

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

**Підпрограма:** BubbleSort(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**

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 – Сортуння масиву на 100 елементів

Microsoft Visual Studio Debug Console

Рисунок 3.2 – Сортуння масиву на 1000 елементів

Microsoft Visual Studio Debug Console

2	2	3	4	4	7	7	7	8	8	9	11	12	12	14	14	14	15	16	16	18	20	23	25	25	26	26	26	27	27	28	28	30	31	33	33	34	36	37	38	42	42	4			
3	43	43	43	44	44	44	45	48	48	49	50	51	53	56	57	58	58	59	60	61	62	66	68	69	70	70	72	75	76	76	78	79	80	81	82	82	83	85	85	86	8				
6	87	89	89	92	93	94	95	95	95	96	97	97	97	99	102	103	104	105	107	107	108	109	113	113	115	115	115	115	117	117	119	119	119	121	121	122	122								
123	123	125	127	128	131	131	131	131	131	131	132	132	132	132	137	139	140	141	142	145	145	147	148	148	148	148	148	148	148	148	148	148	148	148	148	148	148	148	148	148	148	148	148		
157	159	160	161	163	163	164	171	173	175	176	176	177	178	178	179	180	181	181	181	182	182	185	186	186	186	186	186	186	186	186	186	186	186	186	186	186	186	186	186	186	186	186	186	186	
194	196	198	198	200	200	200	203	204	204	204	205	206	207	207	207	209	210	210	211	215	217	218	218	219	220	221	222	222	222	222	222	222	222	222	222	222	222	222	222	222	222	222	222	222	
225	225	225	226	228	228	228	228	228	230	232	232	232	235	236	238	239	239	239	239	240	240	242	242	242	242	243	243	243	243	243	243	243	243	243	243	243	243	243	243	243	243	243	243	243	243
248	250	252	253	254	255	256	256	258	259	260	261	263	263	263	264	265	265	267	267	267	268	268	268	272	274	275	276	279	280	281	281	281	281	281	281	281	281	281	281	281	281	281	281	281	281
281	281	281	283	284	284	286	287	287	288	290	293	294	296	297	298	298	300	301	303	306	307	307	307	308	308	309	309	309	309	309	309	309	309	309	309	309	309	309	309	309	309	309	309	309	309
314	314	314	315	316	316	317	318	318	320	321	322	322	323	323	328	328	329	331	332	332	334	336	337	339	340	342	343	343	343	343	343	343	343	343	343	343	343	343	343	343	343	343	343	343	343
345	345	346	347	350	352	353	356	357	357	361	362	362	364	364	370	376	376	377	380	385	385	385	385	387	387	387	387	387	38																
389	391	394	394	394	394	394	396	397	398	399	400	400	401	402	404	405	407	409	409	410	412	414	416	416	417	417	417	417	417	417	417	417	417	417	417	417	417	417	417	417	417	417	417	417	417
425	425	426	426	427	429	432	432	432	432	432	434	435	437	438	438	439	441	441	441	441	442	442	442	442	443	446	446	449	449	449	449	449	449	449	449	449	449	449	449	449	449	449	449	449	
454	456	457	457	458	458	458	459	460	461	462	463	464	464	466	468	468	470	471	473	473	475	475	476	478	479	481	482	482	482	482	482	482	482	482	482	482	482	482	482	482	482	482	482	482	
488	489	490	492	492	493	494	496	498	501	501	502	503	503	504	504	505	505	507	507	509	510	510	510	510	511	512	512	512	512	512	512	512	512	512	512	512	512	512	512	512	512	512	512	512	
514	514	514	515	515	521	521	522	525	529	529	530	531	532	532	532	533	534	534	535	537	538	543	543	543	543	543	543	544	545	545	545	545	545	545	545	545	545	545	545	545	545	545	545		
546	547	548	549	549	550	551	552	553	553	555	555	556	557	557	557	557	558	559	560	562	562	563	565	570	571	572	573	576																	
576	576	578	578	578	579	581	581	582	583	584	585	585	585	585	586	586	586	589	590	591	591	592	593	595	595	596	596	596	596	596	596	596	596	596	596	596	596	596	596	596	596	596	596		
600	600	600	602	603	604	605	606	608	609	609	610	610	610	611	612	613	613	614	614	617	618	618	619	620	620	620	621	621	621	621	621	621	621	621	621	621	621	621	621	621	621	621	621		
621	622	622	623	623	623	624	624	625	626	627	628	629	631	632	632	634	635	635	636	637	639	639	641	641	643	643	643	647	647	647	647	647	647	647	647	647	647	647	647	647	647	647			
652	652	652	653	653	655	655	660	660	661	662	662	664	666	667	668	668	671	672	672	673	673	676	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677			
679	679	679	679	679	679	680	681	682	682	683	683	687	689	689	689	689	690	691	691	691	693	695	695	695	695	695	695	695	695	695	695	695	695	695	695	695	695	695	695	695	695	695	695		
700	700	702	704	704	705	705	707	707	707	710	711	711	711	713	714	715	715	718	718	720	720	721	721	724	724	727	728	728																	
729	729	729	732	732	733	733	735	735	735	735	736	736	737	738	744	744	744	747	749	750	753	754	754	758	759	759	760	761	763																
763	764	765	765	765	768	769	771	772	774	774	777	777	778	779	780	780	782	783	784	785	786	788	790	790	791	793	797	797	797	797	797	797	797	797	797	797	797	797	797	797	797	797			
799	799	799	800	800	800	801	803	806	807	807	808	810	810	812	815	815	816	816	817	817	818	819	819	821	823	823	823	824	824																
825	828	829	829	831	832	833	834	835	836	836	838	838	840	840	840	840	841	841	843	843	843	844	844	845	846	848	848	848	850	853															
853	858	859	859	859	861	861	862	862	863	864	865	867	867	867	868	868	868	869	869	869	869	872	872	872	872	873	874	875	877	877															
878	878	879	880	880	881	881	882	884	887	888	890	891	893	893	896	899	900	900	900	901	901	902	903	903	903	903	905	905	907	907															
908	909	909	909	911	913	914	917	917	917	921	922	922	923	924	924	925	927	931	931	932	934	934	935	936	937	937	938																		
938	939	940	941	942	943	943	943	945	946	947	948	950	950	951	951	951	952	952	954	954	957	958	960	960	963	965	966	966	967																
968	970	973	974	974	975	975	976	976	977	977	979	981	981	982	983	983	983	983	984	984	986	986	986	987	989	989	989	990	991																
991	991	992	993	993	994	995	995	996	998	998	999																																		

```
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) {...}
40
41 private static void PrintArr(int[] arr) {...}
42
43 static void Main(string[] args)
44 {
45     int[] arr = CreateArray(10);
46     Console.WriteLine("Generated array : ");
47     PrintArr(arr);
48     Console.WriteLine();
49     BubbleSort(arr);
50     Console.WriteLine("Sorted array : ");
51     PrintArr(arr);
52     Console.WriteLine();
53 }
54
55
56
57
58
59
60
61
62
63
64
65
66
67
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)
40 {
41     int[] arr = new int[size];
42     Random rand = new Random();
43     for (int i = 0; i < arr.Length; i++)
44     {
45         arr[i] = rand.Next(0, 1000);
46     }
47     return arr;
48 }
49
50 private static void PrintArr(int[] arr)
51 {
52     Console.WriteLine(string.Join(" ", arr));
53 }
54
55 static void Main(string[] args)
56 {
57     int[] arr = CreateArray(20);
58     Console.WriteLine("Generated array : ");
59     PrintArr(arr);
60     Console.WriteLine();
61     BubleSort(arr);
62     Console.WriteLine("Sorted array : ");
63     PrintArr(arr);
64     Console.WriteLine();
65 }
66
67
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.

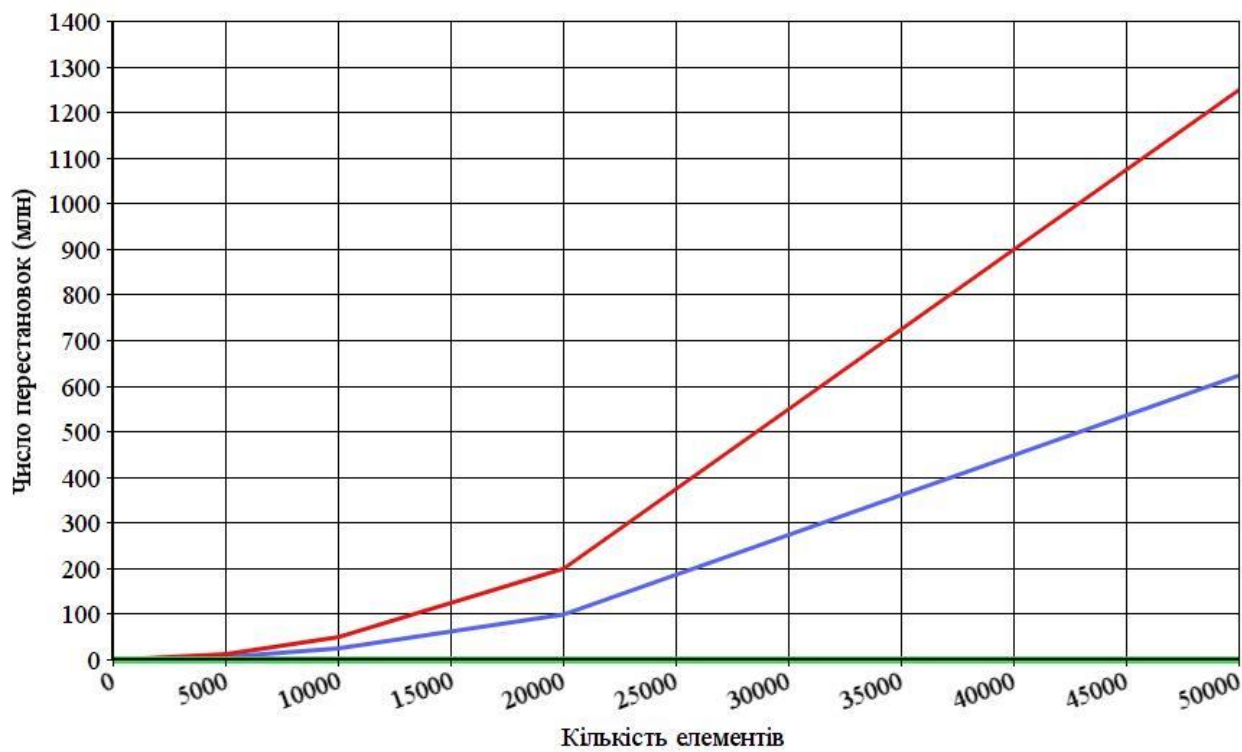
Таблиця 3.4 – Характеристика оцінювання алгоритму сортування бульбашки для випадкової послідовності елементів у масиві.

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

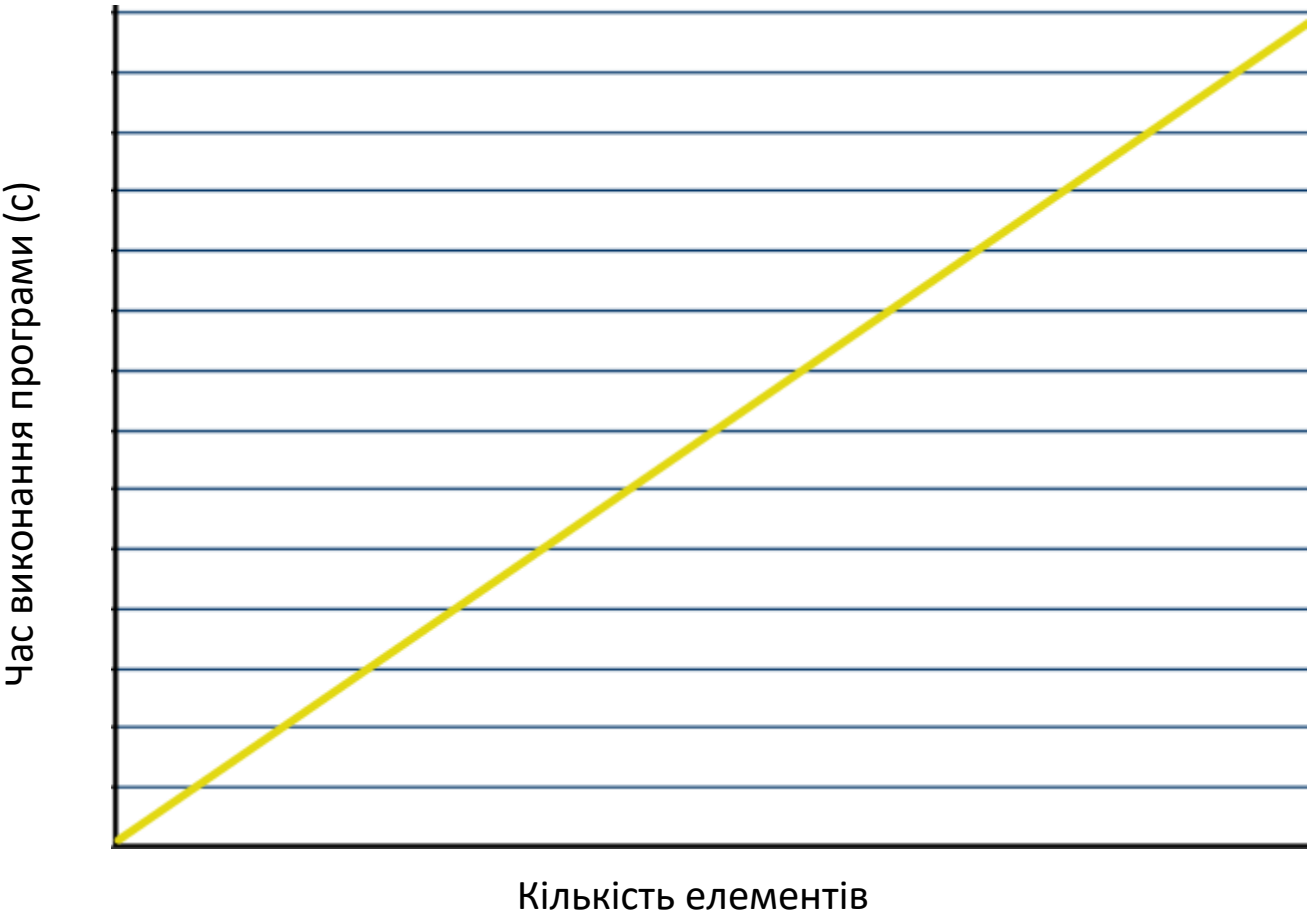
### 3.5.2 Графіки залежності часових характеристик оцінювання від розмірності масиву

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.

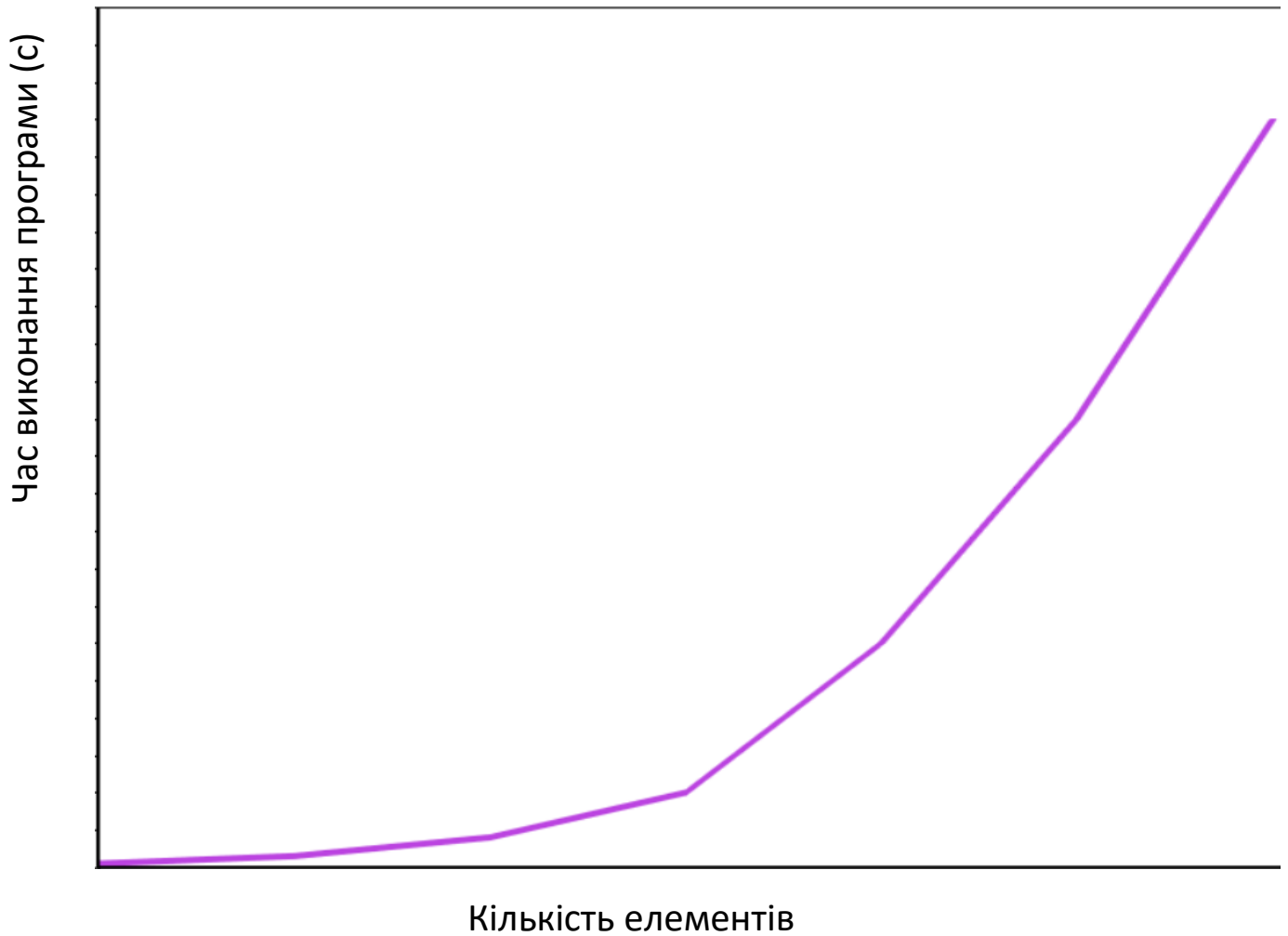
Рисунок 3.3 – Графіки залежності часових характеристик оцінювання



Асимптотична оцінка кращого випадку



### 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 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:  $O(n \cdot \log(n))$

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)
84 {
85     for (int i = 0; i < arr.Length; i++)
86     {
87         Console.Write(arr[i] + " ");
88         if (i % 10 == 9)
89             Console.WriteLine();
90     }
91 }
92
93 static void Main(string[] args)
94 {
95     int[] arr = CreateArray(50000);
96     Console.WriteLine("Generated array : ");
97     PrintArr(arr);
98     Console.WriteLine();
99     CombSort(arr);
100    Console.WriteLine("Sorted array : ");
101    PrintArr(arr);
102    Console.WriteLine();
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

```
generated array :
453 748 765 614 367 471 396 299 724 309 841 171 453 539 561 144 61 40 734 15 233 167 596 738 655 981 323 257 152 942 971
155 214 857 784 461 634 593 293 597 213 7 876 150 700 621 728 294 279 959 869 136 311 645 81 285 296 386 521 6 691 106
357 500 862 621 855 557 61 24 375 958 718 704 521 168 316 115 669 306 678 307 28 83 94 102 877 195 720 917 3 521 903 875
144 75 648 919 579 914

Compares : 1229
Swaps : 243

Sorted array :
3 6 7 15 24 28 40 61 61 75 81 83 94 102 106 115 136 144 144 150 152 155 167 168 171 195 213 214 233 257 279 285 293 294
296 299 306 307 309 311 316 323 357 367 375 386 396 453 461 471 500 521 521 521 539 557 561 579 593 596 597 614 621
621 634 645 648 655 669 678 691 700 704 718 720 724 728 734 738 748 765 784 841 855 857 862 869 875 876 877 903 914 917
919 942 958 959 971 981
```

Рисунок 3.2 – Сортування масиву на 1000 елементів

```
6 664 506 274 177 987 721 310 896 589 658 489 70 576 881 111 431 458 954 362 945 156 912 766 256 337 124 272 483 556 704 283 394 228 902 514 691 580 176 414 895 149 642 189 924 748 649 466 510 736 792 369 392 24
5 336 463 63 20 513 51 966 600 2 75 350 264 336 101 144 523 937 107 712 849 298 877 694 270 408 618 252 949 198 323 974 878 583 327 401 758 464 391 692 563 139 391 967 988 942 465 628 785 930 99 147 869 299 908
159 400 476 528 294 953 69 629 871 880 19 904 468 964 635 575 844 251 354 796 458 769 253 326 422 225 416 592 764 318 95 135 649 567 700 415 689 619 535 313 564 478 191 142 438 590 427 608 450 887 182 944 655 53
179 271 642 27 644 269 961 795 574 442 512 59 234 497 784 519 187 303 628 804 485 949 296 269 738 699 207 574 768 654 575 194 1 638 802 545 200 15 967 694 761 563 946 860 16 127 877 746 797 657 92 606 409 307 2
29 161 137 532 915 419 285 228 301 357 626 755 53 199 161 889 766 695 306 467 63 210 797 135 753 854 627 598 570 93 174 147 662 745 550 383 896 711 692 787 859 641 742 457 646 901 258 538 749 318 849 100 312 316
171 637 383 587 319 405 990 425 770 860 419 599 24 304 793 835 165 699 624 414 250 348 689 972 839 744 58 274 476 769 556 646 810 166 581 793 50 390 429 626 172 716 298 904 871 633 631 572 153 182 606 808 986 5
30 535 276 448 496 580 152 737 39 414 226 992 263 220 48 118 434 539 445 216 969 5 733 894 686 4 789 550 490 737 985 27 913 78 753 953 898 455 386 208 437 445 258 977 568 637 373 988 872 491 603 193 974 654 943
833 108 404 566 747 978 148 243 167 36 708 998 30 723 463 836 218 131 662 363 283 541 939 617 112 249 364 811 950 629 204 792 623 710 244 3 603 16 986 612 401 32 494 194 483 649 752 888 881 590 71 502 276 427 97
1 486 924 149 426 712 912 699 86 583 581 386 359 233 195 18 884 564 702 787 366 551 551 233 376 69 521 900 599 879 985 175 38 805 196 117 379 924 791 57 19 766 172 961 554 828 257 697 957 85 706 317 597 860 683
448 149 711 850 881 339 610 216 41 886 582 386 121 758 517 533 892 660 663 187 480 261 242 225 514 834 22 191 951 930 564 119 922 649 782 176 440 727 250 236 957 897 257 5 347 75 776 142 984 741 912 157 368 877
931 649 659 58 372 559 382 952 633 258 351 352 61 849 591 549 648 85 396 121 75 473 55 563 667 660 985 694 199 305 684 684 962 105 836 136 571 216 187 743 532 54 344 318 932 884 746 631 654 92 772 337 644 494 21
3 993 375 258 220 413 124 261 307 219 416 399 741 846 40 661 948 695 114 513 854 512 828 285 344 857 239 837 534 794 467 499 39 643 47 31 823 305 795 2 577 464 253 316 598 994 150 949 352 456 967 782 987 92 239
212 953 424 999 631 60 73 395 305 854 634 502 180 793 770 248 578 405 752 804 125 898 445 511 385 478 157 71 176 884 801 432 995 584 499 69 7 532 128 560 185 798 953 967 525 13 191 459 675 247 613 130 270 885 74
2 723 869 15 574 22 38 229 909 157 669 216 480 800 740 151 148 323 127 890 871 446 347 292 519 713 149 591 445 487 900 843 701 692 317 287 789 119 275 614 51 455 858 635 831 75 152 648 942 166 558 516 485 788 96
7 97 195 93 824 503 283 940 256 971 841 597 329 64 949 566 738 863 767 591 549 400 304 565 800 876 50 146 796 930 991 305 828 14 366 566 660 482 236 516 805 418 44 858 797 696 819 236 876 944 867 217 312 193 867
583 393 584 468 371 598 695 978 931 390 3 325 292 847 871 869 314 194 245 145 302 252 994 162 471 992

Compares : 22022
Swaps : 3945

Sorted array :
1 1 2 2 2 2 3 3 4 5 5 7 13 14 15 15 16 16 18 19 19 20 22 22 24 27 27 30 31 32 33 36 38 38 39 39 40 40 41 44 46 47 48 50 50 51 51 53 53 54 55 57 58 58 59 60 61 63 63 63 64 69 69 69 70 71 71 73 75 75 75 75 76 77 7
8 85 85 86 91 92 92 92 93 93 95 97 99 100 101 105 107 108 111 112 114 114 117 118 119 119 121 121 123 124 124 125 127 127 127 128 130 130 131 135 135 136 137 139 142 142 144 145 146 147 147 148 148 149 149 149 1
49 150 151 152 152 152 153 156 156 157 157 157 159 159 161 161 162 163 165 166 166 167 168 171 172 172 174 175 176 176 176 177 179 180 182 182 185 187 187 187 189 191 191 191 192 193 193 194 194 194 195
195 195 195 196 198 199 199 200 204 204 207 208 208 210 212 213 216 216 216 216 217 218 219 220 220 221 222 225 225 226 228 228 229 229 230 233 233 234 234 236 236 236 236 237 238 239 239 242 243 244 244 245 245
247 248 249 250 250 251 252 252 253 253 256 256 257 257 257 258 258 258 259 261 261 262 262 263 264 269 269 270 270 271 272 274 274 275 276 276 283 283 283 285 285 287 289 292 292 294 294 295 296 298 29
8 299 299 301 302 303 304 304 305 305 305 305 306 307 307 309 310 312 312 313 314 314 316 316 317 317 318 318 318 318 319 323 323 325 326 326 327 329 330 334 336 336 337 337 339 344 344 344 347 347 348 348 350 3
51 351 351 352 352 354 355 356 356 357 359 359 360 362 363 363 364 366 366 368 369 369 371 372 373 374 375 376 378 378 379 382 383 383 385 386 386 386 388 389 390 390 391 391 392 393 394 395 395 396 396 399 399
400 400 401 401 401 404 405 405 408 409 409 413 414 414 414 415 416 418 416 419 419 422 424 425 425 425 426 427 427 428 429 430 431 432 434 437 438 440 442 444 445 445 445 445 446 448 448 450 453 455 455 456
456 457 458 458 459 463 463 463 464 464 465 466 466 467 467 468 468 469 471 473 473 476 476 478 478 480 480 482 483 483 485 485 486 487 489 490 491 494 494 496 497 499 499 501 502 502 503 506 510 511 512 512 51
3 513 514 514 516 516 516 517 519 519 521 523 525 528 530 532 532 532 533 534 535 535 538 539 540 541 545 549 549 550 550 551 551 551 553 554 556 556 558 559 560 563 563 563 564 564 564 565 566 566 566 567 5
68 568 569 569 570 571 572 573 573 574 574 574 575 575 576 577 578 580 580 581 581 582 583 583 583 584 584 584 584 585 587 587 589 590 590 591 591 591 592 592 597 597 598 598 598 599 599 600 600 601 602 603 603
606 606 608 608 610 610 611 612 613 614 615 617 617 618 619 623 624 626 626 627 628 628 629 629 631 631 631 633 633 634 635 635 636 637 637 638 639 640 641 641 642 642 643 643 644 644 646 646 648 648 649 649 649
649 649 654 654 654 655 656 657 658 659 660 660 661 662 662 663 664 667 669 675 679 683 684 684 684 685 686 689 689 691 692 692 692 694 694 694 695 695 695 696 697 698 699 699 700 701 702 704 706 708 71
0 711 711 712 712 712 713 714 716 717 717 717 723 723 727 733 736 737 737 738 738 738 740 741 741 742 742 743 744 745 746 746 746 747 748 749 750 752 752 752 753 753 754 755 758 758 760 761 763 764 765 766 766 767 768 7
69 769 770 770 772 772 776 777 778 782 782 784 784 785 787 787 788 789 789 790 791 791 792 792 793 793 793 794 795 795 795 796 796 797 797 797 798 800 800 801 802 804 804 805 805 805 807 808 810 811 814 819 823 824 826
828 828 828 831 833 834 834 835 836 836 837 837 839 841 843 844 846 847 849 849 849 849 850 854 854 854 854 857 858 858 859 860 860 860 861 863 867 867 869 869 869 871 871 871 871 872 876 876 877 877 877 878 879 880
881 881 881 884 884 884 885 885 886 886 887 888 888 889 890 890 891 892 894 895 896 896 896 897 898 898 900 900 901 901 902 904 904 908 909 912 912 912 913 913 915 915 916 920 922 924 924 924 925 927 929 930 93
9 930 931 931 932 937 939 940 942 942 943 944 944 944 945 946 946 948 949 949 949 949 950 951 952 953 953 953 953 954 957 957 961 961 962 964 966 967 967 967 967 969 970 971 971 972 974 974 977 977 978 978 97
81 984 985 985 985 985 986 986 987 987 988 988 988 990 991 992 992 992 992 993 994 994 995 997 998 999
```

### 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.

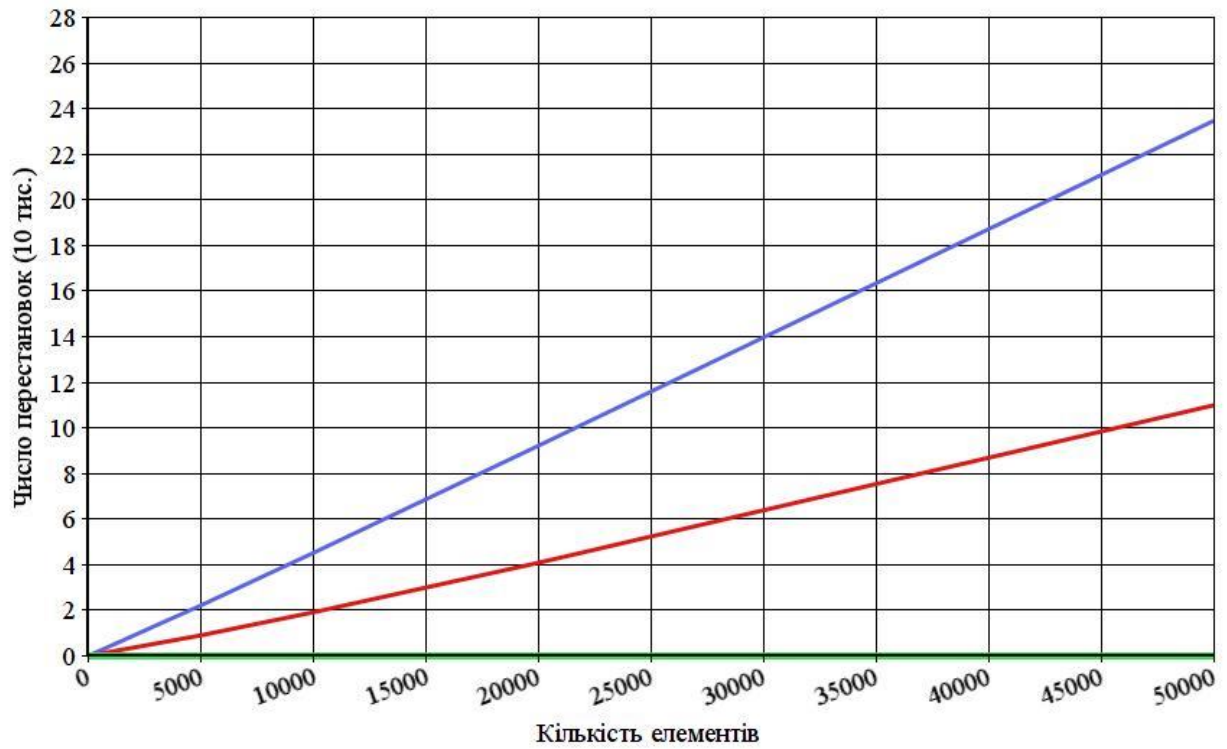
Таблиця 3.4 – Характеристика оцінювання алгоритму сортування гребінцем для випадкової послідовності елементів у масиві.

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

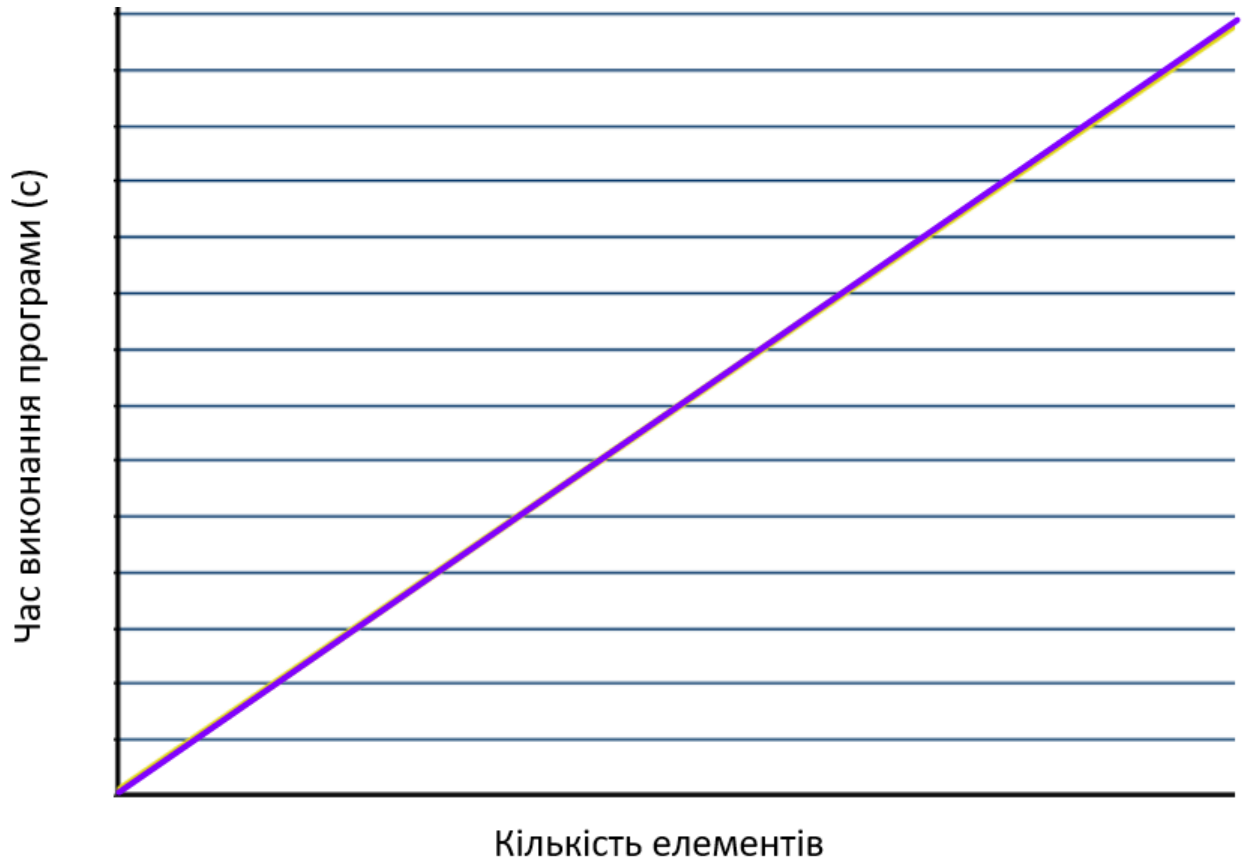
### 3.10.2 Графіки залежності часових характеристик оцінювання від розмірності масиву

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

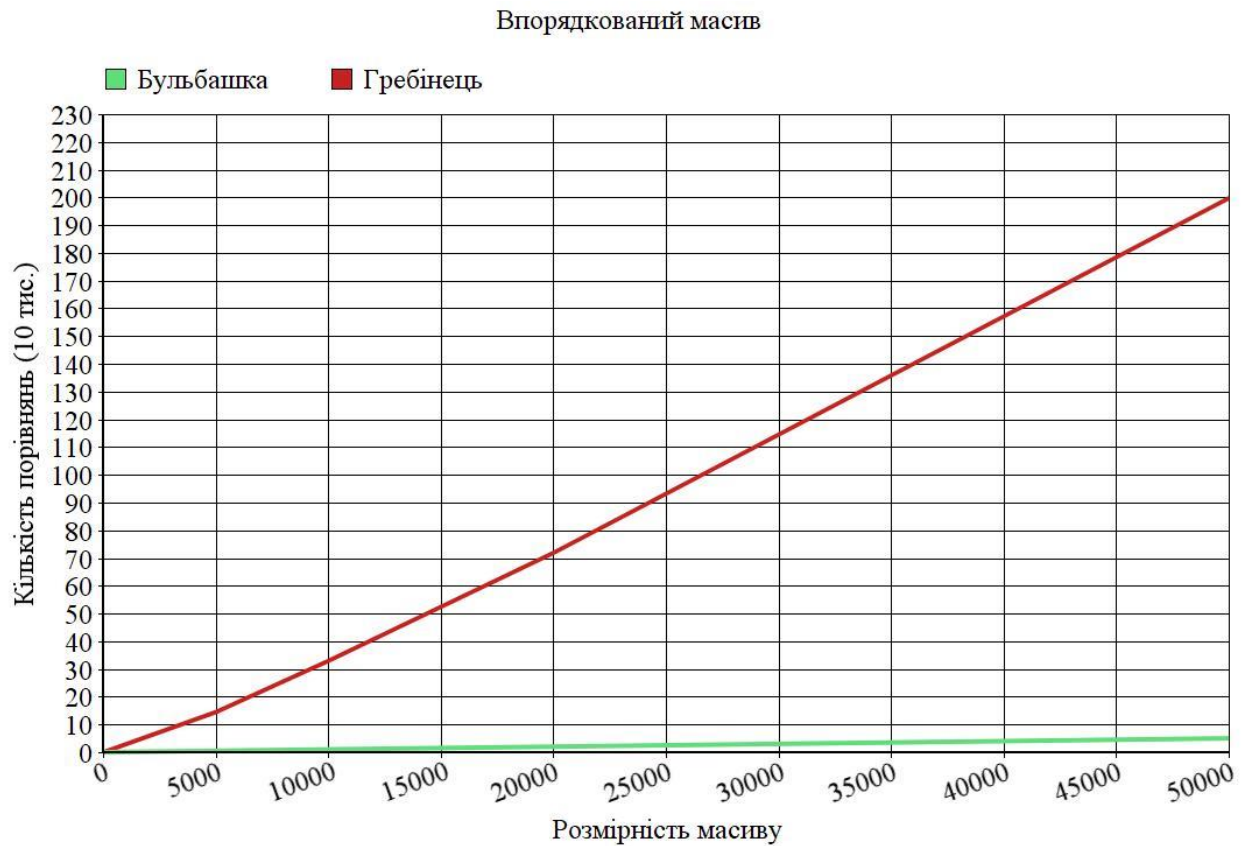


### Порівняння алгоритмів

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.

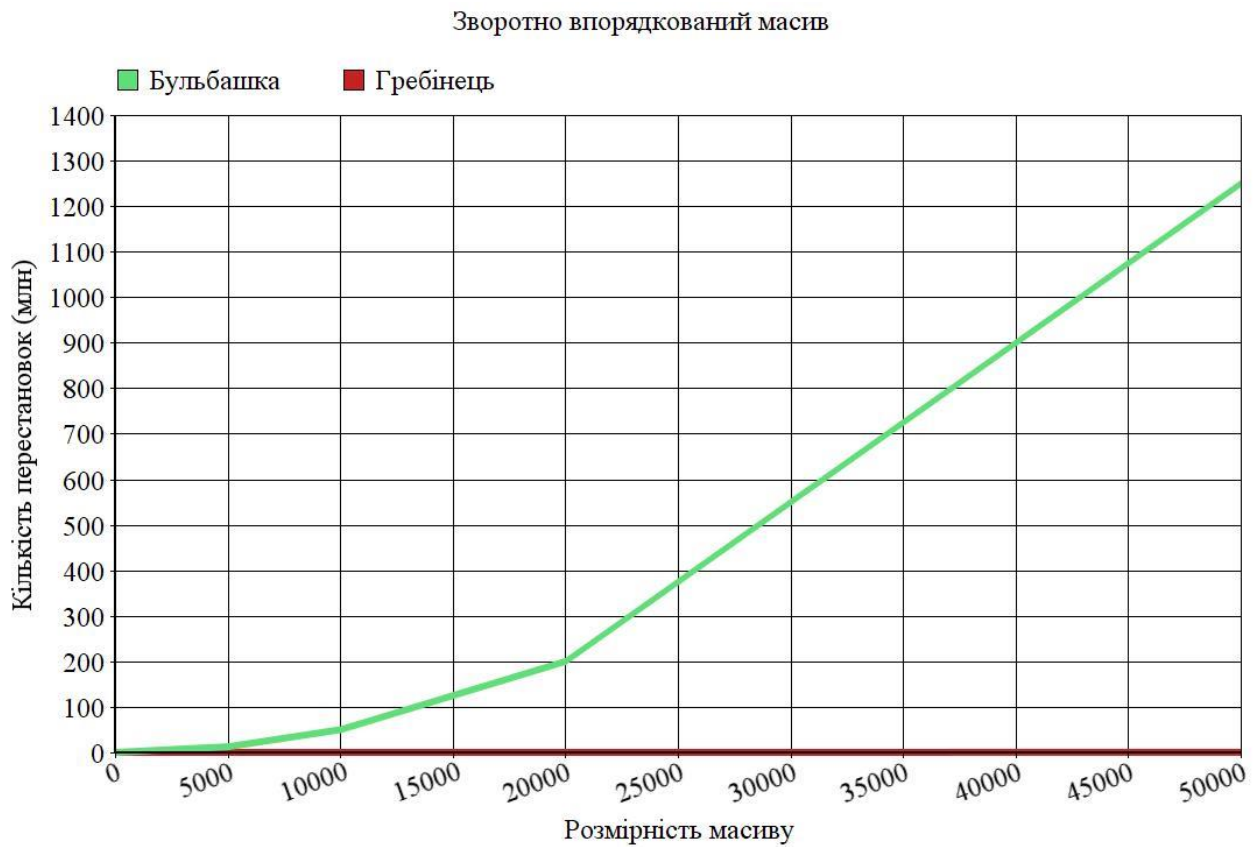
Рисунок 3.4



Щодо роботи 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

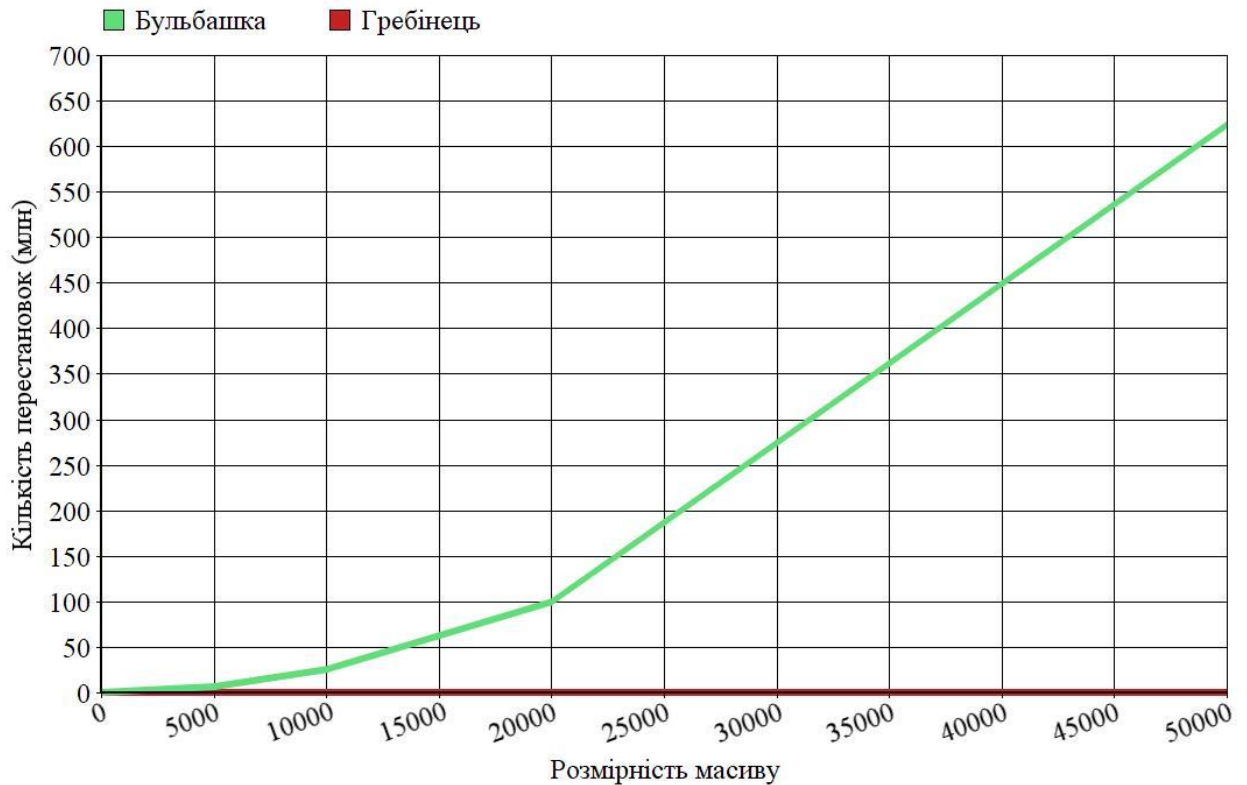


Тепер давайте порівняємо їх роботу на випадково-згенерованій послідовності.

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

Рисунок 3.6

### Випадково-згенерований масив



### 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.