Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний

інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 1 з дисципліни

«Алгоритми та структури даних 2. Структури даних»

«**Проектування і аналіз алгоритмів внутрішнього сортування**»

Виконав студент: ІП-15 Поліщук Валерій Олександрович

(шифр, прізвище, ім'я, по батькові)

Перевірив: Соколовський Владислав Володимирович

(прізвище, ім'я, по батькові)

Київ 2022

Лабораторна робота № 1

Проектування і аналіз алгоритмів внутрішнього сортування

Мета — 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

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	Алгоритм є стійким
«Природність» поведінки (Adaptability)	Алгоритм є природним
Базуються на порівняннях	Алгоритм базується на порівняннях
Необхідність в додатковій пам'яті (об'єм)	Алгоритм не потребує додаткової пам’яті
Необхідність в знаннях про структури даних	Необхідні базові знання про структури даних

3.2 Псевдокод алгоритму

Підпрограма: BubleSort(arr)

Початок

sorted = false

i = 0

Повторити

Поки !sorted

sorted = true

Повторити для j від 0 до до len(arr) – i - 2

Якщо arr[j] > arr[j+1]

То

temp = arr[j]

arr[j] = arr[j+1]

arr[j+1] = temp

sorted = false

Все якщо

Все повторити

i = i + 1

Все повторити

Кінець

1.3 Аналіз часової складності

Найкращий випадок: O(n)

Найгірший випадок: O(n^2)

Середній випадок: O(n^2)

3.4 Програмна реалізація алгоритму

3.4.1 Вихідний код

```
5       private static void BubleSort(int[] arr)
6       {
7           int temp;
8           bool sorted = false;
9           int i = 0;
10
11          while (!sorted)
12          {
13              sorted = true;
14
15              for (int j = 0; j < arr.Length - i - 1; j++)
16              {
17                  if (arr[j] > arr[j + 1])
18                  {
19                      temp = arr[j];
20                      arr[j] = arr[j + 1];
21                      arr[j + 1] = temp;
22                      sorted = false;
23                  }
24              }
25
26              i++;
27          }
28      }
```

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 – Сортування масиву на 100 елементів

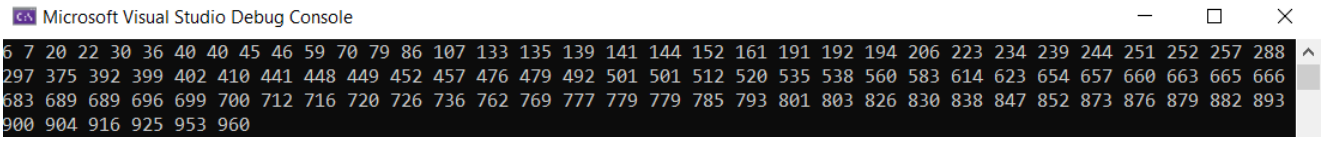
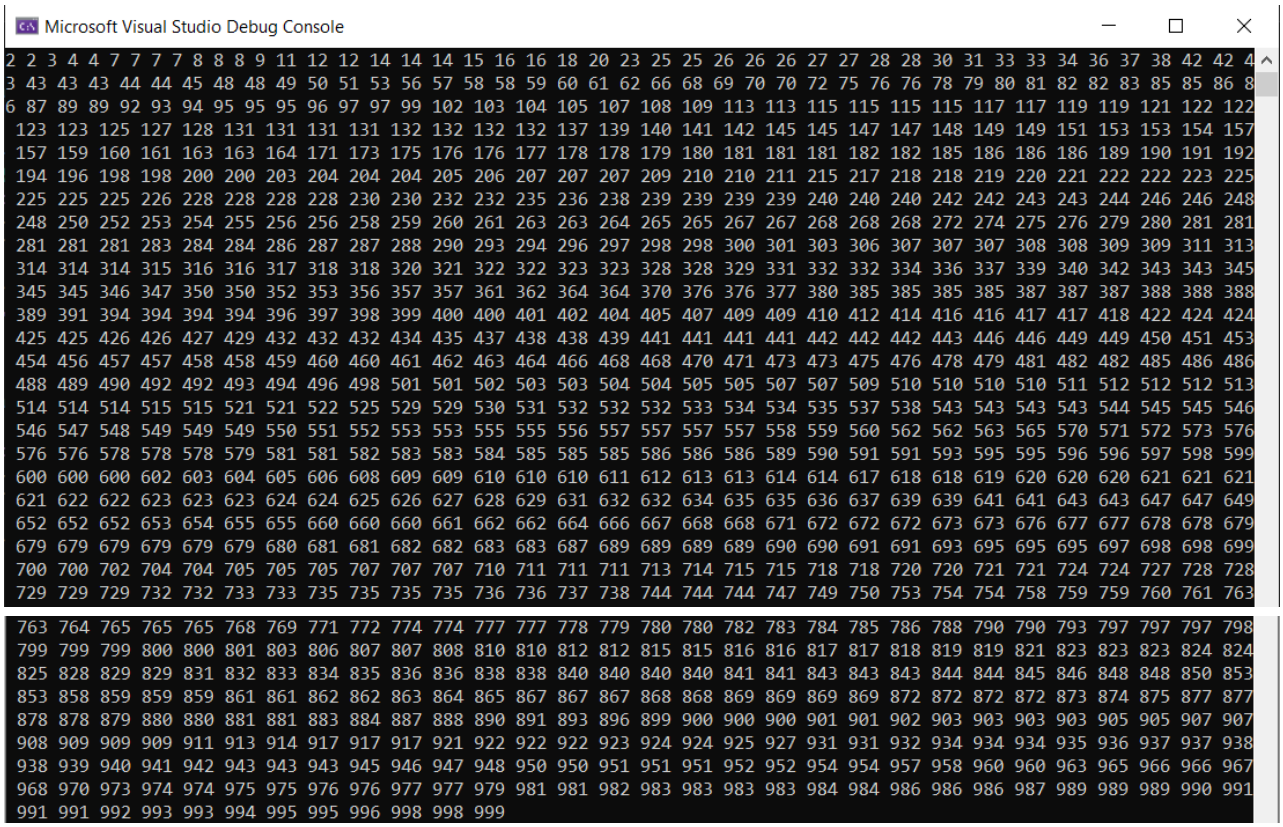


Рисунок 3.2 – Сортування масиву на 1000 елементів



3.5 Тестування алгоритму

```
16
17                  for (int j = 0; j < arr.Length - i - 1; j++)
18                  {
19                      compares++;
20
21                      if (arr[j] > arr[j + 1])
22                      {
23                          temp = arr[j];
24                          arr[j] = arr[j + 1];
25                          arr[j + 1] = temp;
26                          sorted = false;
27                          swaps++;
28                      }
29                  }
30
31                  i++;
32              }
33              Console.WriteLine();
34              Console.WriteLine("Compares : " + compares);
35              Console.WriteLine("Swaps : " + swaps);
36              Console.WriteLine();
37          }
38
39          private static int[] CreateArray(int size)...
49
50          private static void PrintArr(int[] arr)...
57
58          static void Main(string[] args)
59          {
60              int[] arr = CreateArray(10);
61              Console.WriteLine("Generated array : ");
62              PrintArr(arr);
63              Console.WriteLine();
64              BubleSort(arr);
65              Console.WriteLine("Sorted array : ");
66              PrintArr(arr);
67              Console.WriteLine();
68          }
```

```
Generated array :
164 402 399 124 295 735 442 624 558 736

Compares : 30
Swaps : 10

Sorted array :
124 164 295 399 402 442 558 624 735 736
```

```
16
17                  for (int j = 0; j < arr.Length - i - 1; j++)
18                  {
19                      compares++;
20
21                      if (arr[j] > arr[j + 1])
22                      {
23                          temp = arr[j];
24                          arr[j] = arr[j + 1];
25                          arr[j + 1] = temp;
26                          sorted = false;
27                          swaps++;
28                      }
29                  }
30
31                  i++;
32              }
33              Console.WriteLine();
34              Console.WriteLine("Compares : " + compares);
35              Console.WriteLine("Swaps : " + swaps);
36              Console.WriteLine();
37          }
38
39          private static int[] CreateArray(int size)...
49
50          private static void PrintArr(int[] arr)...
57
58          static void Main(string[] args)
59          {
60              int[] arr = CreateArray(20);
61              Console.WriteLine("Generated array : ");
62              PrintArr(arr);
63              Console.WriteLine();
64              BubleSort(arr);
65              Console.WriteLine("Sorted array : ");
66              PrintArr(arr);
67              Console.WriteLine();
68          }
```

```
Generated array :
773 73 855 309 259 696 558 642 403 216 863 599 485 924 94 735 85 269 968 845

Compares : 184
Swaps : 87

Sorted array :
73 85 94 216 259 269 309 403 485 558 599 642 696 735 773 845 855 863 924 968
```

3.5.1 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

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	499 500	499 500
5000	12 497 500	12 497 500
10000	49 995 000	49 995 000
20000	199 990 000	199 990 000
50000	1 249 975 000	1 249 975 000

У таблиці 3.4 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки для масивів різної розмірності, масиви містять випадкову послідовність елементів.

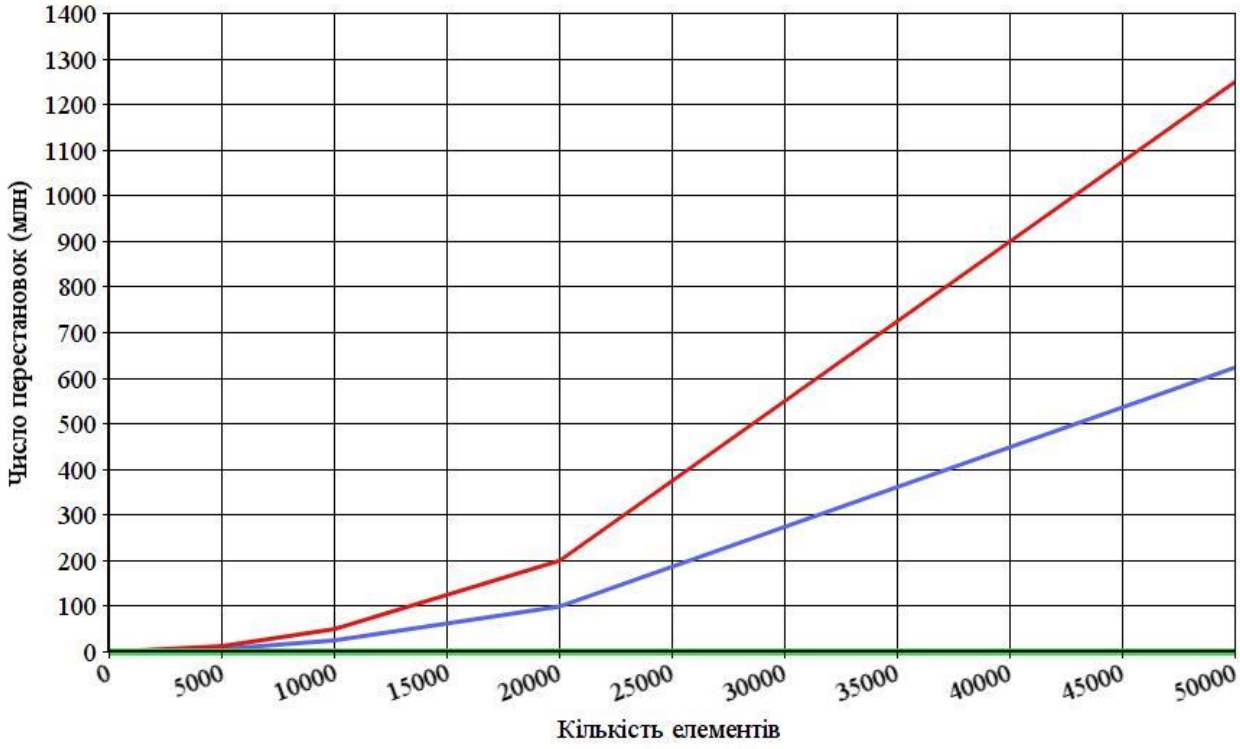
Таблиця 3.4 – Характеристика оцінювання алгоритму сортування бульбашки для випадкової послідовності елементів у масиві.

Розмірність масиву	Число порівнянь	Число перестановок
10	45	28
100	4929	2577
1000	499 269	252 959
5000	12 494 650	6 328 388
10000	49 981 305	25 244 720
20000	199 921 365	99 498 279
50000	1 249 872 622	623 586 651

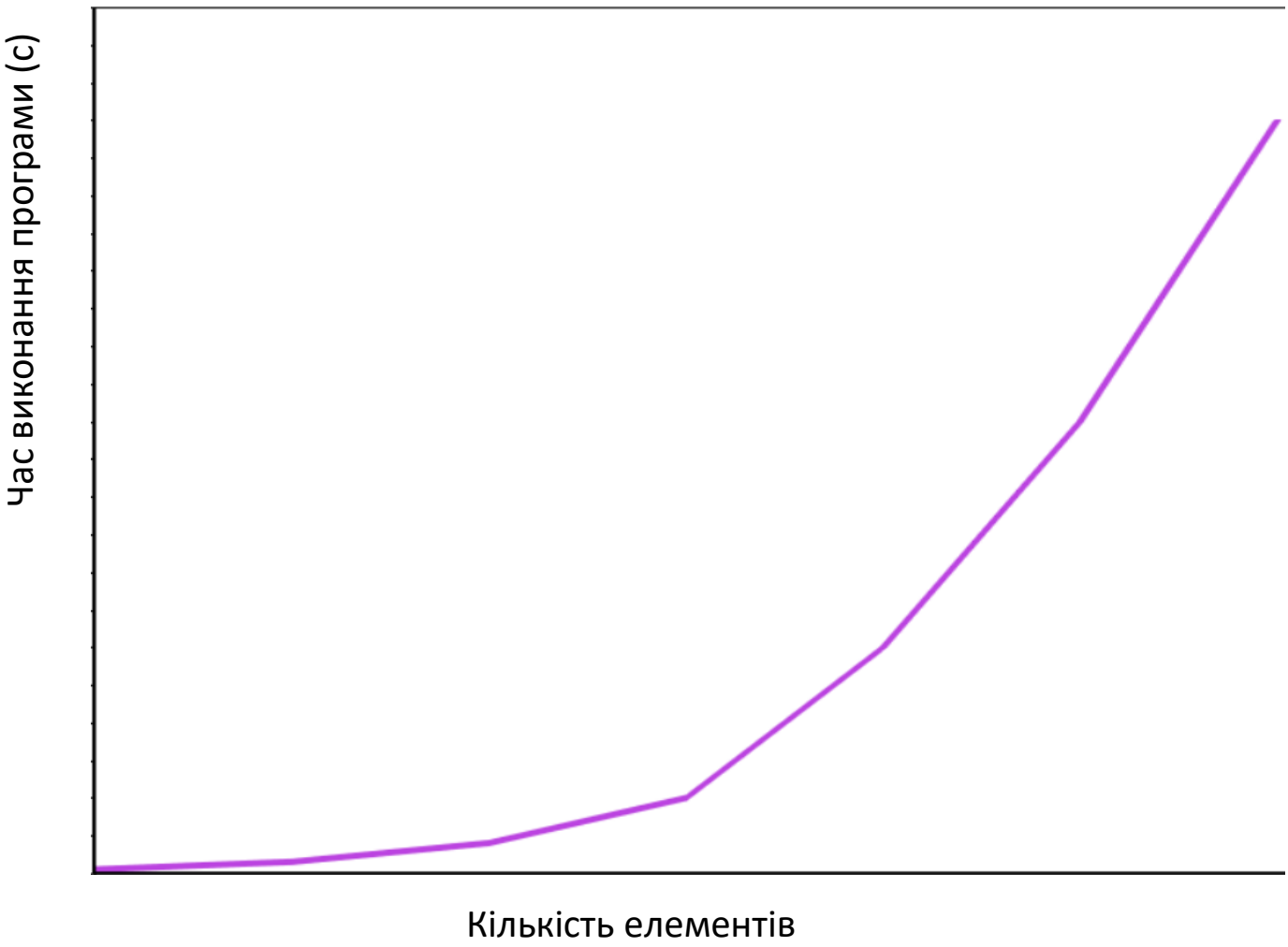
3.5.2 Графіки залежності часових характеристик оцінювання від розмірності масиву

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 – Графіки залежності часових характеристик оцінювання



Асимптотична оцінка гіршого випадку



Алгоритм сортування гребінцем

3.6 Аналіз алгоритму на відповідність властивостям

Аналіз алгоритму сортування гребінцем на відповідність властивостям наведено в таблиці 3.1.

Таблиця 3.1 – Аналіз алгоритму на відповідність властивостям

Властивість	**Сортування гребінцем**
Стійкість	Алгоритм не є стійким
«Природність» поведінки (Adaptability)	Алгоритм є природним
Базуються на порівняннях	Алгоритм базується на порівняннях
Необхідність в додатковій пам'яті (об'єм)	Алгоритм не потребує додаткової пам’яті
Необхідність в знаннях про структури даних	Необхідні базові знання про структури даних

3.7 Псевдокод алгоритму

Підпрограма: CombSort(arr)

Початок

step = len(arr)
const = 1.24733095
Повторити
Поки step > 1
step = step / const
Повторити для j від 0 до до len(arr) – step
Якщо arr[j] > arr[j+step]
То
temp = arr[j]
arr[j] = arr[j+step]
arr[j+step] = temp
Все якщо
Все повторити
Все повторити

Кінець

3.8 Аналіз часової складності

Найкращий випадок: O(n*log(n))

Найгірший випадок: $O(n^2)$

Середній випадок: $\Omega(n^2 / 2^p)$

3.9 Програмна реалізація алгоритму

3.9.1 Вихідний код

```
50          private static void CombSort(int[] arr)
51          {
52              const float constant = 1.24733095f;
53              int step = arr.Length;
54              int temp;
55
56              int compares = 0;
57              int swaps = 0;
58
59              while (step > 1)
60              {
61                  step = (int)(step / constant);
62                  for (int j = 0; j < arr.Length-step; j++)
63                  {
64                      compares++;
65
66                      if (arr[j]> arr[j+step])
67                      {
68                          temp =arr[j];
69                          arr[j] = arr[j+step];
70                          arr[j+step] = temp;
71                          swaps++;
72                      }
73                  }
74              }
75
76              Console.WriteLine();
77              Console.WriteLine("Compares : " + compares);
78              Console.WriteLine("Swaps : " + swaps);
79              Console.WriteLine();
80
81          }
82
83          private static void PrintArr(int[] arr)...
90
91          static void Main(string[] args)
92          {
93              int[] arr = CreateArray(50000);
94              Console.WriteLine("Generated array : ");
95              PrintArr(arr);
96              Console.WriteLine();
97              CombSort(arr);
98              Console.WriteLine("Sorted array : ");
99              PrintArr(arr);
100             Console.WriteLine();
101         }
102
103
104     }
```

3.9.2 Приклад роботи

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

Рисунок 3.1 – Сортування масиву на 100 елементів

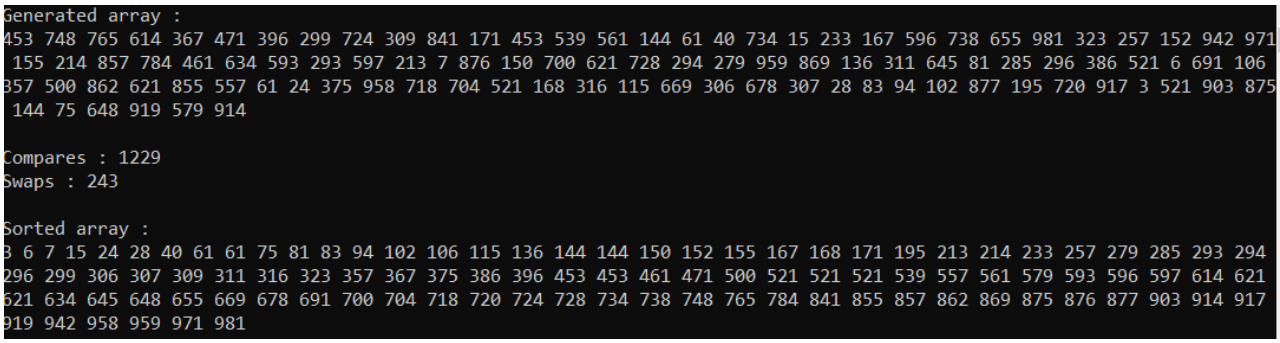
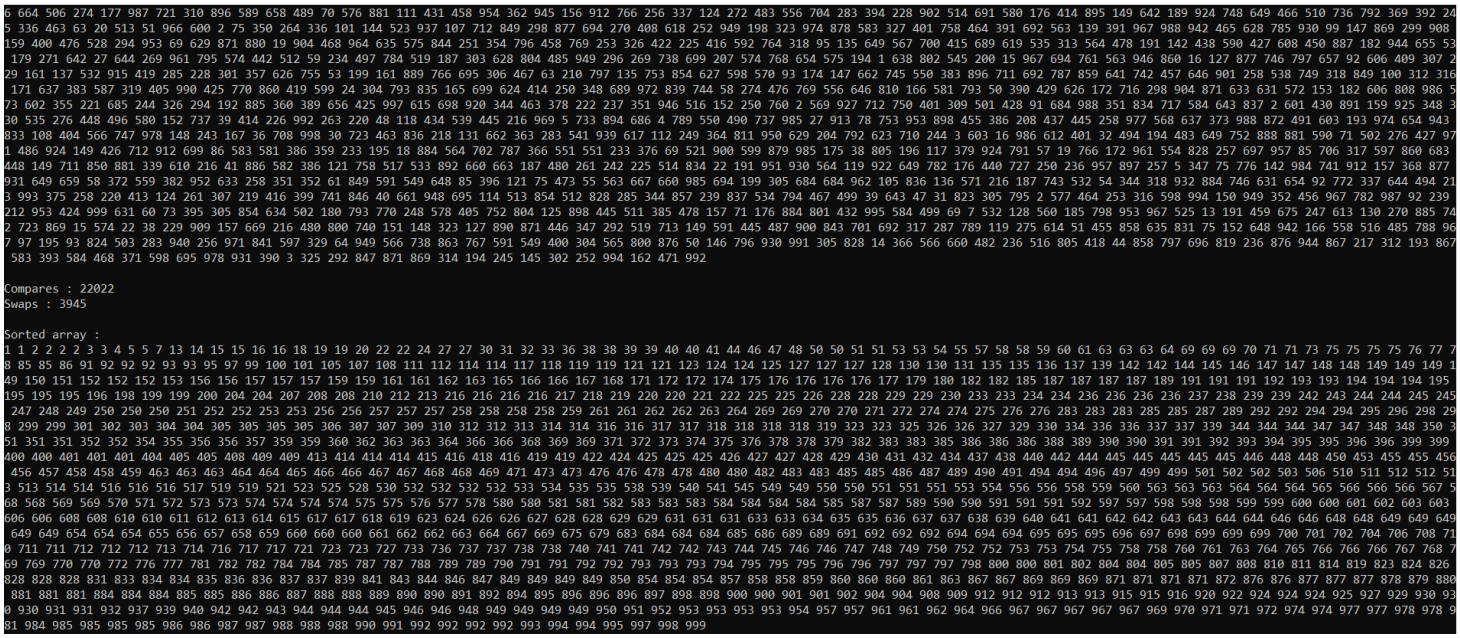


Рисунок 3.2 – Сортування масиву на 1000 елементів



3.10 Тестування алгоритму

3.10.1 Часові характеристики оцінювання

В таблиці 3.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування гребінцем для масивів різної розмірності, коли масив містить упорядковану послідовність елементів.

Таблиця 3.2 – Характеристики оцінювання алгоритму сортування гребінцем для упорядкованої послідовності елементів у масиві

Розмірність масиву	Число порівнянь	Число перестановок
10	36	0
100	1 229	0
1000	22 022	0
5000	144 862	0
10000	329 644	0
20000	719 241	0
50000	1 997 958	0

В таблиці 3.3 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 – Характеристики оцінювання алгоритму сортування гребінцем для зворотно упорядкованої послідовності елементів у масиві.

Розмірність масиву	Число порівнянь	Число перестановок
10	36	9
100	1 229	110
1000	22 022	1 512
5000	144 862	9 016
10000	329 644	19 132
20000	719 241	40 852
50000	1 997 958	109 958

У таблиці 3.4 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування гребінцем для масивів різної розмірності, масиви містять випадкову послідовність елементів.

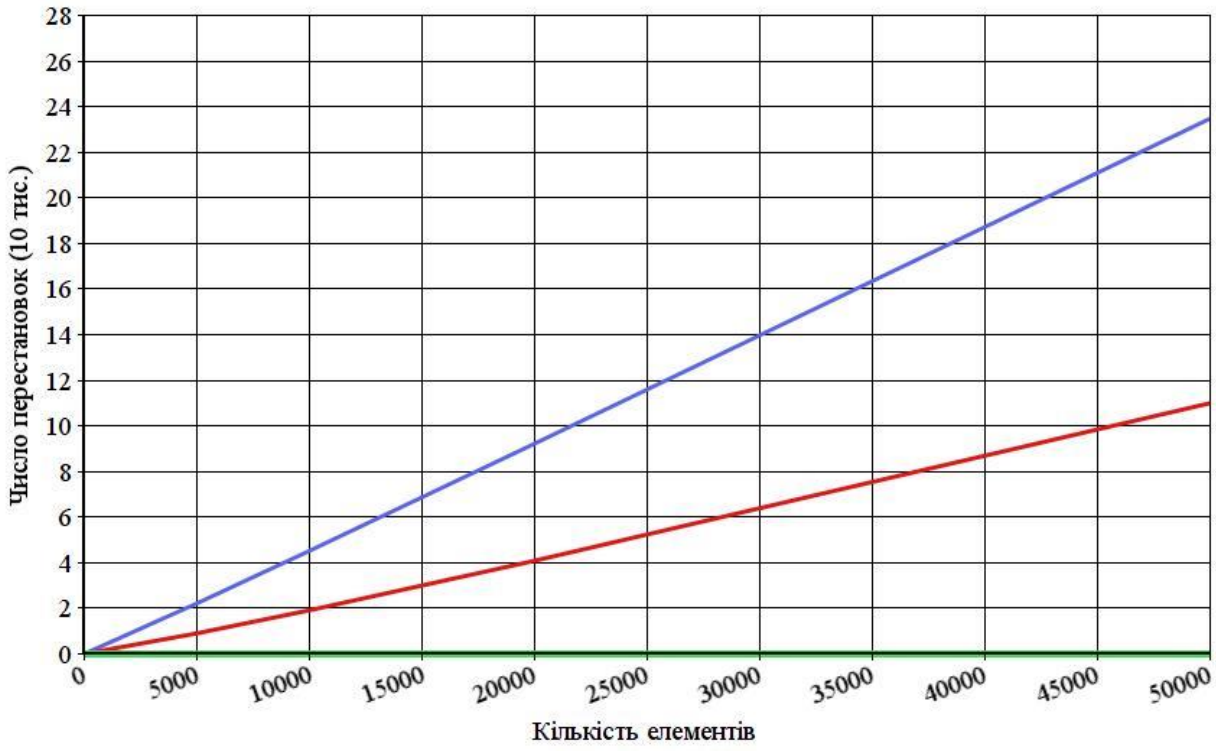
Таблиця 3.4 – Характеристика оцінювання алгоритму сортування гребінцем для випадкової послідовності елементів у масиві.

Розмірність масиву	Число порівнянь	Число перестановок
10	36	15
100	1 229	265
1000	22 022	4 060
5000	144 862	22 150
10000	329 644	45 196
20000	719 241	92 208
50000	1 997 958	234 748

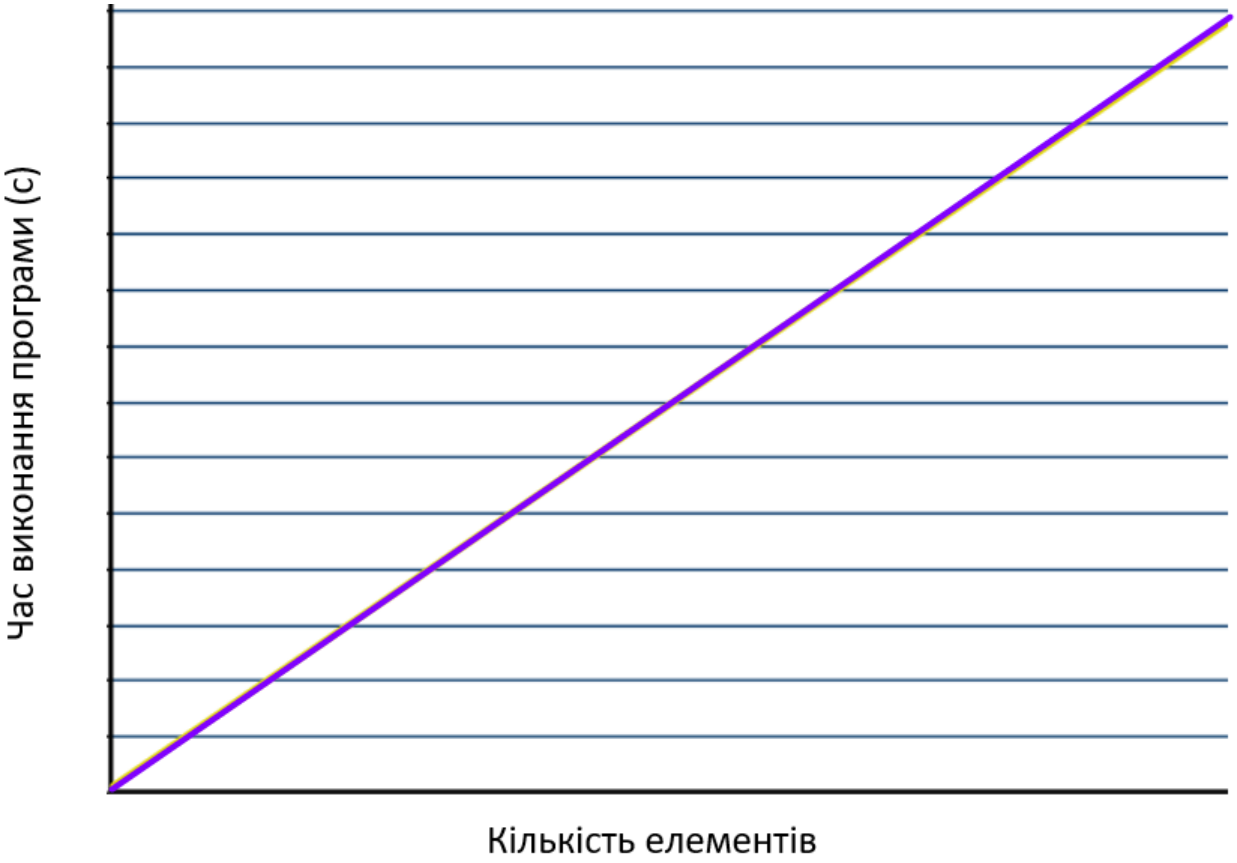
3.10.2 Графіки залежності часових характеристик оцінювання від розмірності масиву

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 – Графіки залежності часових характеристик оцінювання



Асимптотична оцінка гіршого випадку

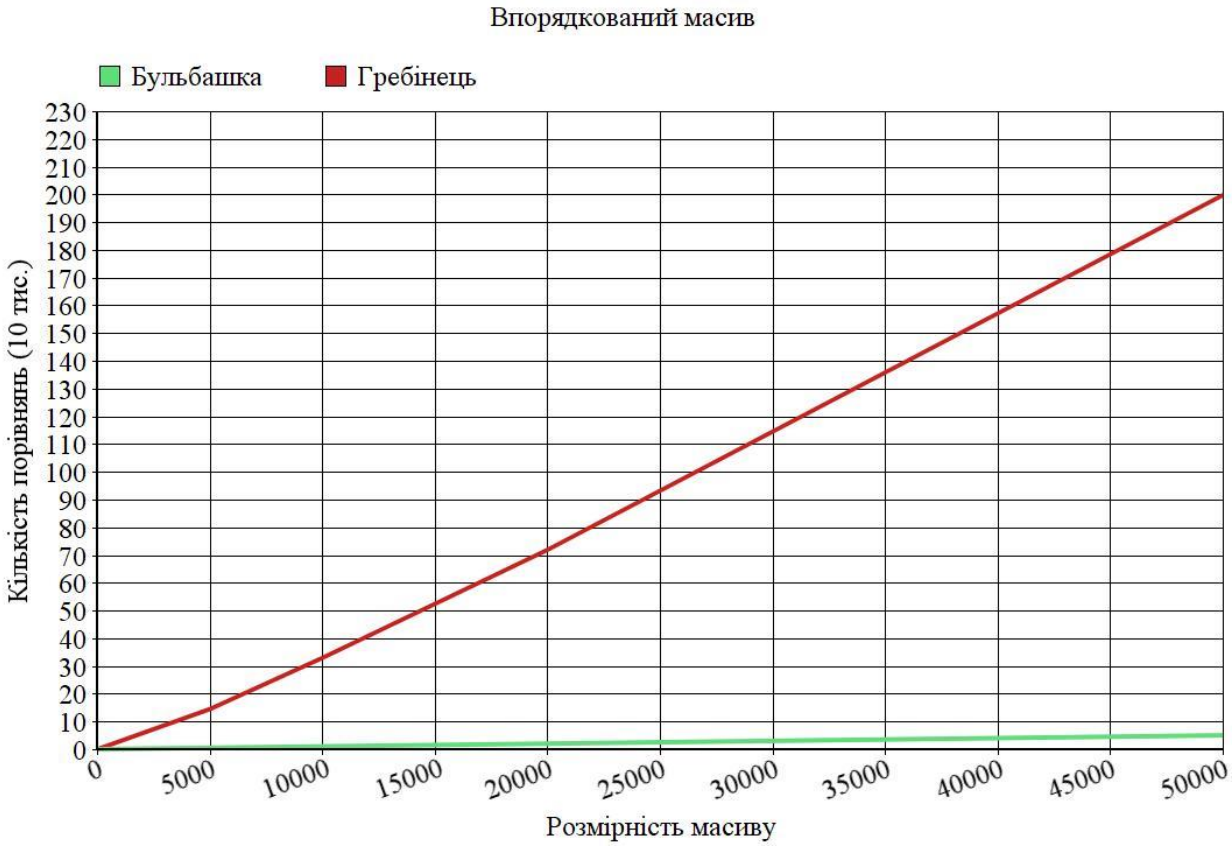


Порівняння алгоритмів

Оскільки у відсортованому масиві жоден алгоритм не робить перестановок (що є логічним) ми можемо порівняти їх за кількість порівнянь.

Спираючись на рисунок 3.4 можна зробити висновок, що на відсортованому сортування бульбашкою працює набагато швидше.

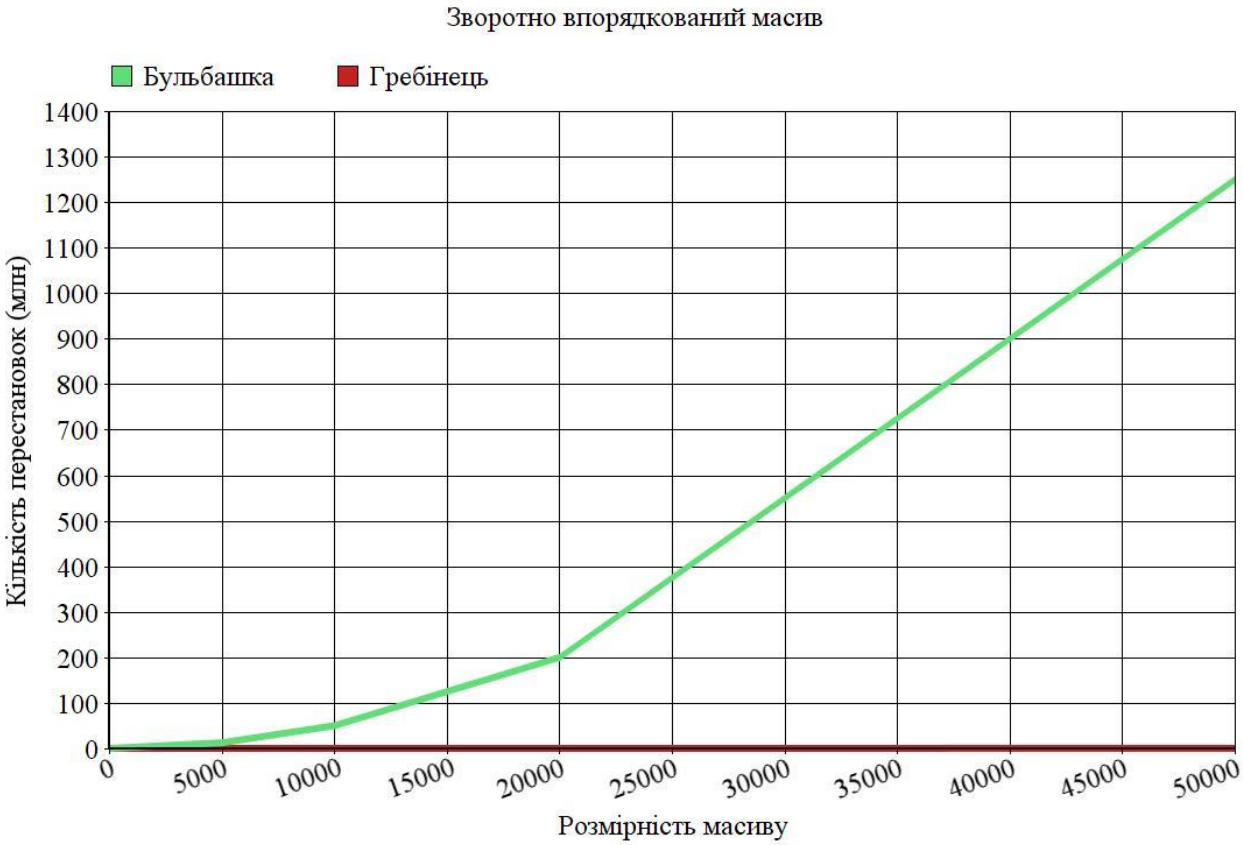
Рисунок 3.4



Щодо робити на зворотно впорядкованому масиві, то тут ми можемо порівняти роботу цих двох алгоритмів спираючись на кількість перестановок.

З рисунку 3.5 можна зрозуміти, що сортування гребінцем працює набагато швидше на зворотно впорядкованому масиві.

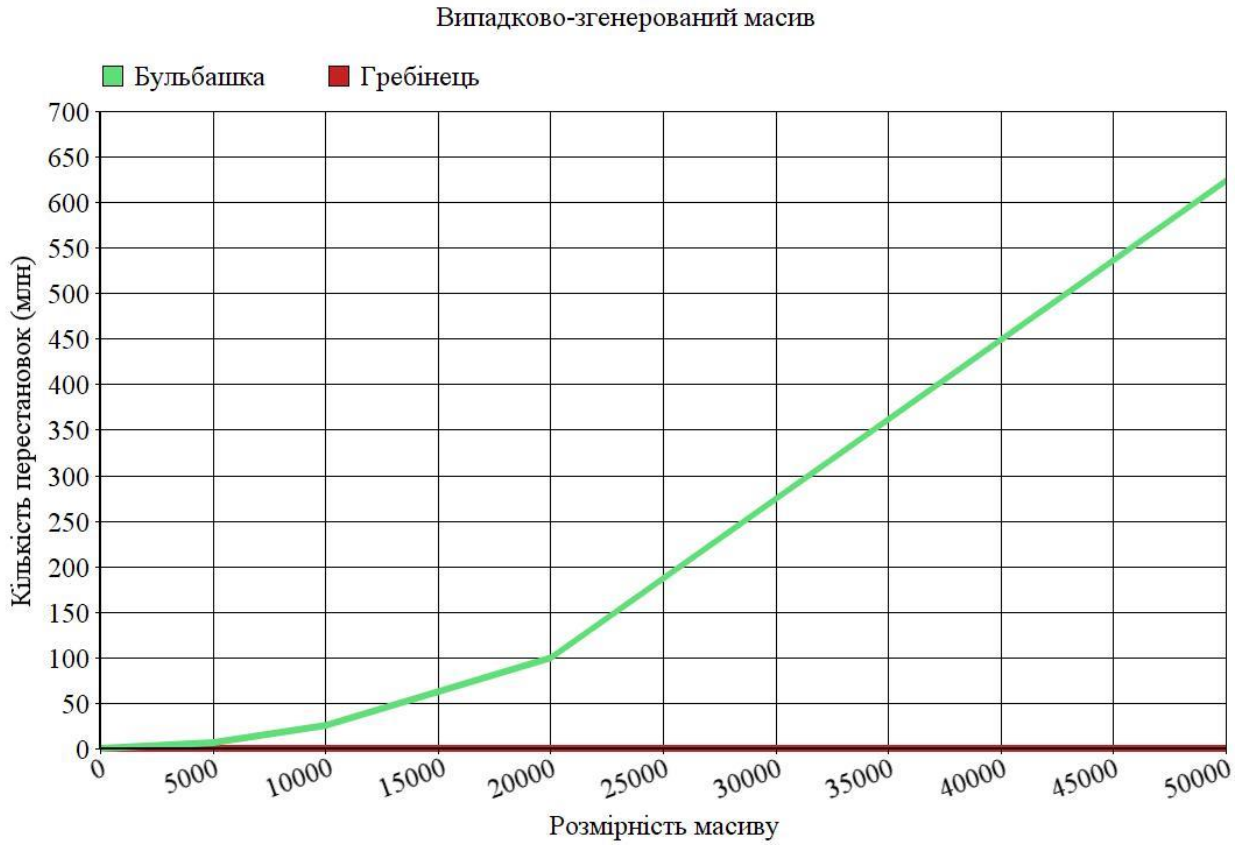
Рисунок 3.5



Тепер давайте порівняємо їх роботу на випадково-згенерованій послідовності.

На випадковій послідовності сортування гребінцем теж працює набагато швидше, що можна побачити на рисунку 3.6.

Рисунок 3.6



Висновок

При виконанні даної лабораторної роботи я ознайомився з основними методами аналізу обчислювальної складності алгоритмів внутрішнього сортування.

У її ході я порівняв два алгоритми : сортування бульбашкою та сортування гребінцем.

У процесі порівняння я дослідив їх властивості, здійснив аналіз їх поведінки на різних вхідних даних та візуалізував результат за допомогою графіків для простішого сприйняття.

Виявилося, що алгоритм сортування бульбашкою є ефективнішим лише при частково або повністю відсортованому масиві.

Виходячи з цього, можна вважати сортування гребінцем об’єктивно кращим у більшості випадків.