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) *Лисенко Анастасія Олегівна ІП-13* _____
(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

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

```
процедура bubble_sort (array):
    len_array ← length[array]
    повторити для i ← 0 до (len_array – 1):
        повторити для j ←  0 до (len_array – i – 1):
            якщо( array[j] > array[j+1])
                то
                    buff  ← array[j]
                    array[j] ← array[j+1]
                    array[j+1] ← buff
            все якщо
        все повторити
    все повторити
все процедура


процедура comb_sort(array):
    len_array ←length[array]
    shrink ←1.247
    якщо len_array > 1:
       то
        gap ← (len_array/shrink)
    інакше:
        gap ←  0
    все якщо
    поки (gap):
        swapped ← False
        повторити для i ← 0 до (len_array – gap):
            якщо ( arr[i + gap] < arr[i] )
                то
                    buff  ← array[i]
                    array[i] ← array[i + gap]
                    array[i + gap] ← buff
                    swapped ← True
            все якщо
        все повторити
        gap ←  (len_array/shrink)
    все повторити
все процедура
```

3.3 Аналіз часової складності

Сортування бульбашкою: O(n) – в найкращому випадку,

$O(n^2)$ – в найгіршому та середньому випадках.

Отже, часова складність сортування бульбашкою: $O(n^2)$

Сортування гребінцем: O(n*log(n)) – в найкращому випадку,

$O(n^2)$ – в найгіршому випадку та середньому випадках.

Отже, часова складність сортування гребінцем: $O(n^2)$

Отже, часова складність усього алгоритму: $O(n^2)$

3.4 Програмна реалізація алгоритму

3.4.1 Вихідний код

Main.py:
```
from arrays import *

size = size_of_array()
select_sorting_method(size)
```

sorting.py:
```
from time import perf_counter_ns

def bubble_sort(array):
    len_array = len(array)
    bubble_comp = 0
    bubble_swap = 0
    time_start = perf_counter_ns()
    for i in range(len_array-1):
        for j in range(len_array-i-1):
            bubble_comp += 1
            if array[j] > array[j+1]:
                bubble_swap += 1
                buff = array[j]
                array[j] = array[j+1]
                array[j+1] = buff
    time_spent = perf_counter_ns() - time_start
    return array, bubble_swap, bubble_comp, time_spent


def comb_sort(arr):
    len_array = len(arr)
    shrink = 1.247
    if len_array > 1:
        gap = int(len_array / shrink)
    else:
        gap = 0
    comb_swap = 0
    comb_comp = 0
    time_start = perf_counter_ns()
    while gap:
        swapped = False
```

```
        for i in range(len_array - gap):
            comb_comp += 1
            if arr[i + gap] < arr[i]:
                comb_swap += 1
                arr[i], arr[i + gap] = arr[i + gap], arr[i]
                swapped = True
        gap = int(gap / shrink)
    time_spent = perf_counter_ns() - time_start
    return arr, comb_swap, comb_comp, time_spent
```

arrays.py:

```
import random
from sorting import *
import time

def size_of_array():
    size = int(input('enter the size of your array:'))
    while (size<0 and size>50000):
        size = int(input('enter correct size:'))
    return size


def create_array(size):
    selected_mode = input("enter the letter 'r' - if you want to create array
with random elements,\nthe letter 'b' - if you want to create the best case
scenario array,\nthe letter 'w' - if you want to create the worst case scenario
array.\n")
    while (selected_mode != "r" and selected_mode != "b" and selected_mode !=
"w"):
        selected_mode = input("enter correct letter:")
    if selected_mode == "r":
        array = random.sample(range(1,size+1), size)
    if selected_mode == "b":
        array = list(range(1, size+1))
    if selected_mode == "w":
        array = list(range(size, 0, -1))

    return array

def select_sorting_method(size):
    selected_sort = input("enter the letter 'b' - if you want to use bubble
sort,\nenter the letter 'c' - if you want to use comb sort.\n")
    while (selected_sort != 'b' and selected_sort != 'c'):
        selected_sort = input("enter correct letter:")
    if selected_sort == 'b':
        array = create_array(size)
        print("Start array:", array)
        arr, bubble_swap, bubble_comp, time_spent = bubble_sort(array)
        print("The sorted array:", arr)
        print("The amount of swaps:", bubble_swap)
        print("The amount of comparisons:", bubble_comp)
        print("Spent time:", time_spent, "in nanoseconds")
    if selected_sort == 'c':
        array = create_array(size)
        print("Start array:", array)
        arr, comb_swap, comb_comp, time_spent = comb_sort(array)
        print("The sorted array:", arr)
        print("The amount of swaps:", comb_swap)
        print("The amount of comparisons:", comb_comp)
        print("Spent time:", time_spent, "in nanoseconds")
```

3.4.2 Приклад роботи

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

Масив на 100 елементів. Сортуємо бульбашкою та гребінцем. Перевіряємо 3 випадки.

Найкращий випадок(бульбашкою):

```
Start 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,
The 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,
The amount of swaps: 0
The amount of comparisons: 4950
Spent time: 775200 in nanoseconds
```

Середній випадок(бульбашкою):

```
Start array: [84, 55, 78, 57, 52, 35, 28, 44, 38, 4, 29, 23, 62, 9, 2, 74, 16, 40, 3, 63, 17, 90, 15, 76, 83, 51, 6, 43, 53, 13, 95, 85, 50, 88, 24, 94, 75, 22, 59, 49, 30, 69, 87,
The 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,
The amount of swaps: 2389
The amount of comparisons: 4950
Spent time: 1374900 in nanoseconds
```

Найгірший випадок(бульбашкою):

```
Start array: [100, 99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59
The 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,
The amount of swaps: 4950
The amount of comparisons: 4950
Spent time: 1964600 in nanoseconds
```

Найкращий випадок(гребінцем):

```
Start 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,
The 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,
The amount of swaps: 0
The amount of comparisons: 1229
Spent time: 164800 in nanoseconds
```

Середній випадок(гребінцем):

```
Start array: [66, 48, 20, 99, 14, 38, 75, 36, 10, 72, 60, 39, 34, 40, 63, 61, 77, 37, 94, 51, 49, 28, 52, 80, 55, 84, 100, 41, 65, 92, 59, 1, 30, 19, 33, 79, 93, 83, 17, 64, 13, 43,
The 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,
The amount of swaps: 269
The amount of comparisons: 1229
Spent time: 260000 in nanoseconds
```

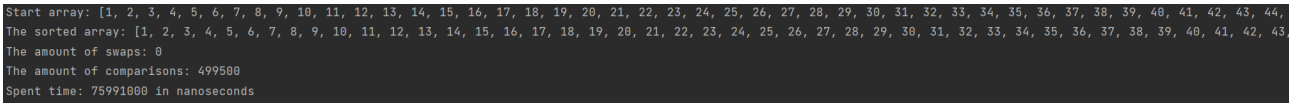
Найгірший випадок(гребінцем):

```
Start array: [100, 99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59
The 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,
The amount of swaps: 110
The amount of comparisons: 1229
Spent time: 342100 in nanoseconds
```

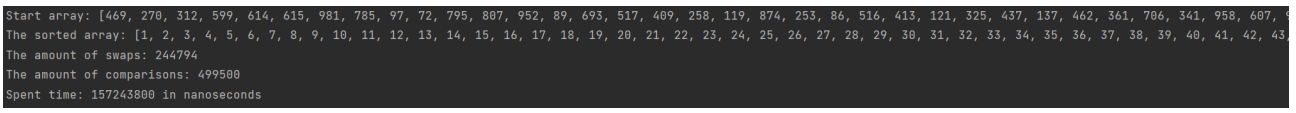
Рисунок 3.1 – Сортування масиву на 100 елементів

Масив на 100 елементів. Сортуємо бульбашкою та гребінцем. Перевіряємо 3 випадки.

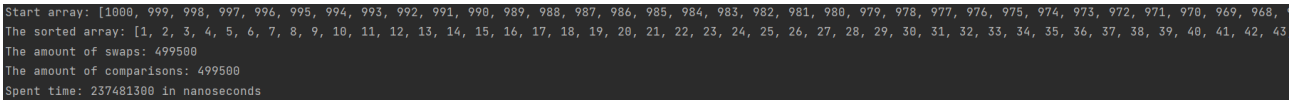
Найкращий випадок(бульбашкою):



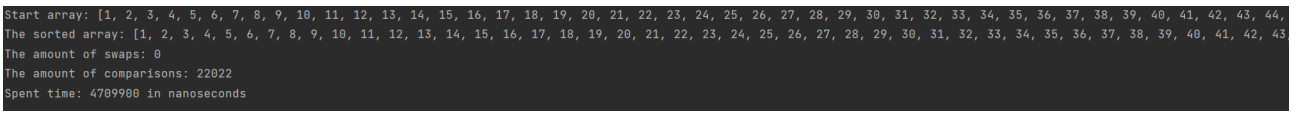
Середній випадок(бульбашкою):



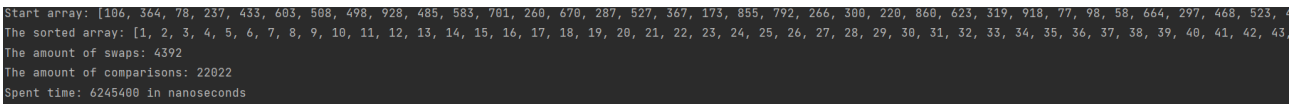
Найгірший випадок(бульбашкою):



Найкращий випадок(гребінцем):



Середній випадок(гребінцем):



Найгірший випадок(гребінцем):

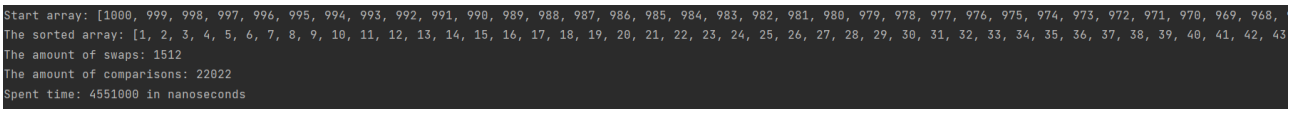


Рисунок 3.2 – Сортування масиву на 1000 елементів

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	45	0
100	4950	0
1000	499500	0
5000	12497500	0
10000	49995000	0
20000	199990000	0
50000	1249975000	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 бульбашки 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	45	45
100	4950	4950
1000	499500	499500
5000	12497500	12497500
10000	49995000	49995000
20000	199990000	199990000
50000	1249975000	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 бульбашкидля 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	45	10
100	4950	2560
1000	499500	244379
5000	12497500	6277358
10000	49995000	24860313
20000	199990000	99896796
50000	1249975000	628051285

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.

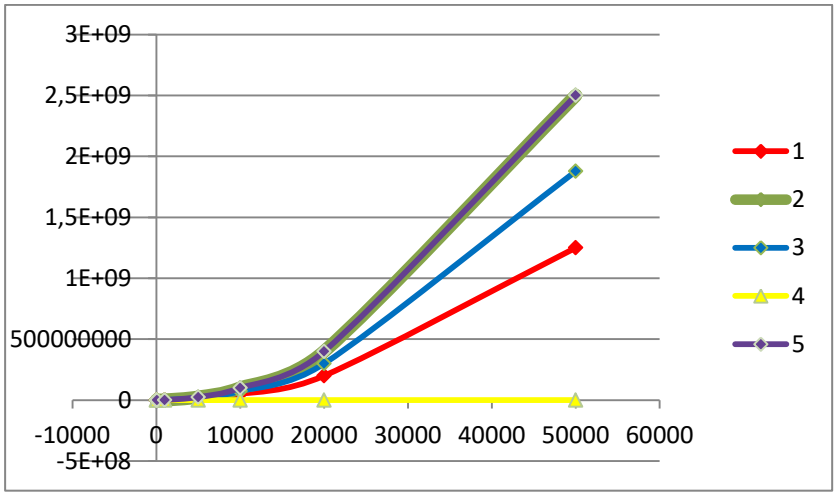
Таблиця 3.4.2 – Характеристика оцінювання алгоритму сортування гребінцем для випадкової послідовності елементів у масиві.

Розмірність масиву	Число порівнянь	Число перестановок
10	45	9
100	1328	233
1000	24020	4147
5000	154830	27367
10000	349596	60059
20000	759134	130656
50000	2097678	366709

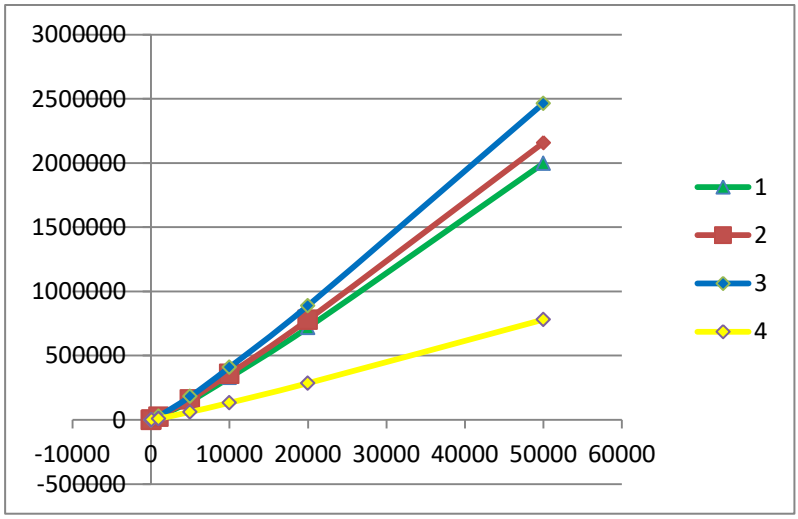
3.5.2 Графіки залежності часових характеристик оцінювання від розмірності масиву

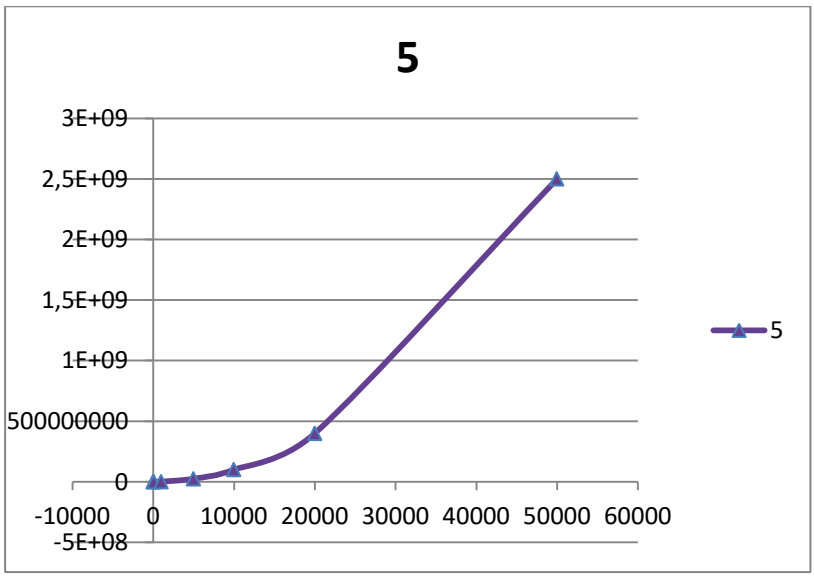
На рисунку 3.3 показані графіки залежності суми кількості порівнянь і перестановок від розмірності масиву для випадків, коли масиви містять упорядковану послідовність елементів (зелений графік), коли масиви містять зворотно упорядковану послідовність елементів (червоний графік), коли масиви містять випадкову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

Сортування бульбашкою:



Сортування гребінцем:





- найгірший випадок

Рисунок 3.3 – Графіки залежності часових характеристик оцінювання

ВИСНОВОК

При виконанні даної лабораторної роботи ми вивчили основні методи аналізу обчислювальної складності алгоритмів внутрішнього сортування і оцінили поріг їх ефективності, проаналізували, реалізували, протестували та порівняли алгоритми сортування бульбашкою та гребінцем, виявили, що хоч ці алгоритми і мають однакову асимптотичну ефективність O(n^2), сортування гребінцем є ефективнішим в середньому та гіршому випадках.

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%.