**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)**

Скрипець Ольга Олександрівна

(шифр, прізвище, ім'я, по батькові)

**Перевірив**

(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1. **МЕТА ЛАБОРАТОРНОЇ РОБОТИ 3**
2. **ЗАВДАННЯ 4**
3. **ВИКОНАННЯ 5**
   1. АНАЛІЗ АЛГОРИТМУ НА ВІДПОВІДНІСТЬ ВЛАСТИВОСТЯМ 5
   2. ПСЕВДОКОД АЛГОРИТМУ 5
   3. АНАЛІЗ ЧАСОВОЇ СКЛАДНОСТІ 5
   4. ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ 5
      1. *Вихідний код 5*
      2. *Приклад роботи 5*
   5. ТЕСТУВАННЯ АЛГОРИТМУ 7
      1. *Часові характеристики оцінювання 7*
      2. *Графіки залежності часових характеристик оцінювання від розмірності масиву 9*

**ВИСНОВОК 10**

**КРИТЕРІЇ ОЦІНЮВАННЯ 11**

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 | Модифікований алгоритм бульбашки |
| 3 | Insertion\_sort |

3 ВИКОНАННЯ

* 1. Аналіз алгоритму на відповідність властивостям

Аналіз алгоритму сортування бульбашкою, Insertion\_sort на відповідність властивостям наведено в таблиці 3.1.

Таблиця 3.1 – Аналіз алгоритму на відповідність властивостям

|  |  |
| --- | --- |
| **Властивість** | **Сортування бульбашкою,**  **Insertion\_sort** |
| Стійкість(зберігає взаємний порядок однакових значень) | + |
| «Природність» поведінки  (Adaptability) | + |
| Базуються на порівняннях | + |
| Необхідність в додатковій пам'яті  (об'єм) | - |
| Необхідність в знаннях про структури  даних | + |

* 1. Псевдокод алгоритму

Bubble Sort



For i 1 to length(array)

 For j 1 to length(array)-i-1

If array[j] >array[j+1]

 temp array[j]

 array[j] array[j+1]

 array[j+1] temp

Modified Bubble Sort

Stop 0

For i 1 to length(array)

 For j 1 to length(array)-i-1

If array[j] >array[j+1]

 temp array[j]

 array[j] array[j+1]

 array[j+1] temp

 stop -1

if stop==0

break

Insertion\_Sort

For step 1 to length(array)

 Do key array[step]

 j step - 1

while j >0 and key < array[j]:

 array[j + 1] array[j]

j j - 1

 array[j + 1] key

* 1. Аналіз часової складності

Bubble\_Sort

Найкращий випадок складність: O(n)

Середній випадок складність: O()

Найгірший випадок складність: O()

Modified Bubble Sort

Найкращий випадок складність: O(n)

Середній випадок складність: O()

Найгірший випадок складність: O()

Insertion\_Sort

Найкращий випадок складність: O(n)

Середній випадок складність: O()

Найгірший випадок складність: O()

* 1. Програмна реалізація алгоритму
     1. Вихідний код

Bubble\_Sort

import numpy

def bubbleSort(array):

for i in range(len(array)):

for j in range(0, len(array) - i - 1):

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

temp = array[j]

array[j] = array[j+1]

array[j+1] = temp

print("------Сортування бульбашкою------")

print("Масив 10 el")

a = [1, 8, 5, 4, 7, 765, 56, 4567, 76, 2]

print(a)

print("Відсортований масив 10 el")

bubbleSort(a)

print(a)

print("Відсортований масив 10 el в зворотньому порядку")

print(a[::-1])

print("----------------------------------")

print("Масив 100 el")

b = numpy.random.randint(100000, size=100)

print(b)

print("Відсортований масив 100 el")

bubbleSort(b)

print(b)

print("Відсортований масив 100 el в зворотньому порядку")

print(b[::-1])

print("----------------------------------")

print("Масив 1000 el")

c = numpy.random.randint(100000, size=1000)

print(c)

print("Відсортований масив 1000 el")

bubbleSort(c)

print(c)

print("Відсортований масив 1000 el в зворотньому порядку")

print(c[::-1])

Modified Bubble Sort

import numpy

def bubbleSort(array):

for i in range(len(array)):

stop=0

for j in range(0, len(array) - i - 1):

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

temp = array[j]

array[j] = array[j+1]

array[j+1] = temp

stop=-1

if stop==0:

break

print("------Сортування бульбашкою------")

print("Масив 10 el")

a = [1, 8, 5, 4, 7, 765, 56, 4567, 76, 2]

print(a)

print("Відсортований масив 10 el")

bubbleSort(a)

print(a)

print("Відсортований масив 10 el в зворотньому порядку")

print(a[::-1])

print("----------------------------------")

print("Масив 100 el")

b = numpy.random.randint(100000, size=100)

print(b)

print("Відсортований масив 100 el")

bubbleSort(b)

print(b)

print("Відсортований масив 100 el в зворотньому порядку")

print(b[::-1])

print("----------------------------------")

print("Масив 1000 el")

c = numpy.random.randint(100000, size=1000)

print(c)

print("Відсортований масив 1000 el")

bubbleSort(c)

print(c)

print("Відсортований масив 1000 el в зворотньому порядку")

print(c[::-1])

Insertion\_Sort

import numpy

def insertionSort(array):

for step in range(1, len(array)):

key = array[step]

j = step - 1

while j >= 0 and key < array[j]:

array[j + 1] = array[j]

j = j - 1

array[j + 1] = key

print("------Insertion Sort------")

print("Масив 10 el")

a = [1, 8, 5, 4, 7, 765, 56, 4567, 76, 2]

print(a)

print("Відсортований масив 10 el")

insertionSort (a)

print(a)

print("Відсортований масив 10 el в зворотньому порядку")

print(a[::-1])

print("----------------------------------")

print("Масив 100 el")

b = numpy.random.randint(100000, size=100)

print(b)

print("Відсортований масив 100 el")

insertionSort (b)

print(b)

print("Відсортований масив 100 el в зворотньому порядку")

print(b[::-1])

print("----------------------------------")

print("Масив 1000 el")

c = numpy.random.randint(100000, size=1000)

print(c)

print("Відсортований масив 1000 el")

insertionSort (c)

print(c)

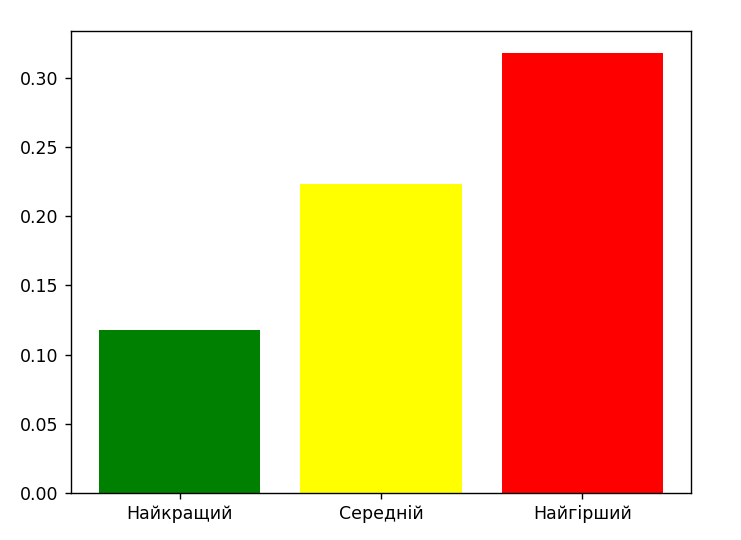
print("Відсортований масив 1000 el в зворотньому порядку")

print(c[::-1])

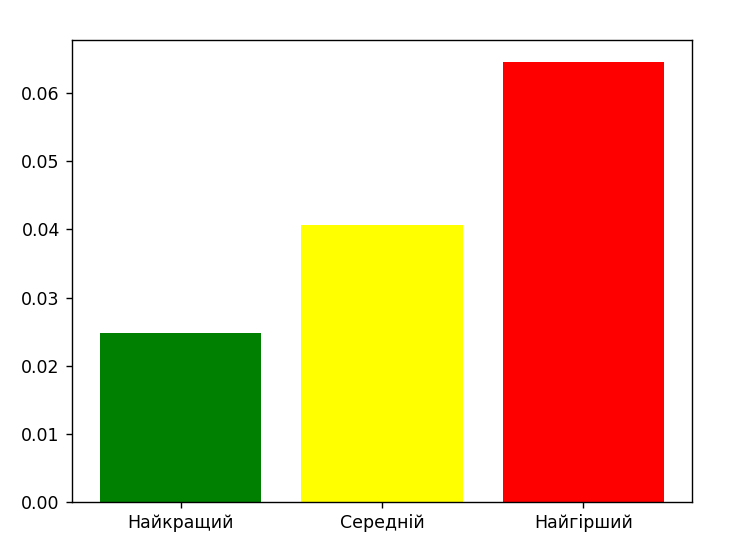
* + 1. Приклад роботи

На рисунках 3.1 і 3.2,3.3 показані приклади роботи програми сортування трьох алгоритмів для масивів на 10, 100 і 1000 елементів відповідно

10 Bubble\_sort



10 Modified Bubble Sort



10 Insertion sort

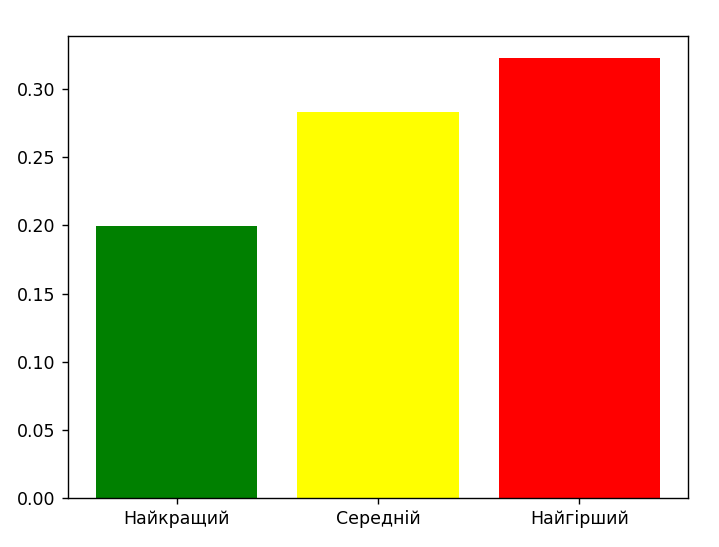
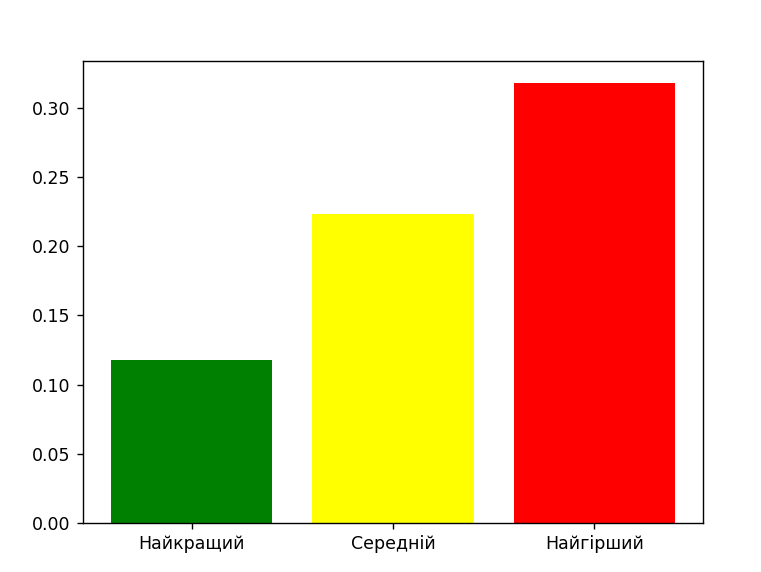
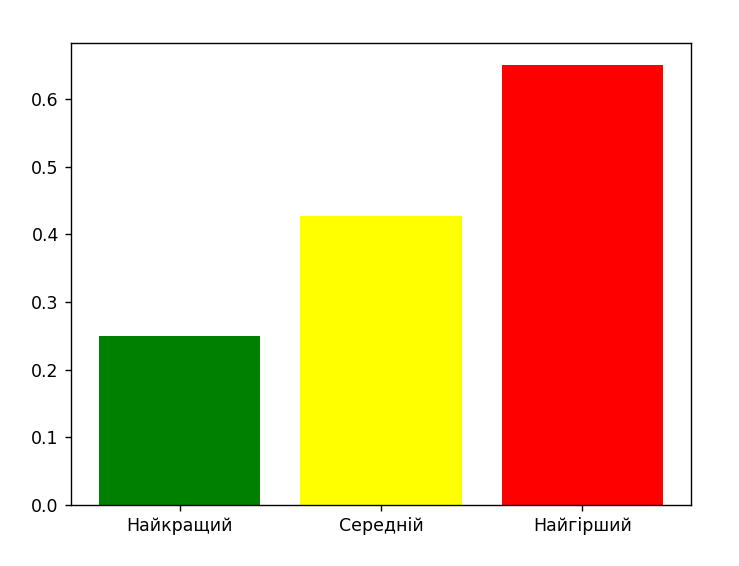


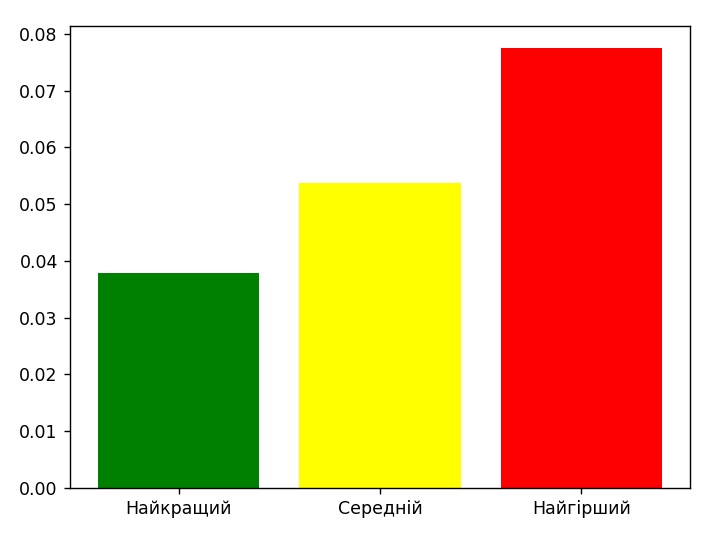
Рисунок 3.1 – Сортування масиву на 10 елементі

100 Bubble\_sort





100 Modified Bubble Sort



100 Insertion sort

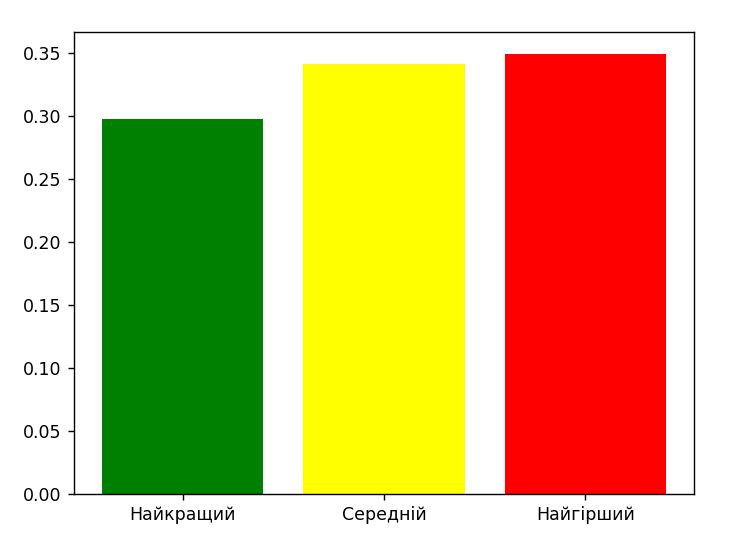
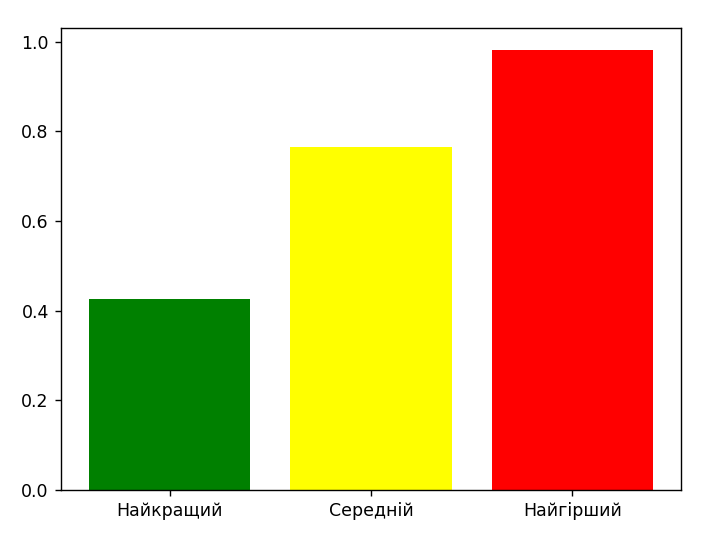
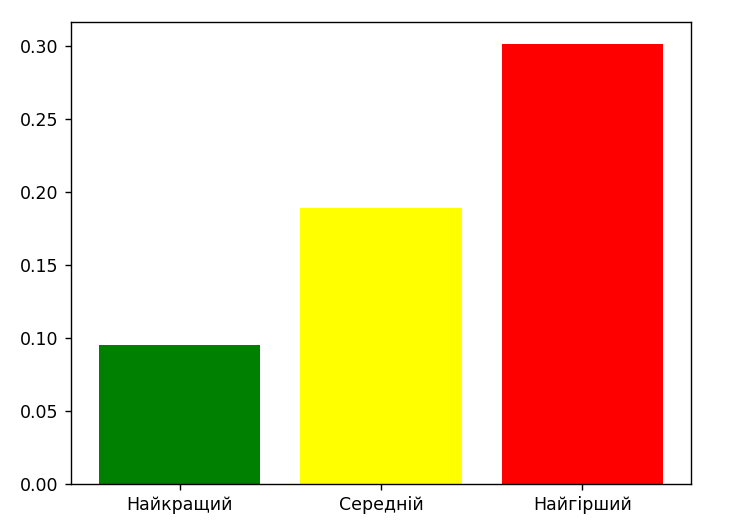


Рисунок 3.2 – Сортування масиву на 100 елементіd

1000 Bubble\_sort

1000 Modified Bubble Sort



1000 Insertion sort

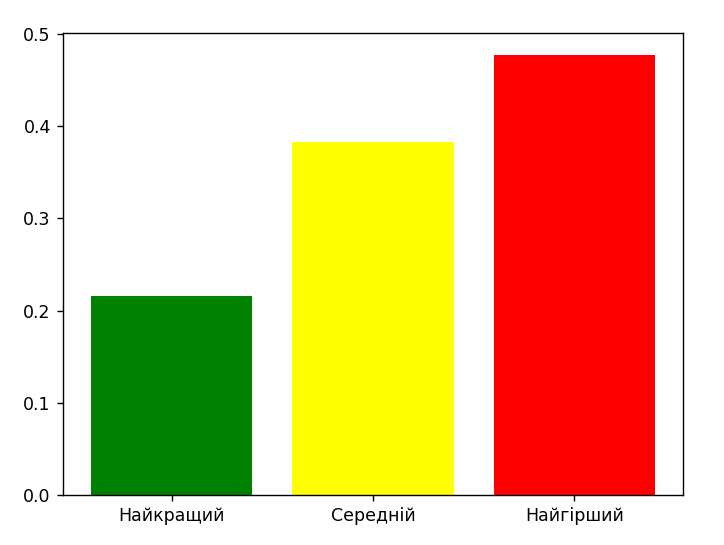


Рисунок 3.3 – Сортування масиву на 1000 елементів

* 1. Тестування алгоритму
     1. Часові характеристики оцінювання

В таблиці 3.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки, модифікованого алгоритму та Insertion\_sort для масивів різної розмірності, коли масив містить упорядковану послідовність елементів.

Таблиця 3.2 – Характеристики оцінювання алгоритму сортування бульбашки, модифікованого алгоритму та Insertion\_sort для упорядкованої послідовності елементів у масиві

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 9 9 | 0 0 0 |
| 100 | 4950 99 99 | 0 0 0 |
| 1000 | 499500 999 999 | 0 0 0 |
| 5000 | 12497500 4999 4999 | 0 0 0 |
| 10000 | 49995000 9999 9999 | 0 0 0 |
| 20000 | 199990000 19999 19999 | 0 0 0 |

В таблиці 3.3 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки, модифікованого алгоритму та Insertion\_sort для масивів різної розмірності, коли масиви містять зворотно упорядковану послідовність елементів.

Таблиця 3.3 – Характеристики оцінювання алгоритму сортування бульбашки, модифікованого алгоритму та Insertion\_sort для зворотно упорядкованої послідовності елементів у масиві.

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 45 54 | 45 45 45 |
| 100 | 4950 4950 5049 | 4950 4950 4950 |
| 1000 | 499500 499500 500599 | 499500 499500 499500 |
| 5000 | 12497500 12497500 12502499 | 12497500 12497500 12497500 |
| 10000 | 49995000 49995000 50004999 | 49995000 49995000 49995000 |
| 20000 | 199990000 199990000 200009999 | 199990000 199990000 199990000 |

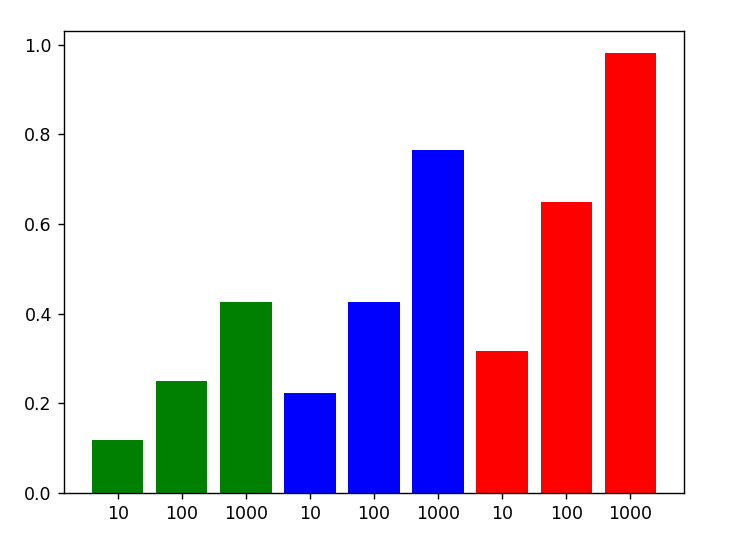
У таблиці 3.4 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки, модифікованого алгоритму та Insertion\_sort для масивів різної розмірності, масиви містять випадкову послідовність елементів.

Таблиця 3.4 – Характеристика оцінювання алгоритму сортування бульбашки, модифікованого алгоритму та Insertion\_sort для випадкової послідовності елементів у масиві.

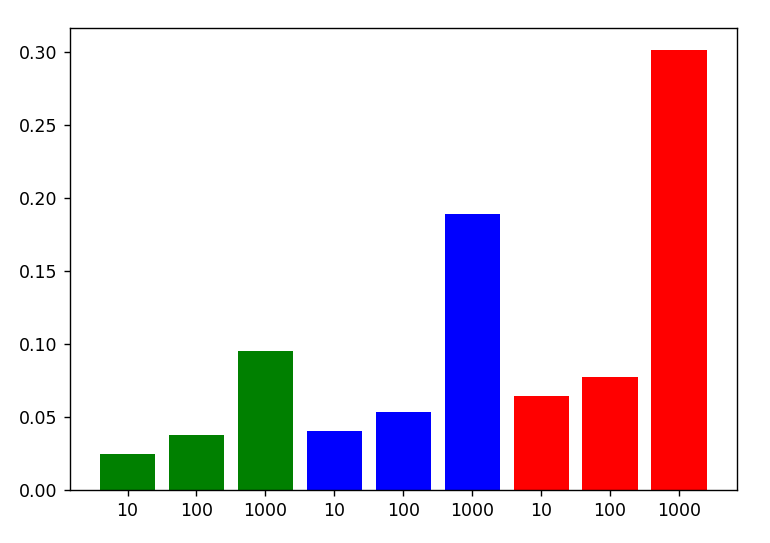
|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 42 23 | 16 29 23 |
| 100 | 4950 4797 2485 | 2473 2122 2485 |
| 1000 | 499500 499247 247273 | 245799 251857 247273 |
| 5000 | 12497500 12497269 6279402 | 6294914 6273382 6279402 |
| 10000 | 49995000 49984122 24886720 | 25015019 25192160 24886720 |
| 20000 | 199990000 199951219 100949799 | 199887658 100439730 100949799 |

* + 1. 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.

Bubble\_sort

Modified Bubble\_sort



Insertion sort

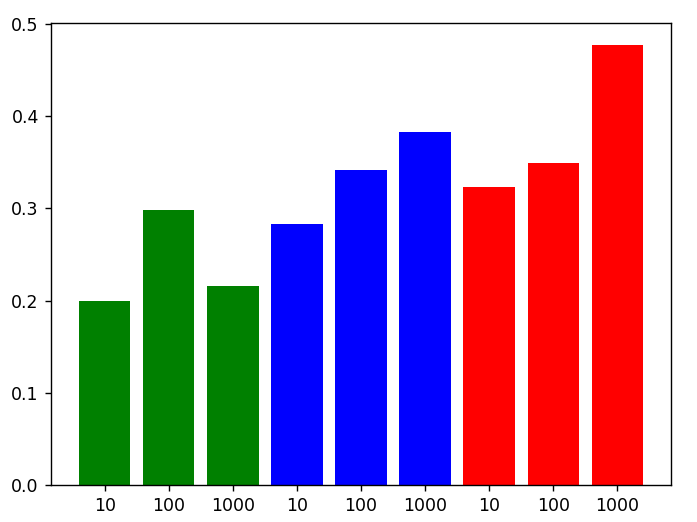


Рисунок 3.3 – Графіки залежності часових характеристик оцінювання

ВИСНОВОК

При виконанні даної лабораторної роботи я виконала аналіз алгоритму внутрішнього сортування на відповідність властивостям стійкості, природності поведінки, базування на порівняннях, необхідності додаткової пам’яті та знань про структуру даних. Також я записала алгоритм бульбашки, модифікований та сортування включенням за допомогою псевдокоду. Провела аналіз часової складності в гіршому, кращому і середньому випадках. І також виконала програмну реалізацію алгоритму на python. Зафіксувала часові характеристики оцінювання (кількість порівнянь, кількість перестановок). Також провела 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 побудувала графіки залежності часових характеристик оцінювання від розмірності масиву.

Завдяки цьому я побачила, на скільки модифікований алгоритм сортування бульбашкою сортує швидше ніж проста бульбашка та зрозуміла як залежить швидкість виконання сортування від величини масиву. Також побачила, що алгоритм сортування включенням має однакову кількість порівнянь та перестановок у середньому випадку. А у найкращому випадку кожен з алгоритмів робить нуль перестановок, в той час як порівняння відрізняються і звичайне сортування бульбашкою має набагато більше ітерацій, ніж модифіковане та сортування включенням.

КРИТЕРІЇ ОЦІНЮВАННЯ

У випадку здачі лабораторної роботи із захистом максимальний бал дорівнює – 5.

Критерії від максимального балу:

* аналіз алгоритму на відповідність властивостям ;
* псевдокод алгоритму ;
* аналіз часової складності ;
* програмна реалізація алгоритму ;
* тестування алгоритму ;
* висновок;
* захист роботи – 2 бали