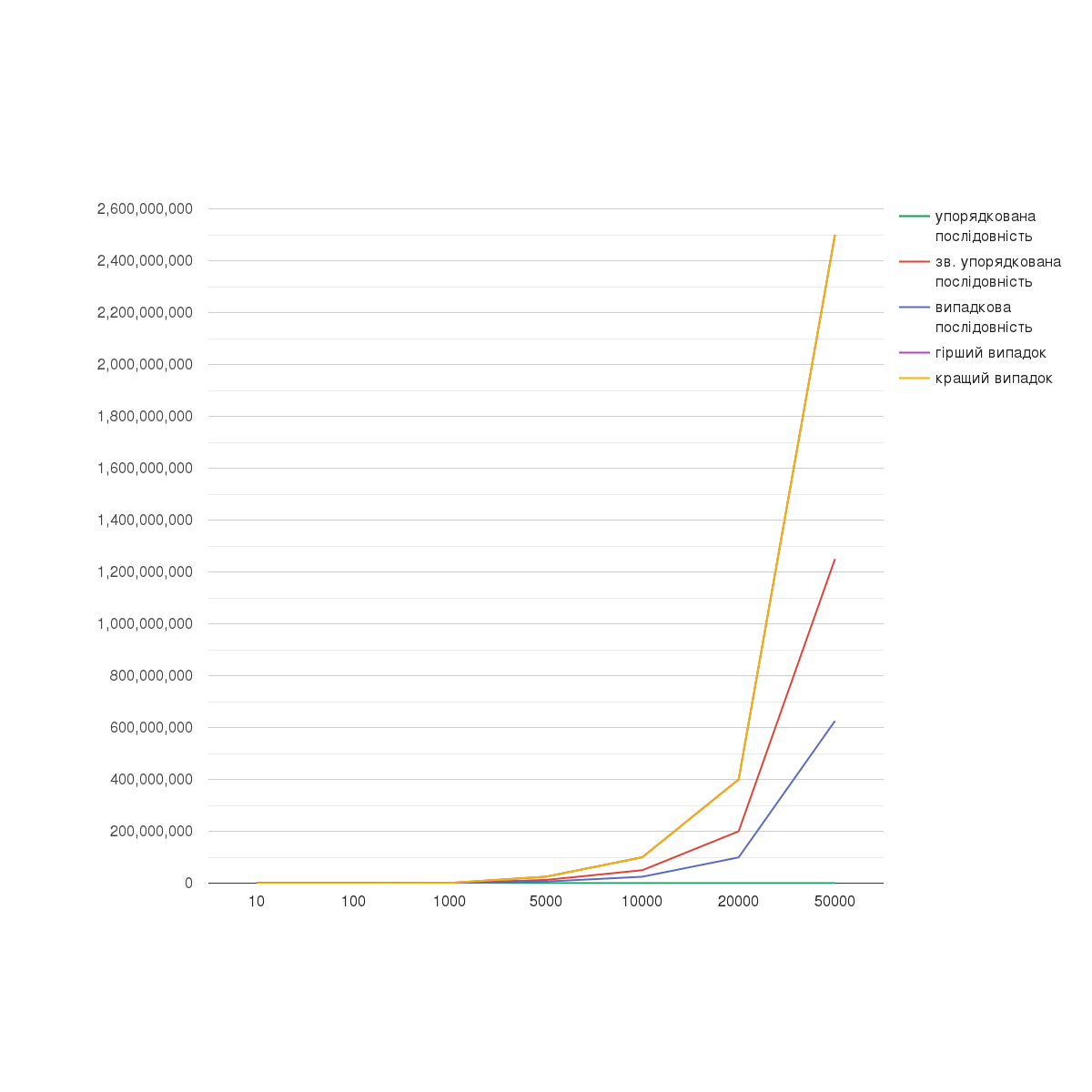
**Гребінець**

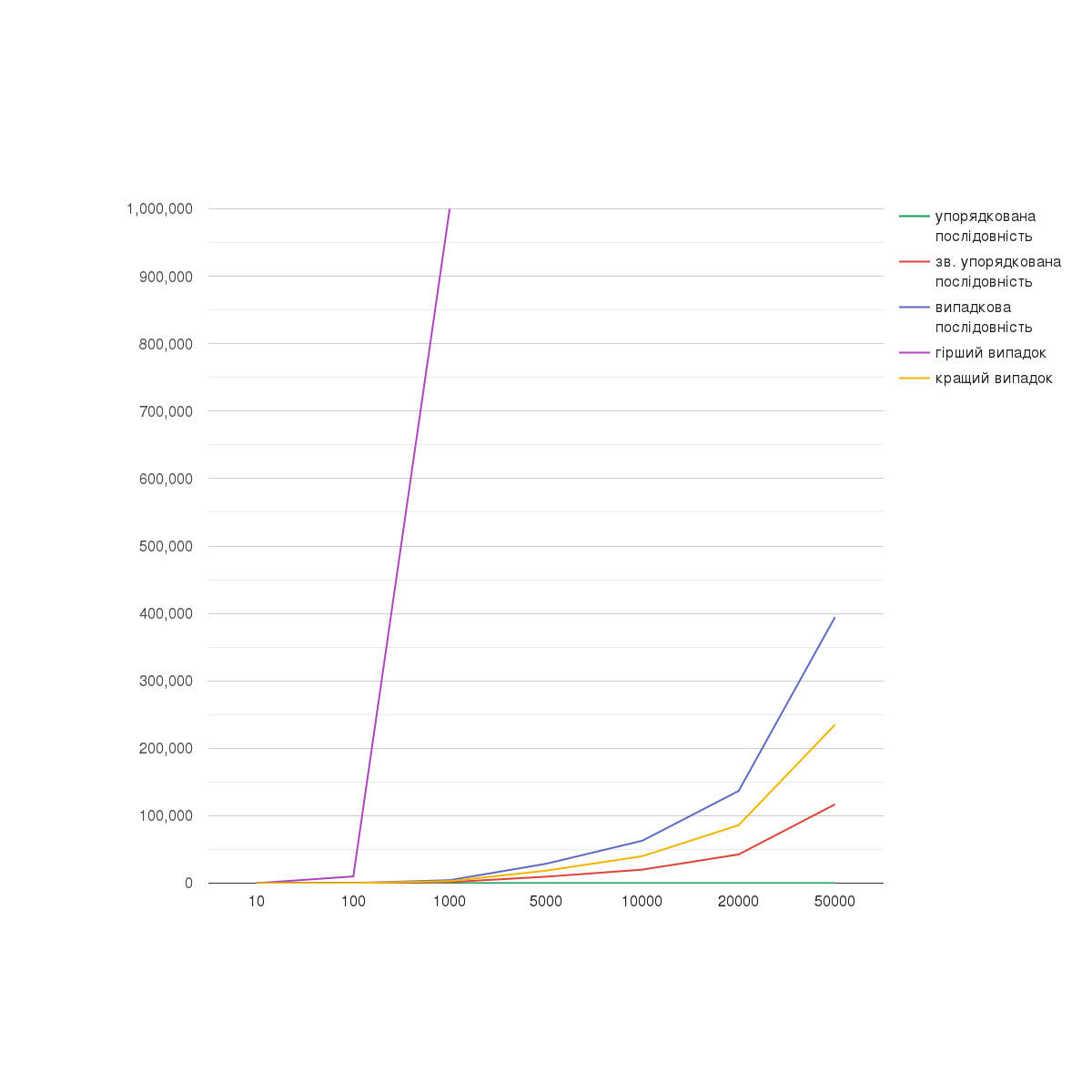
|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь у коді | Число перестановок у коді |
| 10 | 32 | 6 |
| 100 | 1201 | 247 |
| 1000 | 22709 | 4450 |
| 5000 | 143382 | 28724 |
| 10000 | 346732 | 62979 |
| 20000 | 673399 | 136816 |
| 50000 | 2083404 | 394554 |

### 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

**Бульбашка**

 **Гребінець**

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. Було помічено, що кількість порівнянь у звичайному алгоритмі «бульбашка» дорівнює , а у оптимізованому - , що також дорівнює кількості обмінів незалежно від реалізації у найгіршому випадку. Додатково можна було зробити перевірку на те, чи відбувся хоч один обмін у внутрішньому циклу, тоді алгоритм виконував би лише *(n-1)* порівнянь у кращому випадку, однак на практиці краще використовувати, інші, більш швидкодіючі алгоритми. При порівнянні «бульбашки» з «гребінцем» було встановлено, що вирішуючи проблему «черепах», сортування «гребінцем» виявляється набагато швидшим ніж сортування «бульбашкою», до того ж при зворотно впорядкованій послідовності елементів у масивах різниця стає ще більш суттєвою

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%.