**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-1425 Радзівіло Валерія Артемівна Артемінвна*

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

|  |  |
| --- | --- |
| **Властивість** | **Сортування гребінцем** |
| Стійкість | - |
| «Природність» поведінки (Adaptability) | + |
| Базуються на порівняннях | + |
| Необхідність в додатковій пам'яті (об'єм) | - |
| Необхідність в знаннях про структури даних | - |

## Псевдокод алгоритму

Бульбашкою:

**for** i = 0 to length [arr] -1

**for** j =0 to length [arr] – 1

**if** arr[j] < arr[j+1]

**do**

tmp = arr[j]

arr[j] = arr[j+1]

arr[j+1] = tmp

Гребінцем:

swaps = True

gap = length [arr]

**while** gap > 1 or swaps:

gap = max(1, int(gap/1.25))

swaps = False

**for** i = 0 to length[arr] – gap

j = i + gap

**if** arr[i] > arr[j]

**do**

tmp = arr[j]

arr[j] = arr[i]

arr[i] = tmp

swaps = True

## Аналіз часової складності

Бульбашкою:

Гірший : O(n2)

Кращий: O(n)

Гребінцем:

Гірший : O(n2)

Кращий: O(nlogn)

## Програмна реалізація алгоритму

### Вихідний код

Бульбашкою:

import random  
  
  
def bubble\_sort(arr):  
 n = len(arr)  
 for i in range(n-1):  
 for j in range(n-1):  
 if arr[j]>arr[j+1]:  
 arr[j], arr[j+1] = arr[j+1], arr[j]  
  
  
n = int(input("Input size: "))  
variant = input("Choose array( rand, sorted, back): ")  
if variant == "rand":  
 arr = random.sample(range(1, n+1), n)  
if variant == "sorted":  
 arr = list(range(1, n+1))  
if variant == "back":  
 arr = sorted(range(1, n + 1), reverse=True)  
  
print("Before sort: ")  
print(\*arr)  
bubble\_sort(arr)  
print("After sort:")  
print(\*arr)

Гребінцем:

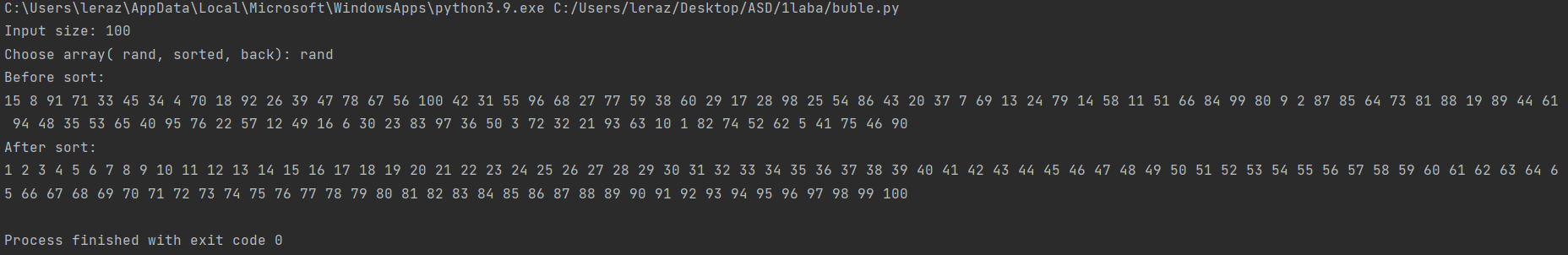
import random  
  
  
def comb\_sort(arr):  
 poriv = 0  
 perest = 0  
 gap = len(arr)  
 swaps = True  
 while gap > 1 or swaps:  
 gap = max(1, int(gap/1.25))  
  
 swaps = False  
 for i in range(len(arr) - gap):  
 poriv += 1  
 j = i+gap  
 if arr[i] > arr[j]:  
 perest += 1  
 arr[i], arr[j]=arr[j], arr[i]  
 swaps = True  
 return [poriv, perest]  
  
  
def arr\_statistics(res):  
 print("Number of comparisons " + str(res[0]) + ", " + "number of permutations " + str(res[1]) + "\n")  
  
  
n = int(input("Input size: "))  
variant = input("Choose array( rand, sorted, back): ")  
if variant == "rand":  
 arr = random.sample(range(1, n+1), n)  
if variant == "sorted":  
 arr = list(range(1, n+1))  
if variant == "back":  
 arr = sorted(range(1, n + 1), reverse=True)  
  
print("Before sort: ")  
print(\*arr)  
result = comb\_sort(arr)  
print("After sort:")  
print(\*arr)  
arr\_statistics(result)

### 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

Бульбашка:



Гребінець:

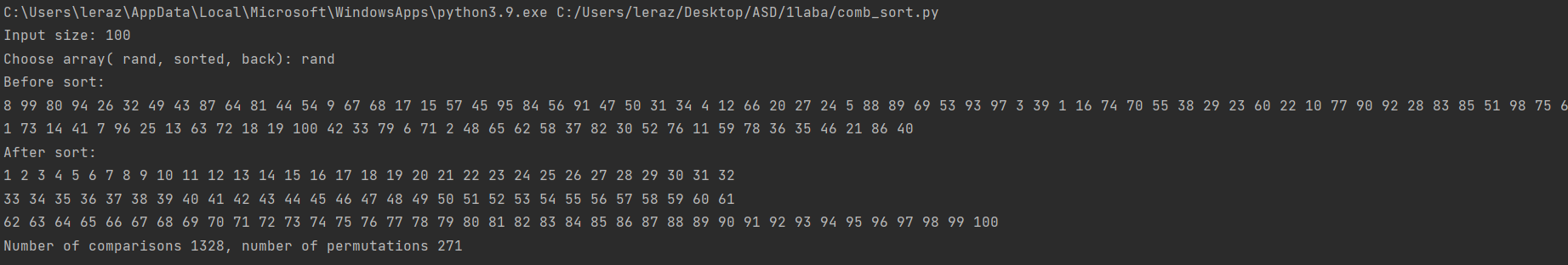
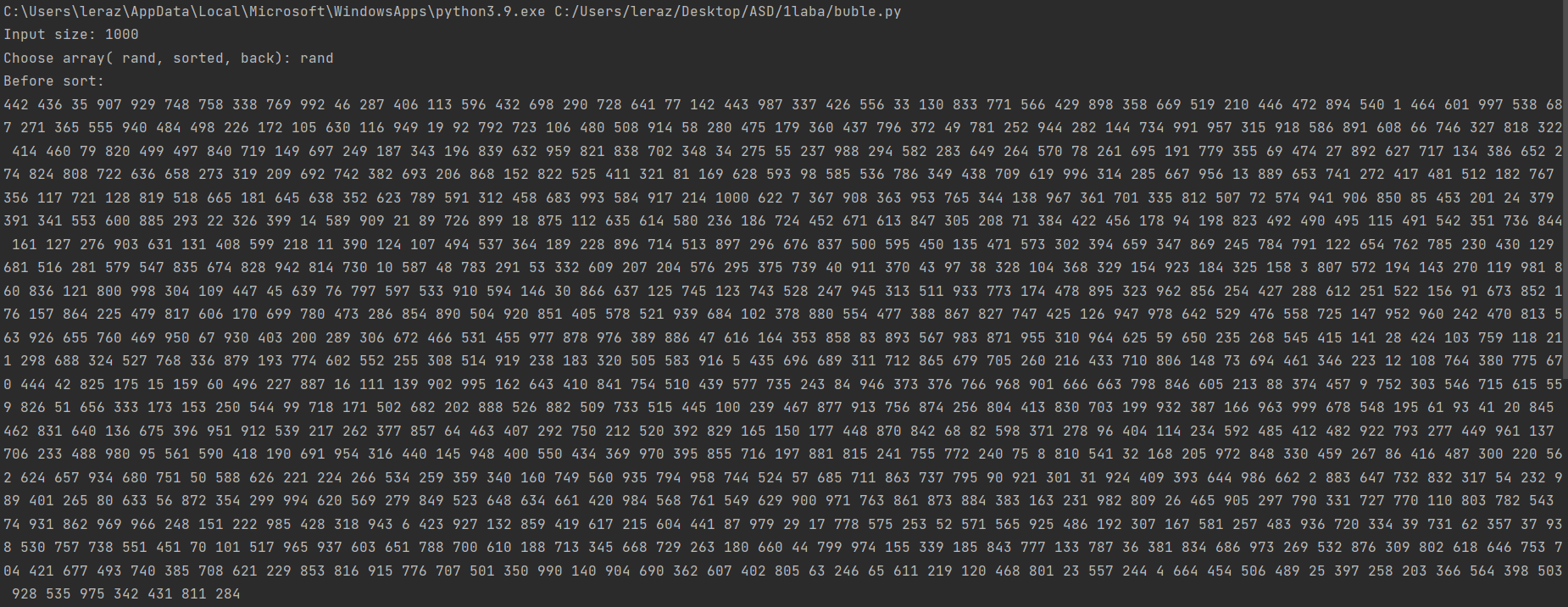


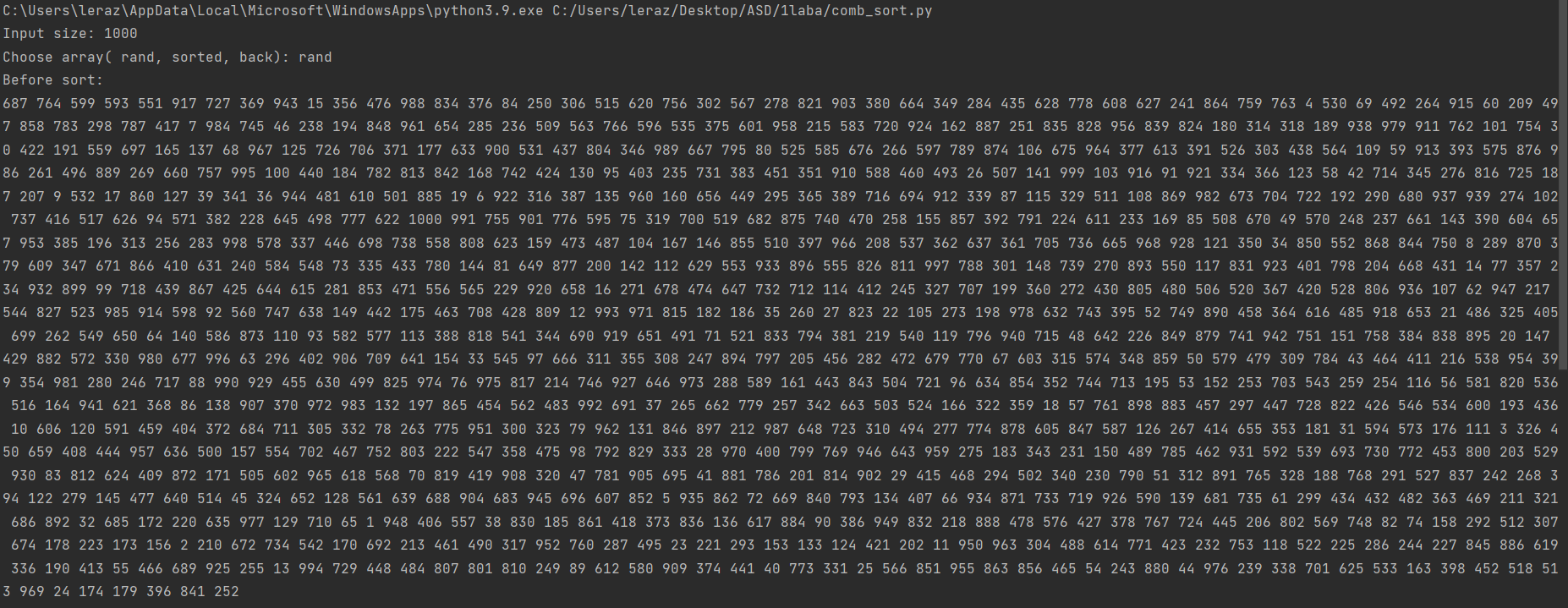
Рисунок 3.2 – Сортування масиву на 1000 елементів

Бульбашкою:





Гребінцем:



## 

## 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 бульбашки для упорядкованої послідовності елементів у масиві

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 0 |
| 100 | 4950 | 0 |
| 1000 | 499500 | 0 |
| 5000 | 12497500 | 0 |
| 10000 | 49995000 | 0 |
| 20000 | 199990000 | 0 |
| 50000 | 1249975000 | 0 |

Гребінцем:

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 36 | 0 |
| 100 | 1229 | 0 |
| 1000 | 22056 | 0 |
| 5000 | 145060 | 0 |
| 10000 | 320071 | 0 |
| 20000 | 700092 | 0 |
| 50000 | 2000058 | 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 бульбашки для зворотно упорядкованої послідовності елементів у масиві.

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 45 |
| 100 | 4950 | 4950 |
| 1000 | 499500 | 499500 |
| 5000 | 12497500 | 12497500 |
| 10000 | 49995000 | 49995000 |
| 20000 | 199990000 | 199990000 |
| 50000 | 1249975000 | 1249975000 |

Гребінцем:

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 9 |
| 100 | 1328 | 110 |
| 1000 | 23055 | 1514 |
| 5000 | 150059 | 9434 |
| 10000 | 330070 | 19740 |
| 20000 | 720091 | 42174 |
| 50000 | 2050057 | 110230 |

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 | 22 |
| 100 | 4950 | 2495 |
| 1000 | 499500 | 242122 |
| 5000 | 12497500 | 6238250 |
| 10000 | 49995000 | 24722446 |
| 20000 | 199990000 | 99823429 |
| 50000 | 1249975000 | 622091815 |

Гребінець:

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 9 |
| 100 | 1328 | 274 |
| 1000 | 24054 | 4184 |
| 5000 | 155058 | 27662 |
| 10000 | 350068 | 60661 |
| 20000 | 780088 | 133334 |
| 50000 | 2100056 | 368004 |

### 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

Бульбашкою:

Коли масиви містять упорядковану послідовність елементів

Коли масиви містять зворотно упорядковану послідовність елементів

Коли масиви містять випадкову послідовність елементів

Гребінець:

Коли масиви містять упорядковану послідовність елементів

Коли масиви містять зворотно упорядковану послідовність елементів

Коли масиви містять випадкову послідовність елементів

Висновок

При виконанні даної лабораторної роботи…

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%.