**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, Мельник Михайло Олександрович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Cопов Олексій Олександрович*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc69772242)

[2 ЗаВдання 4](#_Toc69772243)

[3 Виконання 5](#_Toc69772244)

[3.1 Аналіз алгоритму на відповідність властивостям 5](#_Toc69772245)

[3.2 Псевдокод алгоритму 5](#_Toc69772246)

[3.3 Аналіз часової складності 6](#_Toc69772247)

[3.4 Програмна реалізація алгоритму 8](#_Toc69772248)

[3.4.1 Вихідний код 8](#_Toc69772249)

[3.4.2 Приклад роботи 9](#_Toc69772250)

[3.5 Тестування алгоритму 11](#_Toc69772251)

[3.5.1 Часові характеристики оцінювання 11](#_Toc69772252)

[3.5.2 Графіки залежності часових характеристик оцінювання від розмірності масиву 14](#_Toc69772253)

[Висновок 15](#_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 на відповідність властивостям наведено в таблиці 3.1.

Таблиця 3.1 – Аналіз алгоритму на відповідність властивостям

|  |  |  |
| --- | --- | --- |
| **Властивість** | **Сортування бульбашкою** | **Сортування гребінцем** |
| Стійкість | Стійкий | Стійкий |
| «Природність» поведінки (Adaptability) | Неприродний | Неприродний |
| Базуються на порівняннях | Так | Так |
| Необхідність в додатковій пам'яті (об'єм) | O(1) | O(1) |
| Необхідність в знаннях про структури даних | Масив | Масив |

## Псевдокод алгоритму

1. Bubble sort

Bubble\_sort(A)

1 for i := 0 to length(A)-1

2 for j := 0 to length(A) – 1

3 if A[i] > A[i+1]

4 swap(A[i], A[i+1])

1. Comb sort

Comb\_sort(A)

1 factor := 1.2473309

2 step := length(A) – 1

3 while step >= 1

4 for i := 0 to length(A) - step

5 if A[i] > A[i+1]

6 swap(A[i], A[i+1]

7 step := step div factor

## Аналіз часової складності

|  |  |  |
| --- | --- | --- |
| Bubble\_sort(A) | час | кількість разів |
| 1 for j := 1 to length(A) - 1  2 for i := 1 to length(A) – 1  3 if A[i] > A[i+1]  4 swap(A[i], A[i+1]) | с1  с2  с3  с4 | n  n2  (n-1)2 |

1. **Bubble sort**

Для кожного j позначимо кількість сприятливих перевірок у if як tj.

Таким чином, час роботи алгоритму:

Для алгоритму сортування бульбашкою найсприятливіший випадок — коли усі елементи масиву уже відсортовані, тоді :

Найгірший випадок — коли усі елементи масиву відсортовані у протилежному порядку, тоді :

У середньому випадку, очевидно, отримаємо O(n2) за кількістю перевірок.

Таким чином, для сортування бульбашкою

1. **Comb sort**

Позначимо за tstep кількість сприятливих перевірок у if.

Кількість перевірок умови у циклі for дорівнює кількості перевірок умови у циклі while, помножені на n, мінус сума геометричної прогресії з b1=n-1 & q = 1/f.

|  |  |  |
| --- | --- | --- |
| Comb\_sort(A) | час | кількість разів |
| 1 f := 1.2473309  2 step := length(A) – 1  3 while step >= 1  4 for i := 1 to length(A) - step  5 if A[i] > A[i+step]  6 swap(A[i], A[i+step]  7 step := step div f | с1  с2  с3  с4  c5  c6  c7 | 1  1 |

Таким чином, час роботи алгоритму:

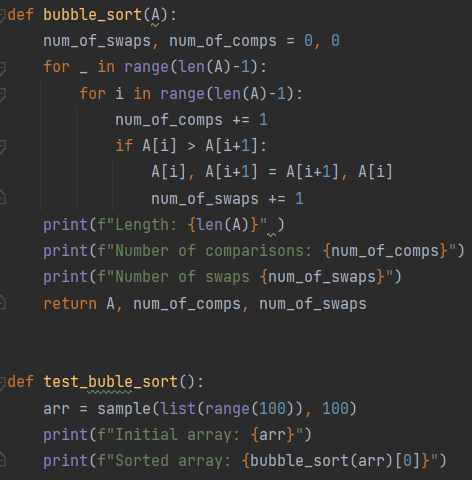
Для алгоритму сортування гребінцем найсприятливіший випадок — коли усі елементи масиву уже відсортовані, тоді :

Доведення часової складності у найгіршому та середньому випадку доводиться методом, що заснований на Колмогоровській складності (див. Analysis of Sorting Algorithms by Kolmogorov Complexity (A Survey)), і дорівнює O(n2) для найгіршого випадку.

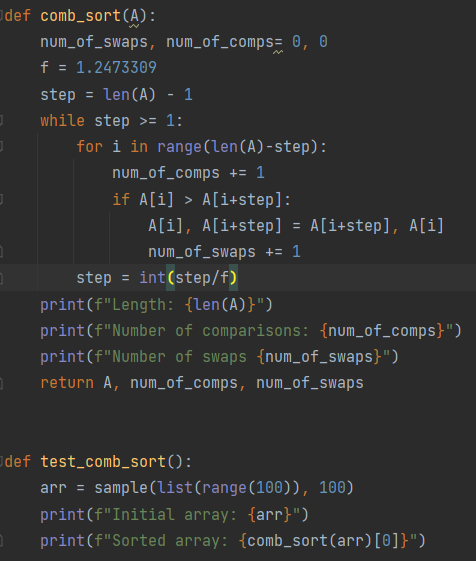
## Програмна реалізація алгоритму

### Вихідний код (Python)

1. **Bubble sort**



1. **Comb sort**



### Приклад роботи

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

1. Bubble sort

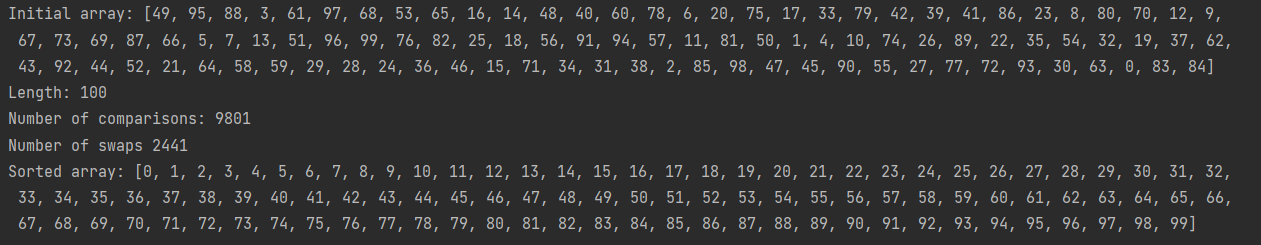


Рисунок 3.1.1 – Сортування масиву на 100 елементів, bubble sort

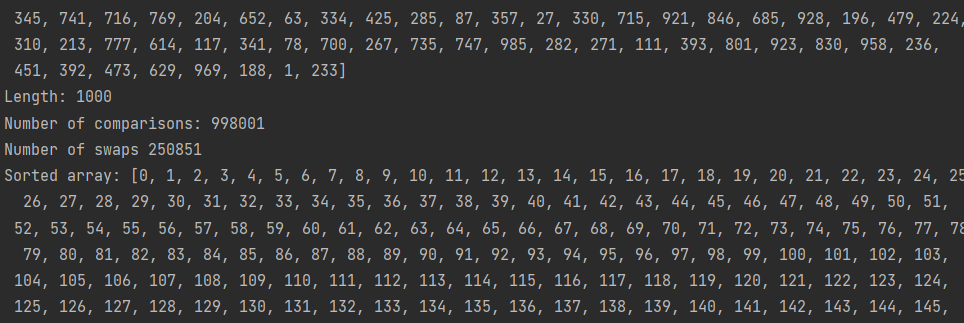


Рисунок 3.1.2 – Сортування масиву на 1000 елементів, bubble sort

1. Comb sort

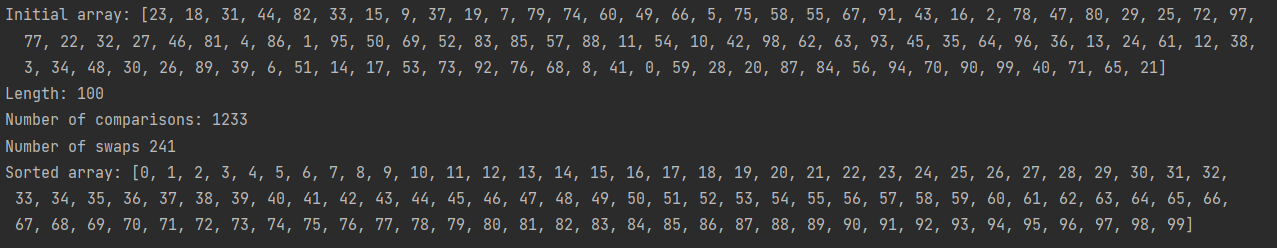


Рисунок 3.2.1 – Сортування масиву на 100 елементів, comb sort

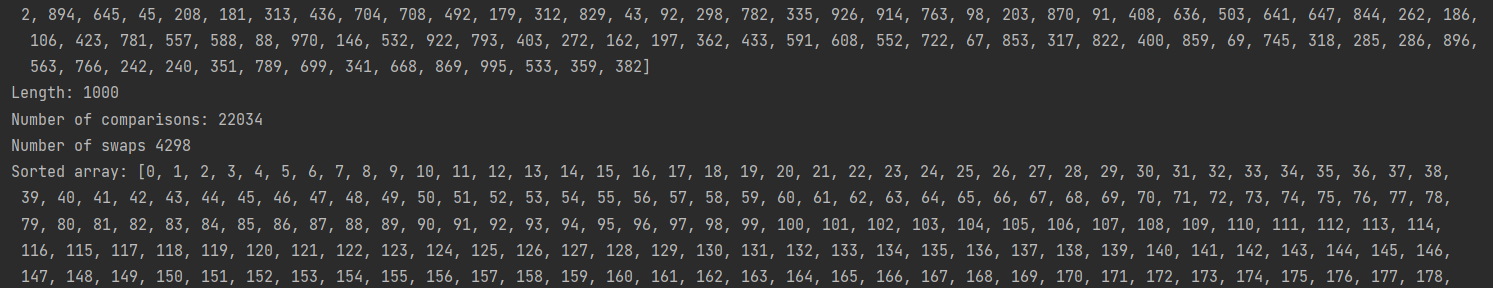


Рисунок 3.2.2 – Сортування масиву на 1000 елементів, comb sort

## Тестування алгоритму

### Часові характеристики оцінювання

В таблиці 3.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритмів сортування 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 | 9801 | 0 |
| 1000 | 998001 | 0 |
| 5000 | 24990001 | 0 |
| 10000 | 99980001 | 0 |
| 20000 | 399960001 | 0 |
| 50000 | 2499900001 | 0 |

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

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 39 | 0 |
| 100 | 1233 | 0 |
| 1000 | 22034 | 0 |
| 5000 | 144865 | 0 |
| 10000 | 329653 | 0 |
| 20000 | 719246 | 0 |
| 50000 | 1997961 | 0 |

В таблиці 3.3 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритмів сортування 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 | 9801 | 4950 |
| 1000 | 998001 | 499500 |
| 5000 | 24990001 | 12497500 |
| 10000 | 99980001 | 49995000 |
| 20000 | 399960001 | 199990000 |
| 50000 | 2499900001 | 1249975000 |

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

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 39 | 5 |
| 100 | 1233 | 106 |
| 1000 | 22034 | 1528 |
| 5000 | 144865 | 9110 |
| 10000 | 329653 | 19164 |
| 20000 | 719246 | 40636 |
| 50000 | 1997961 | 109794 |

У таблиці 3.4 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритмів сортування 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 | 21 |
| 100 | 9801 | 2106 |
| 1000 | 998001 | 250399 |
| 5000 | 24990001 | 6287052 |
| 10000 | 99980001 | 24682084 |
| 20000 | 399960001 | 100395533 |
| 50000 | 2499900001 | 622845426 |

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

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 39 | 6 |
| 100 | 1233 | 263 |
| 1000 | 22034 | 4419 |
| 5000 | 144865 | 27676 |
| 10000 | 329653 | 60312 |
| 20000 | 719246 | 131227 |
| 50000 | 1997961 | 368894 |

### 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.

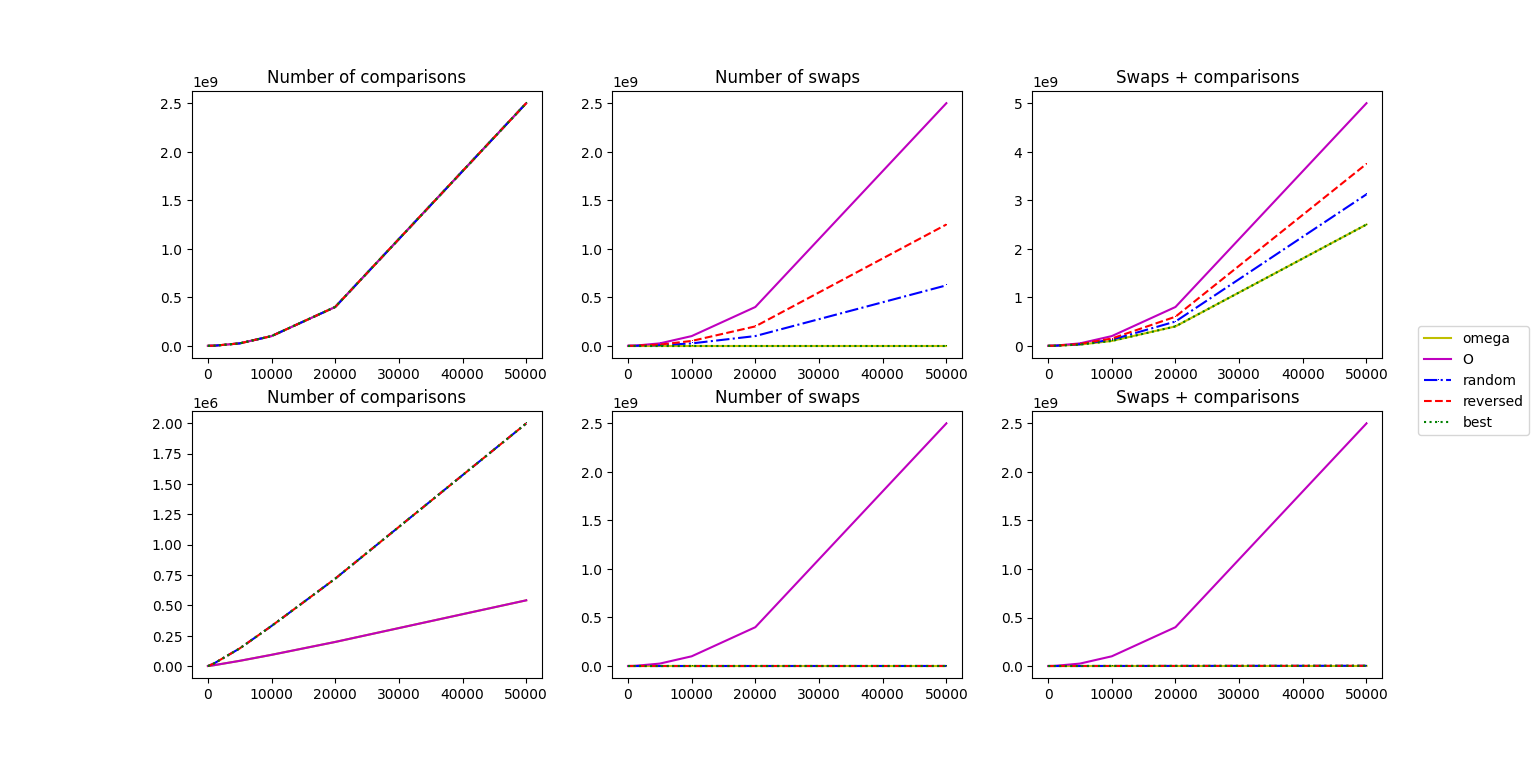


Рисунок 3.3 – Графіки залежності часових характеристик оцінювання

Висновок

Під час виконання даної лабораторної роботи було виконано аналіз алгоритмів внутрішнього сортування «Бульбашкою» та «Добошевича» на стійкість, природність поведінки, базування на порівняннях, необхідність у додатковій пам’яті та необхідність у знаннях про структури даних, записано ці алгоритми за допомогою псевдокоду, проведено аналіз часової складності у найгіршому, кращому і середньому випадках та записано її у асимптотичних оцінках, було виконано програмну реалізацію алгоритмів на мові програмування Python, проведено ряд випробувань алгоритмів на масивах різної розмірності і побудовано графіки залежності часових характеристик оцінювання від розмірності масиву.

Був зроблений висновок, що алгоритм сортуваня Добошевича є більш ефективним методом сортування, ніж неудосконалене сортування Бульбашкою за рахунок оптимізації загальної кількості порівнянь і перестановок.

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%.