**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-13 Козак Антон Миколайович*

**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 3.1.1

Таблиця 3.1.1 – Аналіз алгоритму на відповідність властивостям

|  |  |
| --- | --- |
| **Властивість** | **Сортування бульбашкою** |
| Стійкість | Алгоритм є стійким. |
| «Природність» поведінки (Adaptability) | Алгоритму не властива природність поведінки. |
| Базуються на порівняннях | Алгоритм належить до алгоритмів сортування порівняннями. |
| Необхідність в додатковій пам'яті (об'єм) | Просторова складність алгоритму оцінюється в |
| Необхідність в знаннях про структури даних | Використання алгоритму потребує знання про розмір структури даних. |

Аналіз алгоритму сортування гребінцем на відповідність властивостям наведено в таблиці 3.1.2

Таблиця 3.1.2 – Аналіз алгоритму на відповідність властивостям

|  |  |
| --- | --- |
| **Властивість** | **Сортування гребінцем** |
| Стійкість | Алгоритм не є стійким. |
| «Природність» поведінки (Adaptability) | Алгоритму не властива природність поведінки. |
| Базуються на порівняннях | Алгоритм належить до алгоритмів сортування порівняннями. |
| Необхідність в додатковій пам'яті (об'єм) | Просторова складність алгоритму оцінюється в |
| Необхідність в знаннях про структури даних | Використання алгоритму потребує знання про розмір структури даних. |

## Псевдокод алгоритму

Сортування бульбашкою:

bubbleSort(A)

n = length(A)

**do**

swapped = false

**for** i = 1 **to** n-1 **do**

**if** A[i]>a[i+1] **then**

temp = A[i]

A[i] = A[i+1]

A[i+1] = temp

swapped = true

**end if**

**end for**

**while** swapped == true

Сортування гребінцем:

combSort(A)

gap = length(A)

shrink = 1.3

**do**

gap = floor(gap/shrink)

**if** gap<=1 **then**

gap = 1

sorted = true

**end** **if**

i = 0

**while** i + gap < length(A) **do**

**if** A[i] > A[i+gap] **then**

temp = A[i]

A[i] = A[i+gap]

A[i+gap] = temp

sorted = false

**end** **if**

i=i+1

**end while**

**while** sorted == false

## Аналіз часової складності

Сортування бульбашкою:

Найсприятливішим випадком для алгоритму сортування бульбашкою буде впорядкований масив. Найменш сприятливим – обернено впорядкований. Кількість перевірок виконання умови внутрішнього циклу for завжди буде рівною . Кількість перевірок умови зовнішнього циклу while буде рівною 1 в кращому випадку та в гіршому. Таким чином,

Сортування гребінцем:

Найсприятливішим випадком для алгоритму сортування гребінцем буде впорядкований масив. Кількість перевірок виконання умови внутрішнього циклу while на певшій ітерації зовнішнього циклу буде визначатися формулою , де ­– номер ітерації. Мінімальна кількість перевірок умови зовнішнього циклу while буде рівною . У випадку недоцільного підбору параметру shrink максимальна перевірок умови зовнішнього циклу while буде рівною . Нехай , таким чином,

## Програмна реалізація алгоритму на мові Python

### Вихідний код

Сортування бульбашкою:

def bubbleSort(arr, size):

compares=0

swaps=0

swapped = True

while swapped:

swapped = False

for i in range(size-1):

compares+=1

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

swaps+=1

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

swapped = True

return compares, swaps

Сортування гребінцем:

def combSort(arr, size):

compares=0

swaps=0

gap=size

sorted = False

shrink = 1.3

while not sorted:

gap = int(gap//shrink)

if gap<=1:

gap=1

sorted = True

for i in range(size-gap):

compares+=1

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

swaps+=1

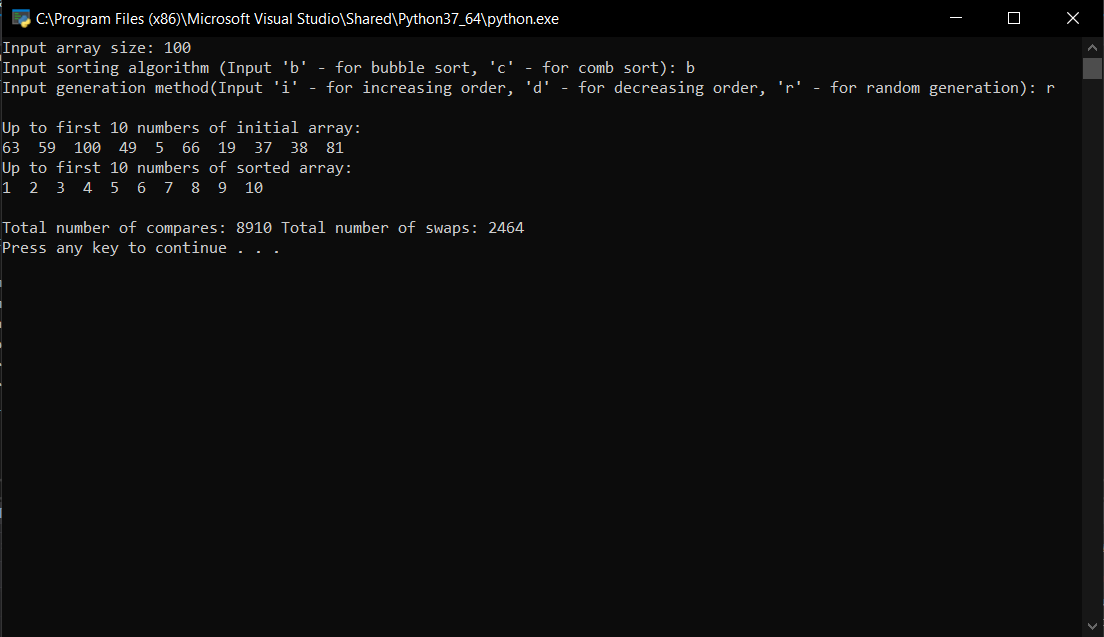
arr[i], arr[i+gap]=arr[i+gap], arr[i]

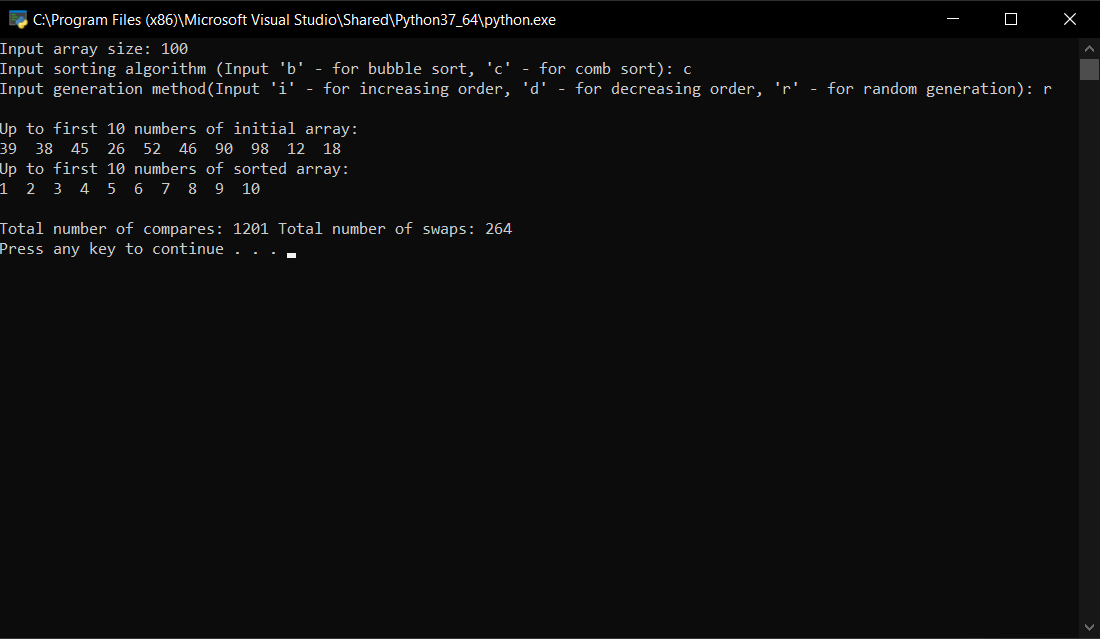
sorted = False

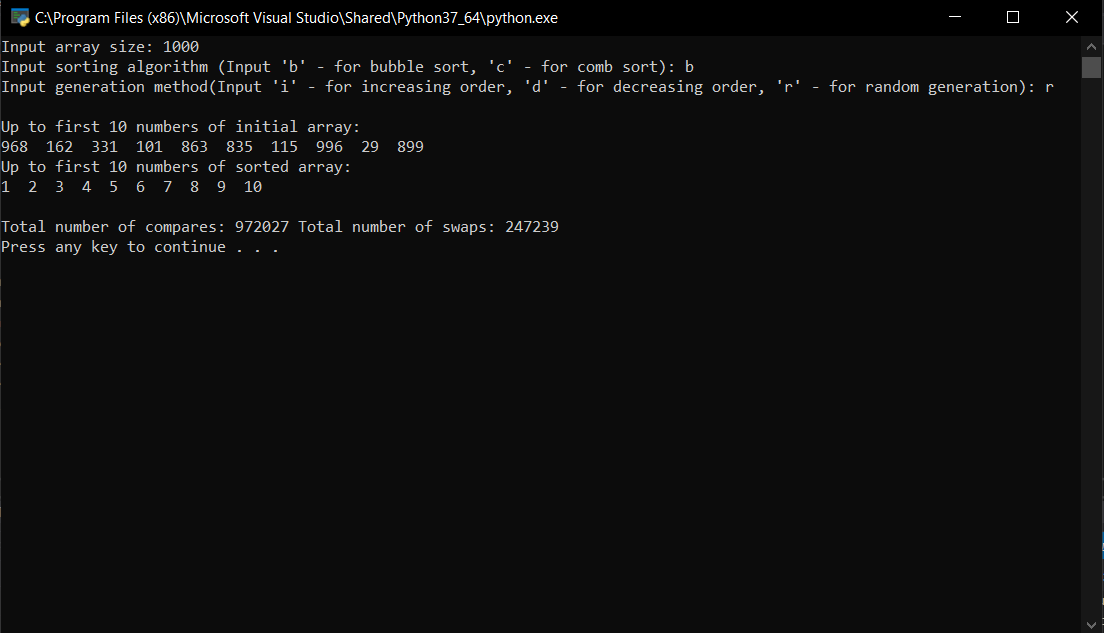
return compares, swaps

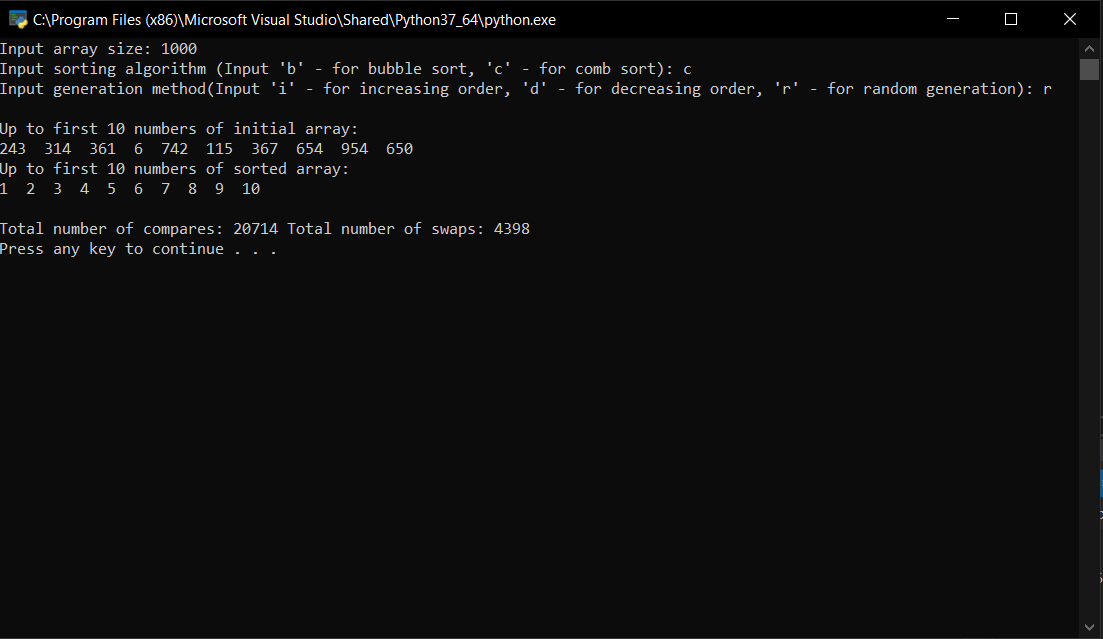
### Приклад роботи

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 алгоритмів сортування бульбашки та гребінця для масивів різної розмірності, коли масив містить упорядковану послідовність елементів.

Таблиця 3.2 – Характеристики оцінювання алгоритмів сортування бульбашки та гребігця для упорядкованої послідовності елементів у масиві

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Розмірність масиву | Число порівнянь | | Число перестановок | |
| Бульбашка | Гребінець | Бульбашка | Гребінець |
| 10 | 9 | 32 | 0 | 0 |
| 100 | 99 | 122 | 0 | 0 |
| 1000 | 999 | 18716 | 0 | 0 |
| 5000 | 4999 | 123395 | 0 | 0 |
| 10000 | 9999 | 276745 | 0 | 0 |
| 20000 | 19999 | 593412 | 0 | 0 |
| 50000 | 49999 | 1683424 | 0 | 0 |

В таблиці 3.3 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритмів сортування бульбашки та гребінця для масивів різної розмірності, коли масиви містять зворотно упорядковану послідовність елементів.

Таблиця 3.3 – Характеристики оцінювання алгоритмів сортування бульбашки та гребінця для зворотно упорядкованої послідовності елементів у масиві.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Розмірність масиву | Число порівнянь | | Число перестановок | |
| Бульбашка | Гребінець | Бульбашка | Гребінець |
| 10 | 90 | 41 | 45 | 7 |
| 100 | 9900 | 1102 | 4950 | 122 |
| 1000 | 999000 | 19715 | 499500 | 1572 |
| 5000 | 24995000 | 128394 | 12497500 | 9552 |
| 10000 | 99990000 | 286744 | 49995000 | 20050 |
| 20000 | 399980000 | 613411 | 199990000 | 43762 |
| 50000 | 2499950000 | 1733423 | 1249975000 | 115908 |

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

Таблиця 3.4 – Характеристика оцінювання алгоритмів сортування бульбашки та гребінця для випадкової послідовності елементів у масиві.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Розмірність масиву | Число порівнянь | | Число перестановок | |
| Бульбашка | Гребінець | Бульбашка | Гребінець |
| 10 | 72 | 41 | 24 | 9 |
| 100 | 9504 | 1300 | 2596 | 271 |
| 1000 | 940059 | 20714 | 256463 | 4339 |
| 5000 | 24765045 | 143391 | 6238068 | 28864 |
| 10000 | 88774400 | 326740 | 26175600 | 63679 |
| 20000 | 355148800 | 713406 | 104706755 | 139231 |
| 50000 | 2219872000 | 2033417 | 654433555 | 385395 |

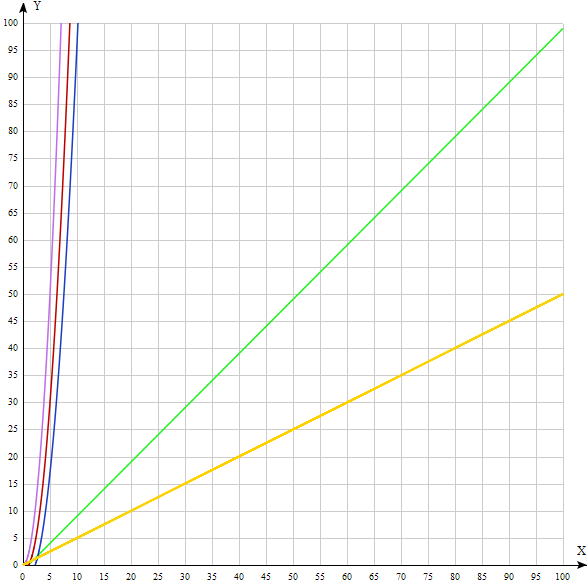
### Графіки залежності часових характеристик оцінювання від розмірності масиву

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

Сортування бульбашкою:

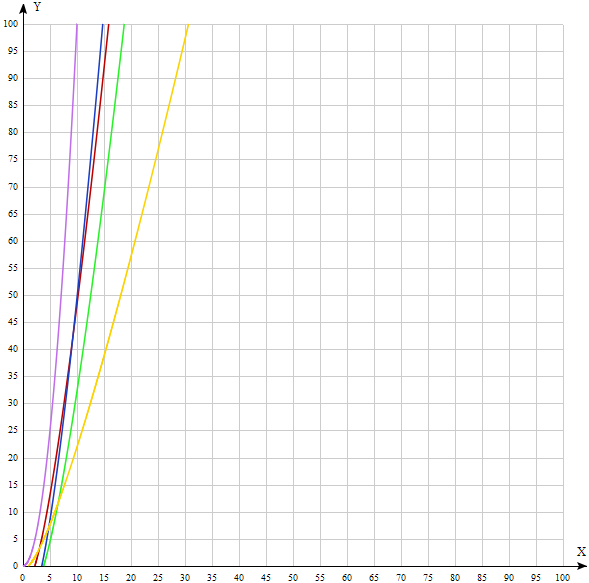
Кількість дій (порівнянь та престановок)



2n2

Розмір масиву

0.5n

Сортування гребінцем:

n2

0.25nlog1.3n

Розмір масиву

Кількість дій (порівнянь та престановок)

Висновок

При виконанні даної лабораторної роботи я вивчив основні методи аналізу обчислювальної складності алгоритмів внутрішнього сортування і оцінив поріг їх ефективності. Виконав програмну реалізацію відповідних алгоритмів та порівняв темпи підвищення часової складності відносно асимптотичних оцінок з використанням графіків, які відповідають різним початковим станам відсортованості масиву.

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%.