**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**”

**Перевірив**

(прізвище, ім'я, по батькові)

*Халус Олена Андріївна*

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-11 Лошак Віктор Іванович*

Київ 2022

**Лабораторна робота 1**

**Мета роботи:** вивчити основні методи аналізу обчислювальної складності алгоритмів внутрішнього сортування і оцінити поріг їх ефективності.

**Постановка задачі:** 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 алгоритмів на відповідність властивостям

Аналіз алгоритмів **сортування бульбашкою** та **сортування гребінцем** на відповідність властивостям наведено в таблиці 3.1.

Таблиця 3.1 – Аналіз алгоритмів на відповідність властивостям

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

## Псевдокод алгоритму

**Алгоритм вдосконаленої бульбашки:**

**BubbleSort(array, size)**

compCounter=0

swapCounter=0

for i in range (size-1):

for j in range (size-i-1):

compCounter+=1

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

swap(arr[j+1], arr[j])

swap\_counter+=1

return compare\_counter, swap\_counter

**combSort(arr, length)**

gap = length

swapCounter = 0

compCounter = 0

swap = true

while swap or gap > 1

do:

swap = false

if gap > 1 do:

gap /= 1.247

else gap = 1;

for i = gap, length, 1

if arr[i] < arr[i - gap] do:

swap(arr[i], arr[i - gap])

swapCounter+=1

swap = true

compCounter+=1

## Аналіз часової складності

|  |  |  |
| --- | --- | --- |
|  | **BubbleSort** | **CombSort** |
| Найгірша швидкодія | О(n²) | О(n²) |
| Найкраща швидкодія | О(n) | О(nlogn) |
| Середня швидкодія | О(n²) | O(n^2/2^p) |

## Програмна реалізація алгоритму

### Вихідний код

void combSort(int\* arr, int length, int& swapCounter, int& compCounter)

{

int gap = length;

swapCounter = 0;

compCounter = 0;

bool swap = true;

while (swap || gap > 1)

{

swap = false;

if (gap > 1) gap = floor(double(gap) / 1.247);

else gap = 1;

for (int i = gap; i < length; i++)

{

if (arr[i] < arr[i - gap])

{

std::swap(arr[i], arr[i - gap]);

swapCounter++;

swap = true;

}

compCounter++;

}

}

}

void bubbleSort(int\* arr, int length, int& swapCounter, int& compCounter)

{

swapCounter = 0;

compCounter = 0;

bool swap = true;

for (int i = 0; i < length - 1 && swap == true; i++)

{

swap = false;

for (int j = 0; j < length - 1 - i; j++)

{

if (arr[j + 1] < arr[j])

{

std::swap(arr[j + 1], arr[j]);

swapCounter++;

swap = true;

}

compCounter++;

}

}

}

### Приклад роботи

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

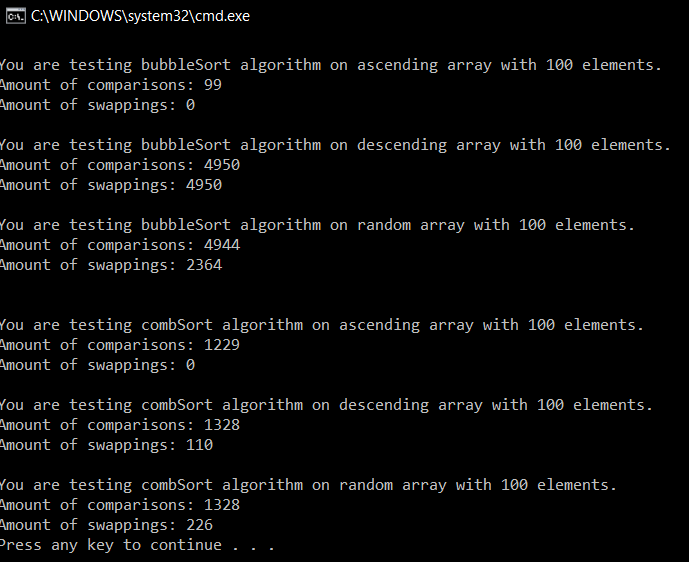


Рисунок 3.1 – Сортування масиву на 100 елементів

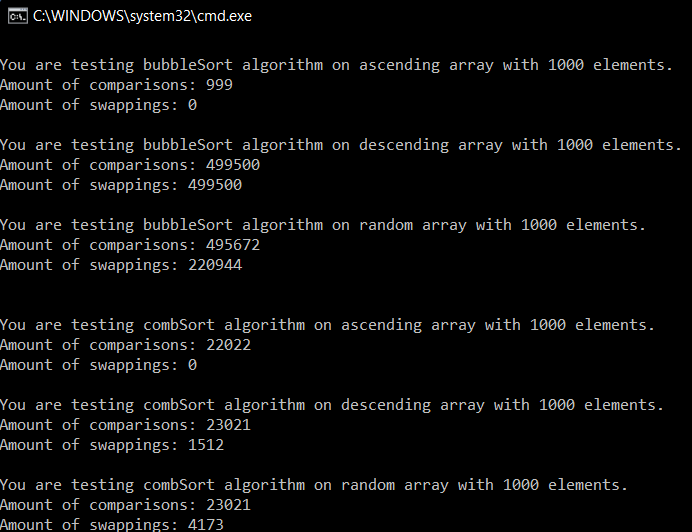


Рисунок 3.2 – Сортування масиву на 1000 елементів

## Тестування алгоритму

### Часові характеристики оцінювання

В таблиці 3.2 наведені характеристики оцінювання числа порівнянь і числа перестановок 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 | 9 | 0 |
| 100 | 99 | 0 |
| 1000 | 999 | 0 |
| 5000 | 4999 | 0 |
| 10000 | 9999 | 0 |
| 20000 | 19999 | 0 |
| 50000 | 49999 | 0 |

**CombSort**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 36 | 0 |
| 100 | 1229 | 0 |
| 1000 | 22022 | 0 |
| 5000 | 144832 | 0 |
| 10000 | 329598 | 0 |
| 20000 | 719136 | 0 |
| 50000 | 1997680 | 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 |

**CombSort**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 9 |
| 100 | 1328 | 110 |
| 1000 | 23021 | 1512 |
| 5000 | 149831 | 9154 |
| 10000 | 339597 | 19018 |
| 20000 | 739135 | 40730 |
| 50000 | 2047679 | 110332 |

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 | 39 | 20 |
| 100 | 4814 | 2087 |
| 1000 | 497670 | 232024 |
| 5000 | 12475764 | 5730890 |
| 10000 | 49979949 | 22599956 |
| 20000 | 199958625 | 86515730 |
| 50000 | 1096140430 | 260676660 |

**CombSort**

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 9 |
| 100 | 1328 | 231 |
| 1000 | 24020 | 4254 |
| 5000 | 154830 | 26910 |
| 10000 | 349596 | 58883 |
| 20000 | 759134 | 127915 |
| 50000 | 2097678 | 228373 |

### Графіки залежності характеристики кількості перестановок оцінювання від розмірності масиву

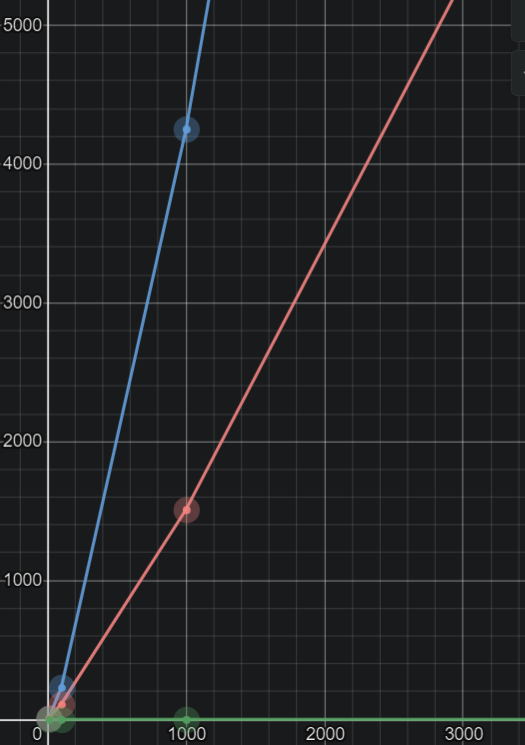
На рисунку 3.3 показані графіки залежності характеристики кількості перестановок від розмірності масиву для випадків, коли масиви містять упорядковану послідовність елементів (зелений графік), коли масиви містять зворотно упорядковану послідовність елементів (червоний графік), коли масиви містять випадкову послідовність елементів (синій графік), також показані асимптотичні оцінки гіршого (фіолетовий графік) і кращого (жовтий графік) випадків для порівняння.

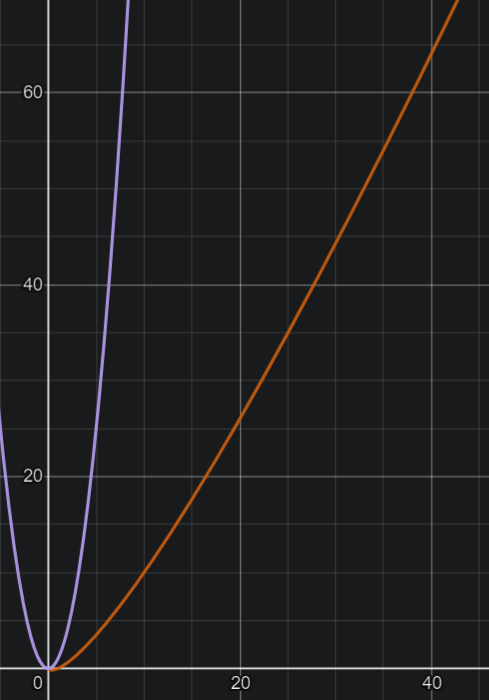
Рисунок 3.3 – Графіки залежності часових характеристик оцінювання

**BubbleSort:** x-розмір масиву, у-кількість перестановок



**CombSort:** x-розмір масиву, у-кількість перестановок





**Висновок:** При виконанні даної лабораторної роботи я вивчив основні методи аналізу обчислювальної складності алгоритмів сортування бульбашкою та гребінцем, і оцінив поріг їхньої ефективності. Я з’ясував, що алгоритм сортування гребінцем значно ефективніший за алгоритм сортування бульбашкою при сортуванні масивів великої розмірності.