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) *Замковий Дмитро Володимирович* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов Олексій Олександрович* _____
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

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	Сортування гребінцем («розчіскою»)

3 ВИКОНАННЯ

3.1 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	Так
«Природність» поведінки (Adaptability)	Ні
Базуються на порівняннях	Так
Необхідність в додатковій пам'яті (об'єм)	Ні
Необхідність в знаннях про структури даних	Так

Аналіз алгоритму сортування гребінцем на відповідність властивостям наведено в таблиці 3.2.

Таблиця 3.2 – Аналіз алгоритму на відповідність властивостям

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

3.2 Псевдокод алгоритму

```
функція bubble_sort(arr)
    comp = 0
    swaps = 0
    повторити для i від 0 до (len(arr) - 1) з кроком 1
        повторити для j від 0 до (len(arr) - 1) з кроком 1
            comp += 1
            якщо arr[j] > arr[j + 1], то
                swaps += 1
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
            все якщо
        все повторити
    все повторити
    повернути arr, comp, swaps
все функція

функція new_gap(gap, if_sorted)
    k = 1,247
    gap = int(gap // k)
    якщо gap <= 1, то
        gap = 1
        if_sorted = True
    все якщо
    повернути gap, if_sorted
все функція
```

```
функція comb_sort(arr)
    comp = 0
    swaps = 0
    gap = len(arr)
    if_sorted = False
    повторити поки не if_sorted
        gap, if_sorted = new_gap(gap, if_sorted)
        повторити для i від 0 до (len(arr) - gap) з кроком 1
            comp += 1
            якщо arr[i] > arr[i + gap], то
                swaps += 1
                arr[i], arr[i + gap] = arr[i + gap], arr[i]
                if_sorted = False
            все якщо
        все повторити
    все повторити
    повернути arr, comp, swaps
все функція
```

3.3 Аналіз часової складності

	Сортування бульбашкою	Сортування гребінцем
Найгірший випадок	$O(n^2)$	$O(n^2)$
Найкращий випадок	$O(n^2)$	$O(n^2)$
Середній випадок	$\Omega(n^2)$	$\Omega(n*\log(n))$

3.4 Програмна реалізація алгоритму

3.4.1 Вихідний код на мові програмування Python

```python
# Bubble sort
def bubble_sort(arr):
    comp = 0
    swaps = 0
    for i in range(len(arr) - 1):
        for j in range(len(arr) - 1):
            comp += 1
            if arr[j] > arr[j + 1]:
                swaps += 1
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr, comp, swaps


def new_gap(gap, if_sorted):
    k = 1.247  # ~1/(1 - 1/(exp^fi))
    gap = int(gap // k)
    if gap <= 1:
        gap = 1
        if_sorted = True
    return gap, if_sorted


# Comb sort
def comb_sort(arr):
    comp = 0
    swaps = 0
    gap = len(arr)
    if_sorted = False
    while not if_sorted:
        gap, if_sorted = new_gap(gap, if_sorted)
        for i in range(len(arr) - gap):
            comp += 1
            if arr[i] > arr[i + gap]:
                swaps += 1
                arr[i], arr[i + gap] = arr[i + gap], arr[i]
                if_sorted = False
    return arr, comp, swaps
```

3.4.2 Приклад роботи

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

Рисунок 3.1.1 – Сортування масиву на 100 елементів (Bubble Sort)

```
C:\Users\Dima\source\PC\ASD\venv\Scripts\python.exe C:\Users\Dima\source\PC\ASD\Lab1\main1.py
Array size
================================================
Use positive number
================================================
Enter size: 100

Fill array
================================================
'r' - to fill array with random numbers
's' - to fill array with sorted numbers
'i' - to fill array with inverse numbers
================================================
Enter method of filling array: r

Sorting
================================================
'b' - sort array by bubble sort
'c' - sort array by comb sort
================================================
Enter a sort method: b

Output array
================================================
Use positive numbers to output n numbers of array
Use 0 to no output array
================================================
Enter how many numbers to output from the array: 100

Not sorted array:
37, 87, 5, 57, 51, 34, 45, 40, 23, 62, 81, 43, 15, 11, 56, 29, 4, 2, 14, 44,
10, 39, 77, 50, 20, 28, 3, 83, 41, 8, 25, 13, 82, 97, 24, 55, 80, 84, 60, 92,
66, 69, 16, 99, 74, 12, 78, 67, 95, 70, 64, 59, 18, 73, 58, 90, 30, 31, 63, 35,
1, 94, 27, 17, 61, 86, 33, 9, 65, 79, 89, 26, 72, 49, 7, 46, 76, 68, 71, 32,
52, 91, 19, 85, 100, 53, 88, 96, 48, 98, 54, 38, 22, 47, 93, 21, 42, 75, 36, 6

Sorted array:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100

Number of comparisons: 9801
Number of swaps: 2157

Process finished with exit code 0
```

Рисунок 3.1.2 – Сортування масиву на 100 елементів (Comb Sort)

```
C:\Users\Dima\source\PC\ASD\venv\Scripts\python.exe C:\Users\Dima\source\PC\ASD\Lab1\main1.py
Array size
================================================
Use positive number
================================================
Enter size: 100

Fill array
================================================
'r' - to fill array with random numbers
's' - to fill array with sorted numbers
'i' - to fill array with inverse numbers
================================================
Enter method of filling array: r

Sorting
================================================
'b' - sort array by bubble sort
'c' - sort array by comb sort
================================================
Enter a sort method: c

Output array
================================================
Use positive numbers to output n numbers of array
Use 0 to no output array
================================================
Enter how many numbers to output from the array: 100

Not sorted array:
94, 47, 92, 90, 84, 55, 72, 89, 65, 73, 98, 30, 19, 59, 8, 91, 58, 86, 85, 10,
78, 48, 52, 15, 35, 2, 76, 22, 51, 11, 45, 71, 32, 88, 60, 67, 42, 37, 82, 43,
26, 39, 33, 18, 54, 21, 100, 87, 75, 97, 16, 70, 27, 81, 56, 99, 93, 25, 63, 49,
17, 6, 83, 36, 3, 69, 44, 12, 7, 41, 24, 50, 4, 80, 53, 28, 77, 9, 64, 5,
1, 62, 68, 46, 96, 31, 14, 57, 23, 20, 74, 61, 66, 40, 95, 34, 38, 13, 29, 79

Sorted array:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100

Number of comparisons: 1328
Number of swaps: 249

Process finished with exit code 0
```

Рисунок 3.2.1 – Сортування масиву на 1000 елементів (Bubble Sort)

```
C:\Users\Dima\source\PC\ASD\venv\Scripts\python.exe C:\Users\Dima\source\PC\ASD\Lab1\main1.py
Array size
================================================
Use positive number
================================================
Enter size: 1000

Fill array
================================================
'r' - to fill array with random numbers
's' - to fill array with sorted numbers
'i' - to fill array with inverse numbers
================================================
Enter method of filling array: r

Sorting
================================================
'b' - sort array by bubble sort
'c' - sort array by comb sort
================================================
Enter a sort method: b

Output array
================================================
Use positive numbers to output n numbers of array
Use 0 to no output array
================================================
Enter how many numbers to output from the array: 100

Not sorted array:
156, 914, 627, 802, 223, 744, 716, 267, 844, 354, 659, 108, 766, 594, 292, 55, 605, 103, 805, 300,
882, 404, 448, 667, 274, 445, 336, 687, 646, 439, 684, 650, 878, 433, 673, 542, 41, 848, 343, 961,
782, 860, 945, 20, 179, 828, 186, 323, 929, 437, 281, 636, 888, 839, 867, 697, 676, 616, 97, 31,
968, 996, 711, 416, 426, 296, 68, 686, 162, 401, 768, 937, 957, 562, 362, 745, 263, 102, 813, 182,
347, 5, 133, 105, 81, 969, 607, 559, 318, 143, 450, 742, 53, 303, 434, 14, 397, 826, 985, 261

Sorted array:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100

Number of comparisons: 998001
Number of swaps: 257772

Process finished with exit code 0
```

Рисунок 3.2.2 – Сортування масиву на 1000 елементів (Comb Sort)

```
C:\Users\Dima\source\PC\ASD\venv\Scripts\python.exe C:\Users\Dima\source\PC\ASD\Lab1\main1.py
Array size
================================================
Use positive number
================================================
Enter size: 1000

Fill array
================================================
'r' - to fill array with random numbers
's' - to fill array with sorted numbers
'i' - to fill array with inverse numbers
================================================
Enter method of filling array: r

Sorting
================================================
'b' - sort array by bubble sort
'c' - sort array by comb sort
================================================
Enter a sort method: c

Output array
================================================
Use positive numbers to output n numbers of array
Use 0 to no output array
================================================
Enter how many numbers to output from the array: 100

Not sorted array:
286, 275, 905, 12, 129, 534, 651, 378, 335, 103, 45, 344, 451, 359, 15, 82, 218, 258, 676, 72,
890, 795, 375, 63, 904, 159, 436, 907, 817, 366, 168, 68, 106, 289, 223, 394, 942, 715, 855, 282,
459, 131, 3, 745, 913, 234, 56, 231, 351, 349, 705, 768, 655, 427, 414, 468, 959, 1000, 363, 683,
583, 401, 872, 893, 540, 141, 891, 91, 456, 149, 315, 226, 450, 180, 187, 95, 755, 608, 480, 992,
757, 447, 70, 229, 207, 407, 968, 198, 113, 318, 750, 677, 140, 632, 771, 71, 73, 948, 532, 514

Sorted array:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100

Number of comparisons: 23021
Number of swaps: 4240

Process finished with exit code 0
```

3.5 Тестування алгоритму

3.5.1 Часові характеристики оцінювання

В таблиці 3.2.1 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки 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	99801	0
5000	24990001	0
10000	99980001	0
20000	399960001	0
50000	2499900001	0

В таблиці 3.2.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування гребінцем для масивів різної розмірності, коли масив містить упорядковану послідовність елементів.

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

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 3.3.1 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашкою для масивів різної розмірності, коли масиви містять зворотно упорядковану послідовність елементів.

Таблиця 3.3.1 – Характеристики оцінювання алгоритму сортування бульбашкою для зворотно упорядкованої послідовності елементів у масиві

Розмірність масиву	Число порівнянь	Число перестановок
10	81	45
100	9801	4590
1000	99801	49950
5000	24990001	12497500
10000	99980001	49995000
20000	399960001	199990000
50000	2499900001	1249975000

В таблиці 3.3.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування гребінцем для масивів різної розмірності, коли масиви містять зворотно упорядковану послідовність елементів.

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

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 3.4.1 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки 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	15
100	9801	2399
1000	99801	239258
5000	24990001	6293731
10000	99980001	24789932
20000	399960001	100281022
50000	2499900001	726493845

У таблиці 3.4.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування гребінцем для масивів різної розмірності, масиви містять випадкову послідовність елементів.

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

Розмірність масиву	Число порівнянь	Число перестановок
10	45	11
100	1328	260
1000	23021	4330
5000	149831	27354
10000	339597	60201
20000	739135	130823
50000	2047679	368327

3.5.2 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 3.3.1 – Графіки залежності часових характеристик оцінювання сортування бульбашкою

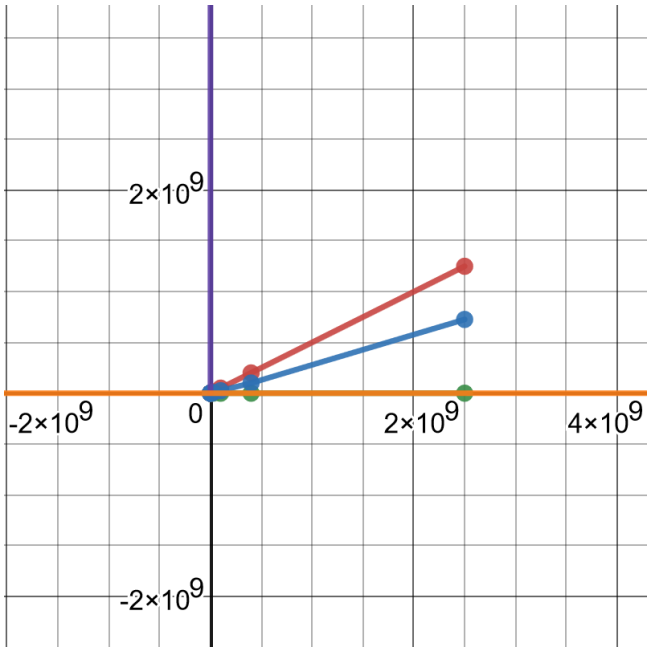
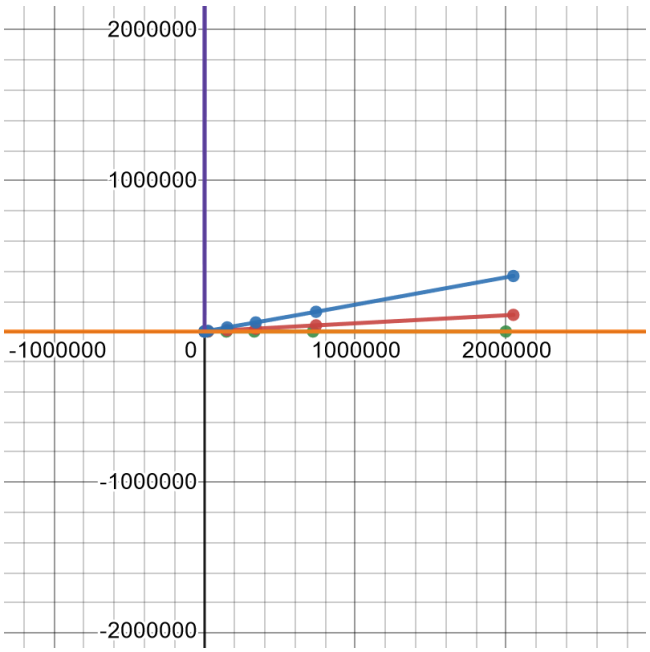


Рисунок 3.3.1 – Графіки залежності часових характеристик оцінювання сортування бульбашкою



Висновок

При виконанні даної лабораторної роботи я дослідив алгоритм сортування бульбашкою та гребінцем (bubble sort та comb sort відповідно), провів аналіз алгоритмів на відповідність властивостям, та проаналізував часову складність алгоритмів та порівняв часові характеристики.

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%.