

Алгоритми та структури даних 2. Структури даних

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»

Варіант 14

Виконав студент: ПІ-15 Кондрацька Соня Леонідівна

Перевірів: Соколовський Владислав Володимирович

Київ-2022

МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – 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

1.1 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 стійкий
«Природність» поведінки (Adaptability)	Алгоритм є природним
Базуються на порівняннях	Алгоритм базується на порівняннях
Необхідність в додатковій пам'яті (об'єм)	Не потребує додаткової пам'яті
Необхідність в знаннях про структури даних	Для використання алгоритму потрібно мати базові знання про структури даних

1.2 Псевдокод алгоритму

Підпрограма sort(arr, size)

Початок

flag: = 0

повторити для i від 0 до size

повторити для j від 0 до size-i-1

якщо arr[j] > arr[j + 1] **то**

flag:=+1

temp: = arr[j]

arr[j]: = arr[j + 1]

arr[j + 1]: = temp

все якщо
все повторити
якщо flag== 0 **то**
 break

все якщо
все повторити

Кінець

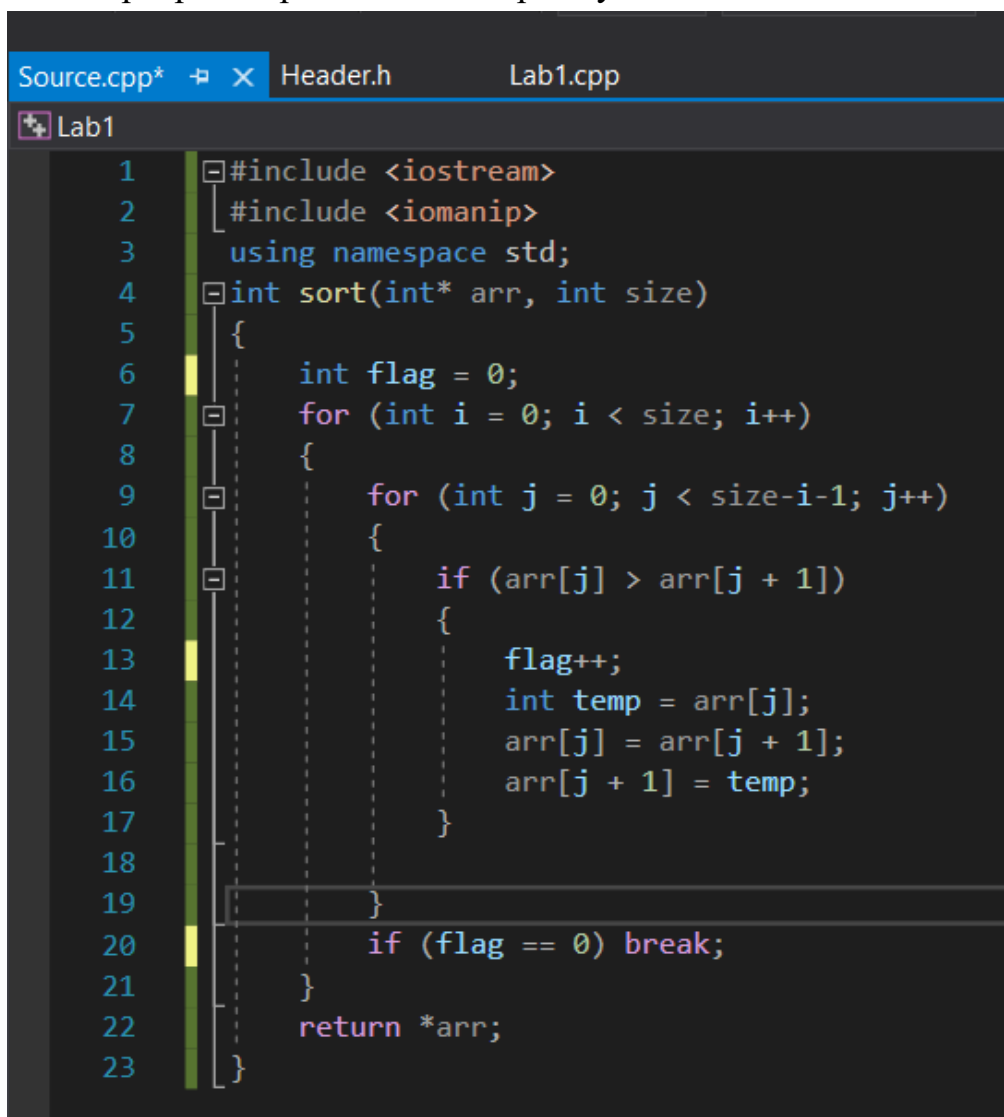
1.3 Аналіз часової складності

Найгірший випадок: $O(n^2)$

Середній випадок: $O(n^2)$

Найкращий випадок: $O(n)$

1.4 Програмна реалізація алгоритму



```
Source.cpp*  Header.h  Lab1.cpp
Lab1
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4  int sort(int* arr, int size)
5  {
6      int flag = 0;
7      for (int i = 0; i < size; i++)
8      {
9          for (int j = 0; j < size-i-1; j++)
10         {
11             if (arr[j] > arr[j + 1])
12             {
13                 flag++;
14                 int temp = arr[j];
15                 arr[j] = arr[j + 1];
16                 arr[j + 1] = temp;
17             }
18         }
19     }
20     if (flag == 0) break;
21 }
22 return *arr;
23 }
```

1.4.1 Вихідний код

```
Source.cpp Header.h Lab1.cpp
Lab1
1 #pragma once
2 #include <iostream>
3 using namespace std;
4 int sort(int*, int);
5

Source.cpp Header.h Lab1.cpp
Lab1
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4 int sort(int* arr, int size)
5 {
6     int flag = 0;
7     for (int i = 0; i < size; i++)
8     {
9         for (int j = 0; j < size-i-1;
10             {
11                 if (arr[j] > arr[j + 1])
12                 {
13                     flag++;
14                     int temp = arr[j];
15                     arr[j] = arr[j + 1];
16                     arr[j + 1] = temp;
17                 }
18             }
19         if (flag == 0) break;
20     }
21     return *arr;
22 }
23

Source.cpp Header.h Lab1.cpp
Lab1
1 #include <iostream>
2 #include "Header.h"
3 using namespace std;
4 int main()
5 {
6     int r = 1000;
7     int* A = new int[r];
8     for (int i = 0; i < r; i++)
9     {
10         A[i] = rand()%1000+1;
11     }
12     cout << "Start array:" << endl;
13     for (int i = 0; i < r; i++)
14     {
15         cout << A[i] << " ";
16     }
17     sort(A, r);
18     cout << endl<<"Sort array:" << endl;
19     for (int i = 0; i < r; i++)
20     {
21         cout << A[i] << " ";
22     }
23     delete[] A;
24 }
```

1.4.2 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

```
Start array:
42 68 35 1 70 25 79 59 63 65 6 46 82 28 62 92 96 43 28 37 92 5 3 54 93 83 22 17 19 96 48 27 72 39 70
13 68 100 36 95 4 12 23 34 74 65 42 12 54 69 48 45 63 58 38 60 24 42 30 79 17 36 91 43 89 7 41 43 6
5 49 47 6 91 30 71 51 7 2 94 49 30 24 85 55 57 41 67 77 32 9 45 40 27 24 38 39 19 83 30 42
Sort array:
1 2 3 4 5 6 6 7 7 9 12 12 13 17 17 19 19 22 23 24 24 24 25 27 27 28 28 30 30 30 30 32 34 35 36 36
37 38 38 39 39 40 41 41 42 42 42 42 43 43 43 45 45 46 47 48 48 49 49 51 54 54 55 57 58 59 60 62 63 63
65 65 65 67 68 68 69 70 70 71 72 74 77 79 79 82 83 83 85 89 91 91 92 92 93 94 95 96 96 100
```

Рисунок 3.2 – Сортування масиву на 1000 елементів

Число перестановок та перевірок для:

а) Найкращого випадку

Chislo perevirok: 999

Chislo perestanovok: 0

б) Найгіршого випадку

Chislo perevirok: 333333000

Chislo perestanovok: 499500

в)ВИПАДКОВОГО

Chislo perevirok: 196363995

Chislo perestanovok: 253384

```
Консоль отладки Microsoft Visual Studio
41 467 334 500 169 724 478 358 962 464 705 145 281 827 961 491 995 942 827 436 391 604 902 153 292 382 421 716 718 895 447 726 771 538 869 912 667 299 35 894 703 811 322
333 673 664 141 711 253 868 547 644 662 757 37 859 723 741 529 778 316 35 190 842 288 106 40 942 264 648 446 805 890 729 370 350 6 101 393 548 629 623 84 954 756 840
966 376 931 308 944 439 626 323 537 538 118 82 929 541 833 115 639 658 704 930 977 306 673 386 21 745 924 72 270 829 777 573 97 512 986 290 161 636 355 767 655 574 31 5
2 350 150 941 724 966 430 107 191 7 337 457 207 753 383 045 009 209 758 221 588 422 946 506 30 413 168 900 591 762 655 410 359 624 537 540 483 595 41 602 350 291 836 37
4 20 596 211 348 199 660 484 281 734 53 999 418 938 900 788 127 467 728 893 648 483 807 421 310 617 813 514 309 616 935 451 600 269 519 556 798 303 224 8 844 609 989 702
195 485 93 343 523 587 314 503 448 200 458 618 580 796 798 281 589 798 0 157 472 622 538 292 38 179 190 657 958 191 815 888 156 511 202 634 272 55 328 646 362 886 875
433 869 142 844 416 881 998 322 651 21 699 557 476 892 389 75 712 600 510 3 869 861 688 401 789 255 423 2 585 182 285 88 426 617 757 832 932 169 154 721 189 976 329 36
8 692 425 555 434 549 441 512 145 60 718 753 139 423 279 996 687 529 549 437 866 949 193 195 297 416 286 105 488 282 455 734 114 701 316 671 786 263 313 355 185 53 912
808 832 945 313 756 321 558 646 982 481 144 196 222 129 161 535 450 173 466 44 659 292 439 253 24 154 510 745 649 186 313 474 22 168 18 787 905 958 391 202 625 477 414
314 824 334 874 372 159 833 70 487 297 518 177 773 270 763 668 192 985 102 480 213 627 802 99 527 625 543 924 23 972 61 181 3 432 505 593 725 31 492 142 222 286 64 900
187 360 413 974 270 170 235 833 711 760 896 667 285 550 140 694 695 624 19 125 576 694 658 302 371 466 678 593 851 484 18 464 119 152 800 87 60 926 10 757 170 315 576
227 43 750 164 109 882 86 565 487 577 474 625 627 629 928 423 520 902 962 123 596 727 261 105 595 264 260 202 116 30 326 11 771 411 547 153 520 790 924 188 763 940 851
562 829 900 713 958 778 365 7 477 200 58 439 303 760 357 324 477 108 113 887 801 850 460 428 993 384 405 540 111 704 835 356 72 350 823 485 556 216 626 357 526 357 337
271 869 361 896 22 617 112 717 696 585 41 423 129 229 565 559 432 296 855 53 962 584 734 654 972 457 369 532 963 607 483 911 635 67 848 675 938 223 142 754 511 741 175
459 825 221 870 626 934 205 783 850 398 279 701 193 734 637 534 556 993 176 705 962 548 881 300 413 641 855 855 142 462 611 877 424 678 752 443 296 673 40 313 875 72 818
610 17 932 112 695 169 831 40 488 685 90 497 589 990 145 353 314 651 740 44 258 335 759 192 605 264 181 503 829 775 608 292 997 549 556 561 627 467 541 129 240 813 174
601 77 215 683 213 992 824 601 392 759 670 428 27 84 75 786 498 970 287 847 604 503 221 663 706 363 10 171 489 240 164 542 619 913 591 704 818 232 750 205 975 539 303
422 98 247 584 648 971 864 913 75 545 712 546 678 769 262 519 985 289 944 865 540 245 508 318 870 601 323 132 472 152 87 570 763 901 103 423 527 600 969 15 565 28 543
347 88 943 637 409 463 49 681 588 342 608 60 221 758 954 888 146 690 949 843 430 620 748 67 536 783 35 226 185 38 853 629 224 748 923 359 257 766 944 955 318 726 411 25
355 1 549 496 584 515 964 342 75 913 142 196 948 72 426 606 173 429 404 705 626 812 375 93 505 36 736 141 814 994 256 652 936 830 482 355 15 131 230 841 625 11 637 18
6 690 650 662 634 893 353 416 452 8 262 233 454 303 634 303 256 148 124 317 213 109 28 200 80 218 859 50 155 361 264 903 676 643 909 902 561 489 948 202 653 674 220 402
923 831 369 878 259 8 619 971 3 945 781 504 392 685 313 698 589 722 938 37 410 461 234 508 961 959 493 515 269 937 869 58 700 971 264 117 215 555 815 330 39 212 288 8
2 954 85 710 484 774 380 815 951 541 115 679 110 898 73 788 977 132 956 689 113 8 941 790 723 363 28 184 778 200 71 885 974 71 333 867 153 295 168 825 676 629 650 598
309 693 686 80 116 249
Chislo perevirok: 196363995
Chislo perestanovok: 253384

Sort array:
1 2 3 3 3 3 6 7 7 8 8 8 8 9 10 10 11 11 15 15 17 18 18 19 20 21 21 21 22 22 23 24 25 27 28 28 30 30 31 31 35 35 35 36 37 37 38 38 39 40 40 40 41 41 41 43
44 44 49 50 52 53 53 53 55 58 58 60 60 60 61 64 67 67 70 71 71 72 72 72 72 73 75 75 75 75 77 80 80 82 82 84 84 85 86 87 87 88 88 90 93 93 97 98 99 101 102 103
105 106 107 108 109 109 110 111 112 112 113 113 114 115 115 116 116 117 118 119 123 124 125 127 129 129 129 131 132 132 139 140 141 141 141 142 142 142 142 144 145 145 1
45 146 148 150 152 152 153 153 153 154 154 155 156 157 159 161 161 164 164 168 168 168 169 169 169 170 170 171 173 173 174 175 176 177 179 181 181 182 184 185 185 186 18
6 187 188 189 190 190 191 191 192 192 193 193 195 195 195 196 196 199 200 200 200 200 202 202 202 205 205 209 212 213 213 213 215 215 216 220 221 221 221 222 222 223
224 224 226 227 229 230 232 233 234 235 240 240 245 247 249 249 253 253 255 256 256 257 258 259 260 261 262 262 263 264 264 264 264 264 269 270 270 270 271 272 279 279
281 281 281 282 282 285 285 286 287 287 288 288 289 290 291 292 292 292 295 296 296 297 297 299 300 302 303 303 303 303 303 306 308 309 309 310 311 311 312 312
313 314 314 314 315 316 316 317 318 318 321 322 322 323 323 324 326 328 329 330 333 333 334 334 335 337 337 342 342 343 347 348 350 350 350 350 353 353 355 355 355 3
35 356 357 357 357 358 359 359 360 361 361 362 363 363 365 368 369 369 370 371 372 374 375 376 380 382 383 384 386 389 391 391 392 392 393 398 401 402 404 405 409 410 41
0 411 411 413 413 413 414 414 416 416 416 418 421 421 422 422 423 423 423 423 424 425 426 426 428 428 429 430 430 432 433 434 436 437 439 439 439 441 443 446 447 448 450
451 452 454 455 457 457 458 459 460 461 462 463 464 464 466 466 467 467 467 472 472 474 474 476 477 477 477 478 480 481 482 483 483 483 484 484 484 484 485 485 487 487 488
488 489 489 491 492 493 496 497 498 500 503 503 503 504 505 506 508 508 510 510 511 511 512 512 514 515 515 518 519 519 520 520 523 525 526 527 527 529 529 532 534 535
536 537 537 538 538 539 540 540 541 541 541 542 543 543 545 546 547 547 548 548 548 549 549 549 549 550 555 555 555 556 556 556 557 558 559 561 561 565 565 565 565 5
70 573 574 576 576 577 578 580 584 584 584 585 585 587 588 588 589 589 591 591 593 593 595 596 596 598 600 600 601 601 601 602 604 604 605 606 607 608 608 609 61
0 611 616 617 617 617 618 619 619 620 622 623 624 624 625 625 625 625 626 626 627 627 627 629 629 629 629 629 634 634 635 636 637 637 637 639 641 643 644 646 646
648 648 648 649 650 650 651 652 653 654 655 655 657 658 658 659 662 662 662 663 664 667 667 668 668 670 671 673 673 673 674 675 676 676 676 678 678 679 679 681 682 685
685 686 687 688 689 690 690 692 693 694 694 695 695 696 698 699 700 701 701 702 703 704 704 704 705 705 705 706 710 711 711 712 712 713 716 717 718 718 721 722 723 723
724 724 725 726 726 728 729 734 734 734 734 736 737 740 741 741 745 745 748 748 750 752 753 753 754 756 756 757 757 757 757 758 758 758 759 759 760 760 762 763 763 763 766 7
67 769 771 771 773 774 775 777 778 778 781 783 786 786 787 788 788 789 790 790 796 798 798 798 800 801 802 805 807 808 811 812 813 813 814 815 815 815 818 818 823 82
4 824 825 825 827 827 829 829 829 831 831 832 832 833 833 835 836 838 840 841 842 843 844 844 847 848 850 850 851 851 853 855 855 855 858 859 861 864 865 866 867 868
869 869 869 869 869 870 870 874 875 875 877 878 881 881 882 882 885 886 887 888 888 890 892 893 893 894 895 896 896 898 900 900 900 900 901 902 902 902 903 905 909 909 911
```

1.5 Тестування алгоритму

Додатково протестуємо алгоритм сортування бульбашкою на випадково згенерованому масиві з розмірністю 20 елементів (елементи можуть повторюватись).

Код:

```
1 #include <iostream>
2 #include "Header.h"
3 using namespace std;
4 int main()
5 {
6     int r = 20;
7     int* A = new int[r];
8     for (int i = 0; i < r; i++)
9     {
10         A[i] = rand()%20+1;
11     }
12     cout << "Start array:" << endl;
13     for (int i = 0; i < r; i++)
14     {
15         cout << A[i] << " ";
16     }
17     sort(A, r);
18     cout << endl<<"Sort array:" << endl;
19     for (int i = 0; i < r; i++)
20     {
21         cout << A[i] << " ";
22     }
23     delete[] A;
24 }
```

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4 int sort(int* arr, int size)
5 {
6     int flag = 0;
7     for (int i = 0; i < size; i++)
8     {
9         for (int j = 0; j < size-i-1; j++)
10        {
11            if (arr[j] > arr[j + 1])
12            {
13                flag++;
14                int temp = arr[j];
15                arr[j] = arr[j + 1];
16                arr[j + 1] = temp;
17            }
18        }
19        if (flag == 0)
20        {
21            break;
22        }
23    }
24    return *arr;
25 }
```

Результат виконання програми:

```
Start array:
2 8 15 1 10 5 19 19 3 5 6 6 2 8 2 12 16 3 8 17
Sort array:
1 2 2 3 3 5 5 6 6 8 8 8 10 12 15 16 17 19 19
```

Як бачимо, програма коректно відсортувала випадково згенерований масив. Отже, наш алгоритм сортування бульбашкою працює правильно.

1.5.1 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	4 950	950
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
--------------------	-----------------	--------------------

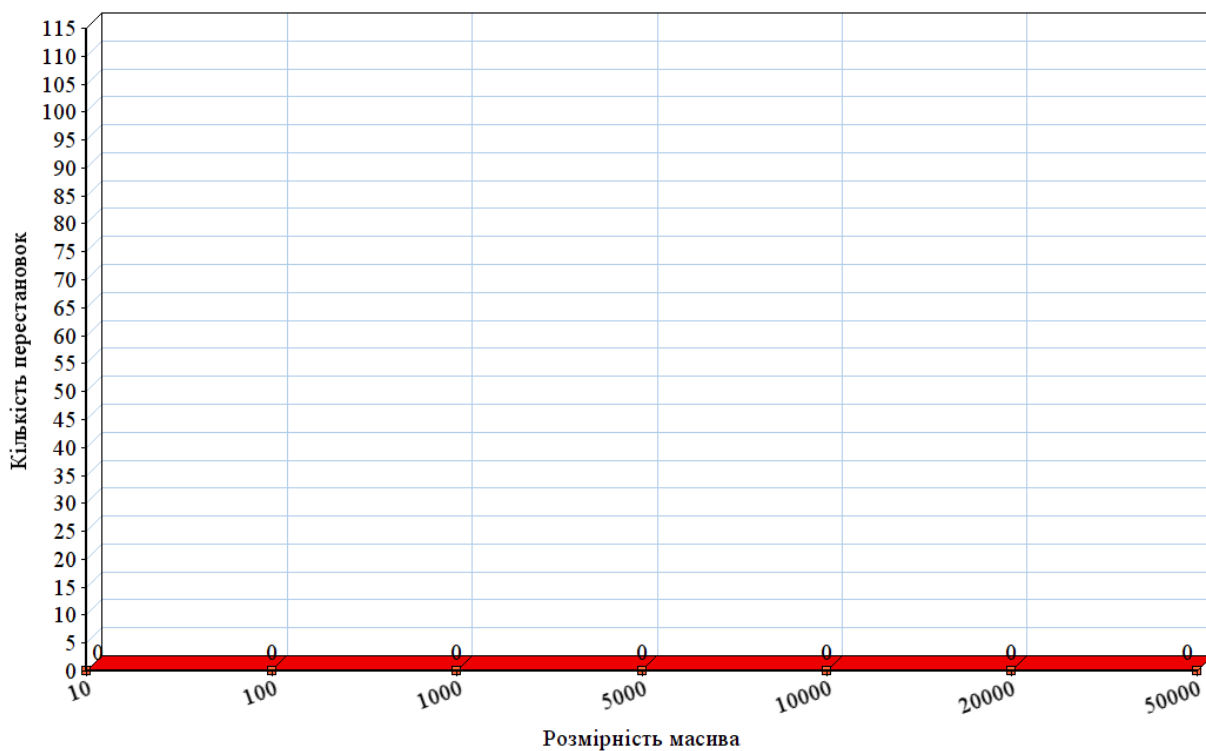
10	45	18
100	4 950	2 609
1000	499 500	253 384
5000	12 497 500	6 237 337
10000	49 995 000	25 054 030
20000	199 990 000	99 787 368
50000	1 249 975 000	627 109 693

1.5.2 Графіки залежності часових характеристик оцінювання від розмірності масиву

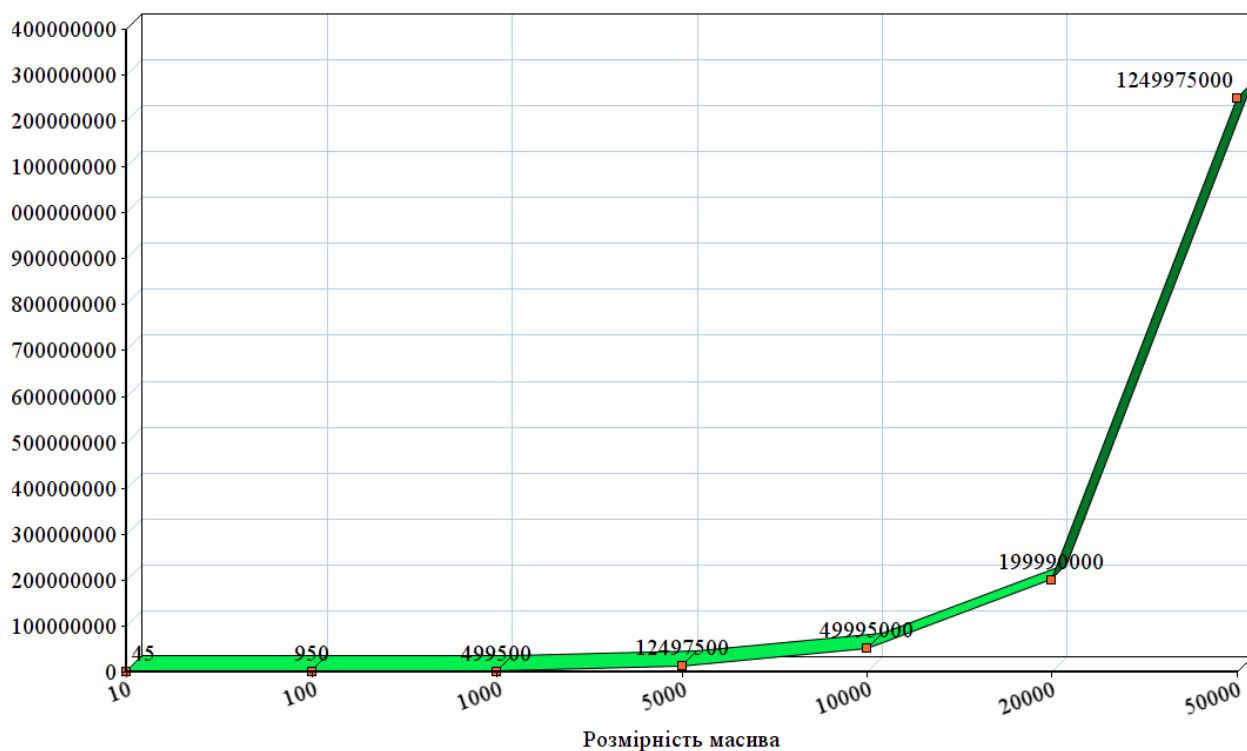
На рисунку 3.3 показані графіки залежності часових характеристик оцінювання від розмірності масиву для випадків, коли масиви містять упорядковану послідовність елементів (зелений графік), коли масиви містять зворотно упорядковану послідовність елементів (червоний графік), коли масиви містять випадкову 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

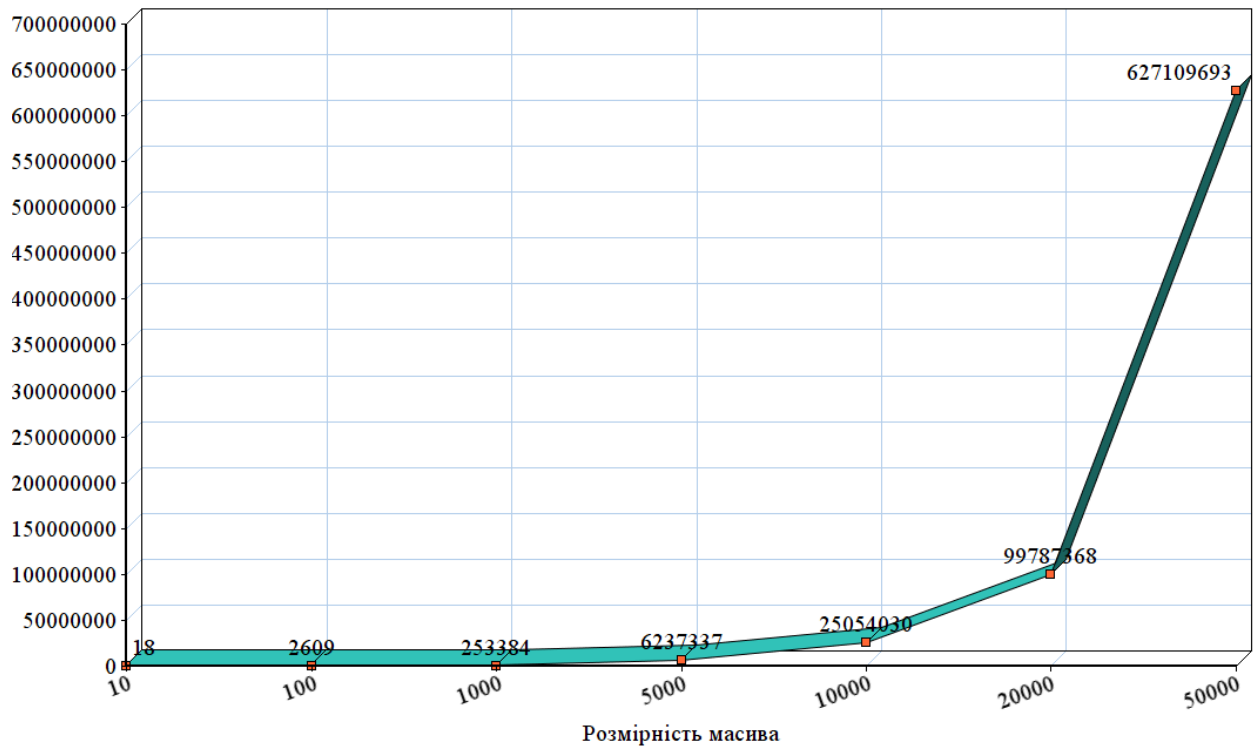
Графік залежності часових характеристик оцінювання від розмірності масива (упорядкований)



Графік залежності часових характеристик оцінювання від розмірності масива (зворотно упорядкований)

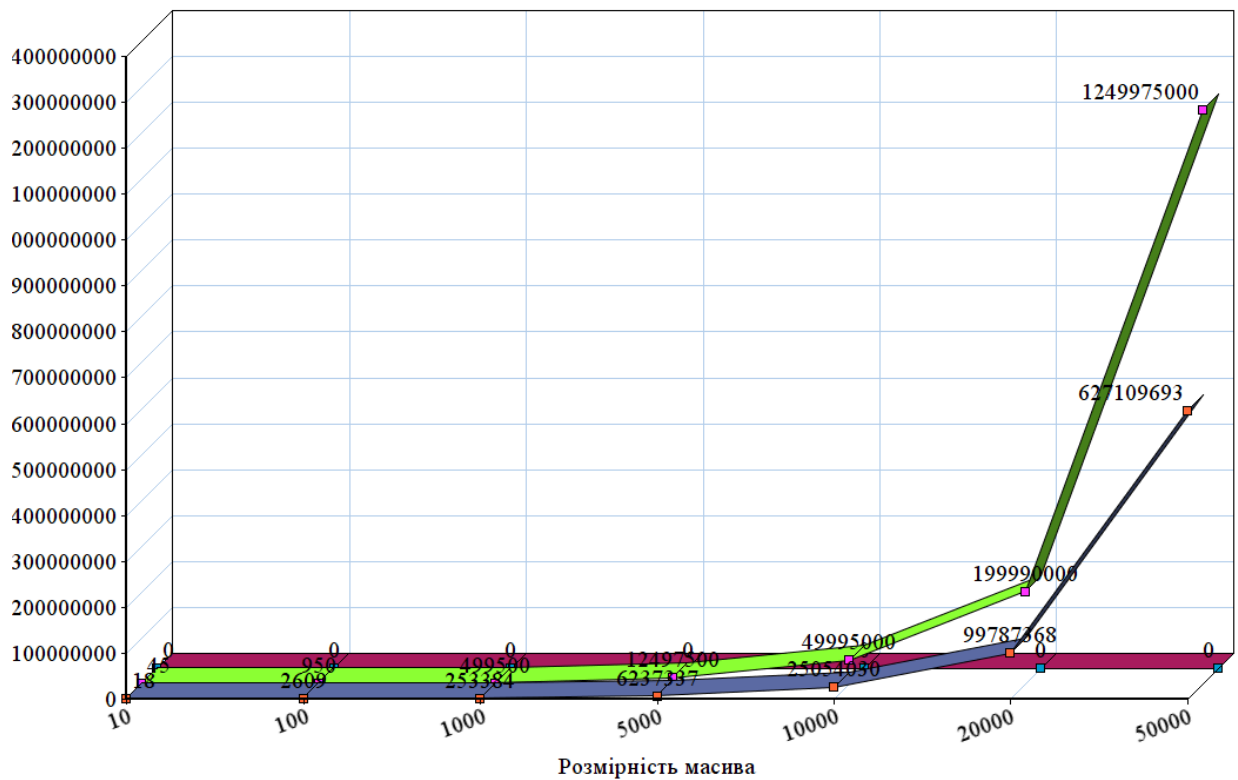


Графік залежності часових характеристик оцінювання від розмірності масива (випадково упорядкований)

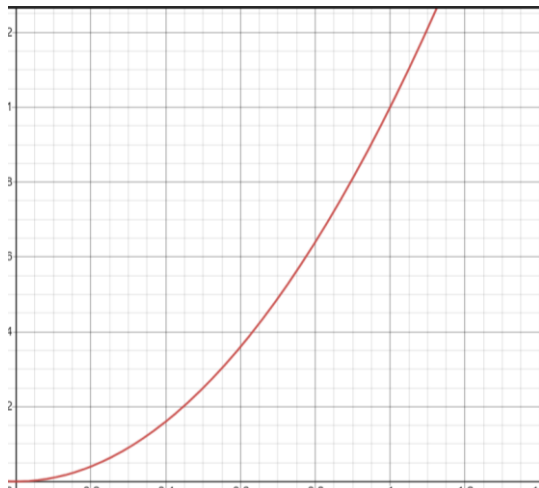


Графік залежності часових характеристик оцінювання від розмірності масива (випадково упорядкований)

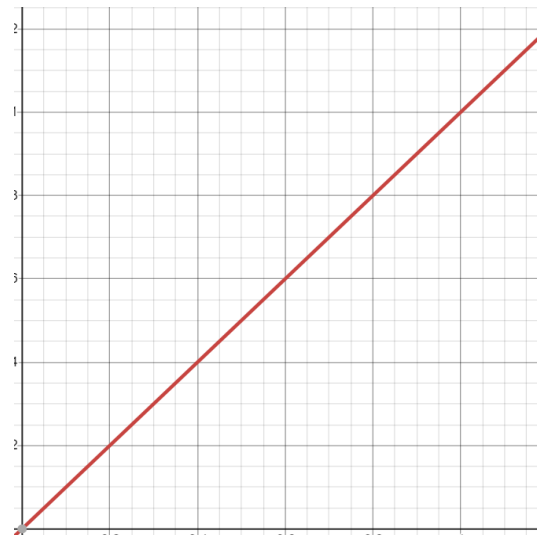
■ випадково упорядкований
 ■ зворотно упорядкований
 ■ упорядкований



Асимптотична оцінка в гіршому випадку



Асимптотична оцінка в кращому випадку



2.1 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 стійкий
«Природність» поведінки (Adaptability)	Алгоритм є природним
Базуються на порівняннях	Алгоритм базується на порівняннях
Необхідність в додатковій пам'яті (об'єм)	Не потребує додаткової пам'яті

Необхідність в знаннях про структури даних	Для використання алгоритму потрібно мати базові знання про структури даних
--	--

2.2 Псевдокод алгоритму

Підпрограма sort(arr, size)

Початок

factor:=1.247

step:=size/factor

повторити поки step>1

повторити для i від 0 до size-step

j:=i+step

якщо arr[j] > arr[j + 1] **то**

temp: = arr[i]

arr[i]: = arr[j]

arr[j]: = temp

все якщо

все повторити

step:=step/factor

якщо flag== 0 **то**

break

все якщо

все повторити

Кінець

2.3 Аналіз часової складності

Найгірший випадок алгоритму: $O(n^2)$

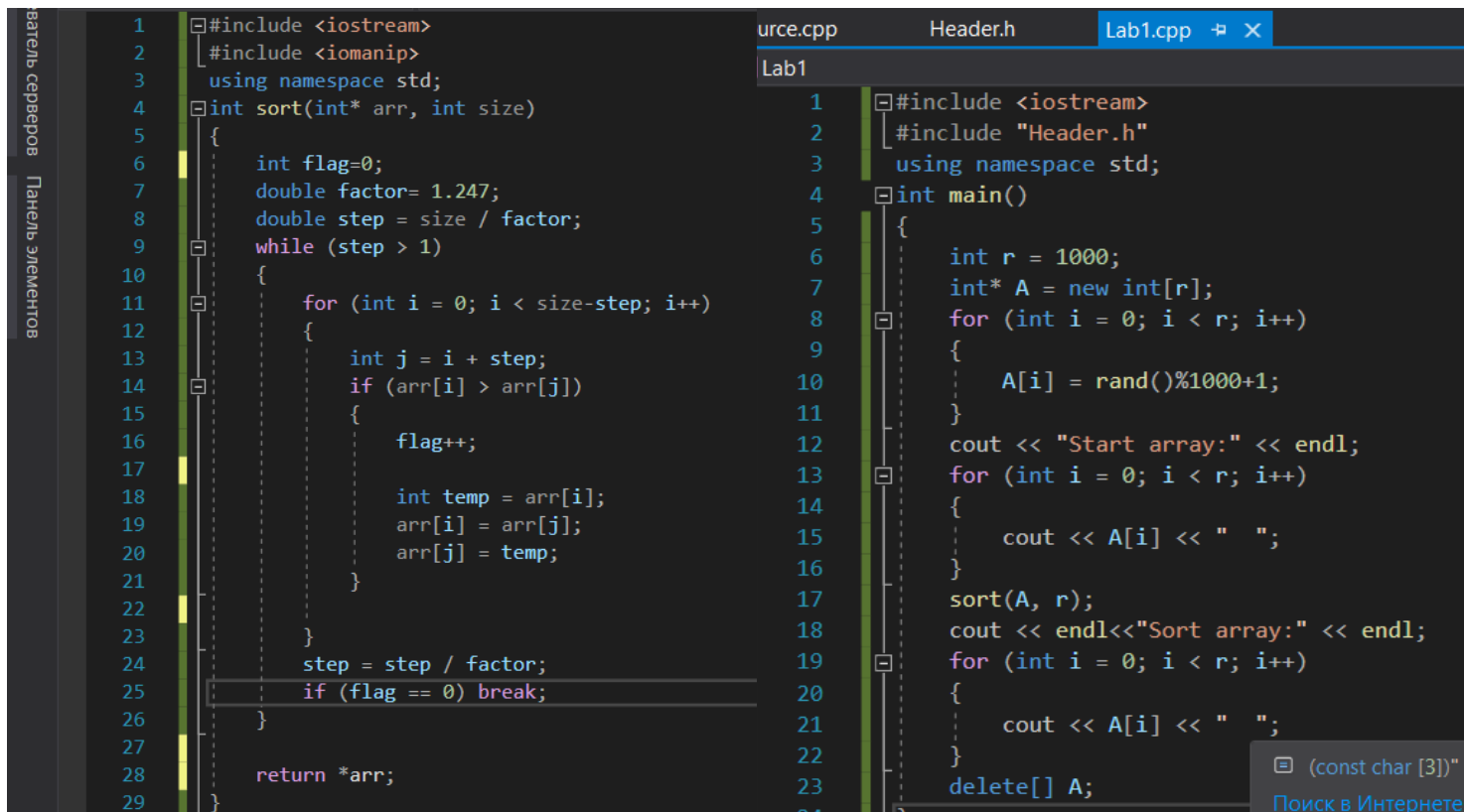
Найкращий випадок: $O(n)$

Середній випадок: $O(n * \log(n))$

2.4 Програмна реалізація алгоритму

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4  int sort(int* arr, int size)
5  {
6
7      double factor= 1.247;
8      double step = size / factor;
9      while (step > 1)
10     {
11         for (int i = 0; i < size-step; i++)
12         {
13             int j = i + step;
14             if (arr[i] > arr[j])
15             {
16
17                 int temp = arr[i];
18                 arr[i] = arr[j];
19                 arr[j] = temp;
20             }
21         }
22         step = step / factor;
23     }
24
25
26
27     return *arr;
28 }
```

2.5 Вихідний код

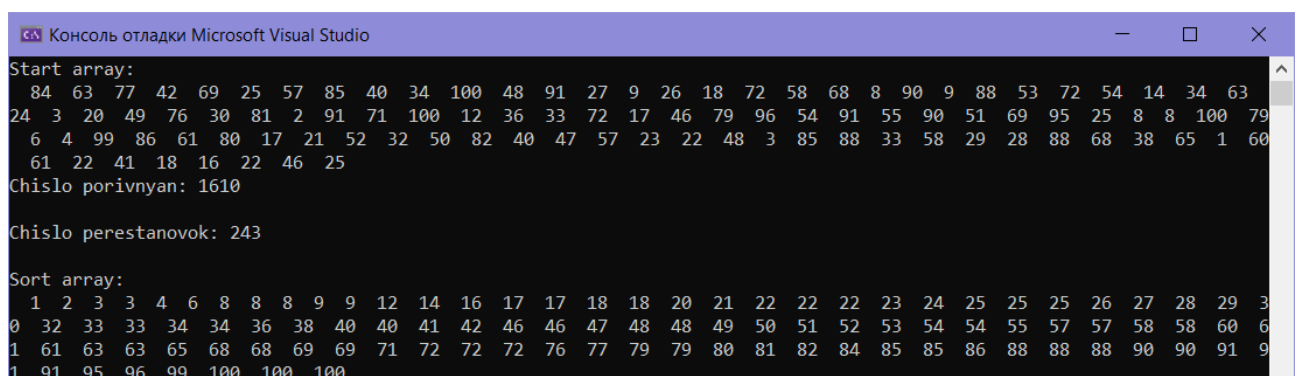


```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4 int sort(int* arr, int size)
5 {
6     int flag=0;
7     double factor= 1.247;
8     double step = size / factor;
9     while (step > 1)
10    {
11        for (int i = 0; i < size-step; i++)
12        {
13            int j = i + step;
14            if (arr[i] > arr[j])
15            {
16                flag++;
17
18                int temp = arr[i];
19                arr[i] = arr[j];
20                arr[j] = temp;
21            }
22        }
23        step = step / factor;
24        if (flag == 0) break;
25    }
26    return *arr;
27 }
28
29
source.cpp
Header.h
Lab1.cpp
Lab1
1 #include <iostream>
2 #include "Header.h"
3 using namespace std;
4 int main()
5 {
6     int r = 1000;
7     int* A = new int[r];
8     for (int i = 0; i < r; i++)
9     {
10        A[i] = rand()%1000+1;
11    }
12    cout << "Start array:" << endl;
13    for (int i = 0; i < r; i++)
14    {
15        cout << A[i] << " ";
16    }
17    sort(A, r);
18    cout << endl<<"Sort array:" << endl;
19    for (int i = 0; i < r; i++)
20    {
21        cout << A[i] << " ";
22    }
23    delete[] A;
```

2.6 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



```
Консоль отладки Microsoft Visual Studio
Start array:
84 63 77 42 69 25 57 85 40 34 100 48 91 27 9 26 18 72 58 68 8 90 9 88 53 72 54 14 34 63
24 3 20 49 76 30 81 2 91 71 100 12 36 33 72 17 46 79 96 54 91 55 90 51 69 95 25 8 8 100 79
6 4 99 86 61 80 17 21 52 32 50 82 40 47 57 23 22 48 3 85 88 33 58 29 28 88 68 38 65 1 60
61 22 41 18 16 22 46 25
Chislo porivnyan: 1610
Chislo perestanovok: 243
Sort array:
1 2 3 3 4 6 8 8 8 9 9 12 14 16 17 17 18 18 20 21 22 22 22 23 24 25 25 25 26 27 28 29 3
0 32 33 33 34 34 36 38 40 40 41 42 46 46 47 48 48 49 50 51 52 53 54 54 55 57 57 58 58 60 6
1 61 63 63 65 68 68 69 69 71 72 72 72 76 77 79 79 80 81 82 84 85 85 86 88 88 88 90 90 91 9
1 91 95 96 99 100 100 100
```

Рисунок 3.2 – Сортування масиву на 1000 елементів

Выбрать Консоль отладки Microsoft Visual Studio

Start array:
999 762 264 657 103 245 503 625 661 77 206 355 597 73 732 559 534 921 14 732 167 724 280 199 577 355 694 265 510 590 211 4 264 373 713 222 189 258 26 231 401 758 416
728 916 264 88 340 118 117 148 604 400 100 216 53 717 402 869 316 210 318 773 254 89 205 702 785 884 336 219 167 448 164 967 52 223 971 923 685 475 689 225 515 592 130
538 881 945 394 670 490 630 301 74 79 133 640 895 400 214 671 232 632 323 465 11 830 967 113 900 257 878 550 775 843 270 421 917 394 13 189 728 227 803 723 914 503 513
503 915 531 978 519 722 155 937 811 5 247 835 673 725 200 758 290 444 372 42 876 224 632 527 73 189 142 144 904 133 112 511 83 667 203 288 737 544 510 666 804 718 734
809 773 532 483 243 710 465 293 525 537 372 124 865 102 387 724 77 461 466 440 102 706 247 400 569 11 364 997 278 259 516 372 472 60 498 50 158 859 190 439 102 507 473
190 490 193 243 636 107 376 227 119 654 702 799 801 556 862 276 421 774 208 544 721 945 550 766 132 718 952 395 514 485 722 375 405 99 412 565 183 745 698 694 502 768 76
6 122 47 225 643 417 646 546 363 786 252 790 859 986 978 835 684 671 165 522 950 965 370 315 473 513 772 352 388 348 964 434 345 906 337 705 304 123 16 912 956 346 149
487 667 540 395 118 729 963 680 622 559 332 477 922 144 366 292 598 685 709 210 111 569 885 512 131 863 2 973 358 363 911 934 829 486 550 912 334 670 640 220 728 981 837
373 401 713 526 967 165 908 800 664 876 781 756 819 415 703 398 370 191 480 362 532 223 660 638 616 947 611 652 353 25 784 38 730 158 148 511 428 348 123 79 541 43 541
495 515 318 503 390 698 151 115 859 818 782 901 932 620 814 336 303 54 312 65 535 14 531 633 426 483 684 305 40 82 83 625 502 96 143 81 563 395 571 60 986 766 1000 98
762 524 253 845 787 398 286 898 710 16 568 566 814 324 29 916 550 397 489 760 297 738 780 469 461 946 91 911 528 919 521 495 255 283 586 414 571 936 976 429 28 34
802 509 506 660 937 907 466 312 109 142 862 264 80 164 385 933 129 689 361 587 715 466 462 541 991 617 305 826 32 706 272 87 165 253 58 701 804 253 910 84 605 872 545
423 525 208 215 346 380 498 465 898 841 980 352 901 940 180 985 114 412 503 466 73 170 152 68 229 166 625 449 433 617 62 967 770 723 896 254 3 733 632 679 294 769 149
390 394 529 7 553 784 142 101 193 811 314 676 848 575 231 736 443 704 697 176 931 94 186 261 831 40 989 972 716 759 973 517 831 980 70 890 666 741 148 819 293 668 308
649 583 144 735 498 510 704 382 798 875 37 418 839 297 127 3 286 231 123 988 131 420 780 556 436 779 360 120 643 863 12 543 553 97 122 247 631 823 169 422 863 879 782 5
95 134 29 562 774 461 985 692 949 312 692 145 729 447 597 865 113 363 327 604 165 915 85 540 701 519 49 996 787 964 664 57 653 922 194 337 520 274 913 922 290 349 744 8
19 783 769 159 256 551 740 60 666 229 454 723 438 866 586 641 544 891 277 126 903 201 714 449 834 440 318 824 971 114 264 624 707 588 138 682 614 617 270 767 49 310 707
757 937 807 991 314 333 832 30 32 439 605 224 242 439 182 839 470 153 517 585 475 944 770 948 485 563 169 417 407 803 130 866 105 700 490 631 802 44 202 842 95 793 603
738 812 159 609 205 938 19 731 346 394 58 168 174 369 269 590 452 965 483 696 706 863 471 413 138 401 820 133 47 594 942 761 441 119 467 35 138 29 202 588 777 572 506
765 375 702 982 864 849 680 711 392 125 464 235 670 416 896 965 534 135 290 637 249 93 453 458 755 708 646 108 448 919 154 975 603 304 795 493 575 814 353 843 890 905 8
19 127 959 801 95 712 923 115 46 174 297 885 355 996 514 877 870 764 807 477 380 9 865 761 601 370 372 828 272 670 774 183 410 755 806 436 829 988 475 883 367 698 971 2
83 396 33 502 723 652 431 993 728 441 616 951 970 711 798 501 803 516 376 845 983 426 808 573 666 491 881 584 687 701 128 702 485 630 522 128 447 353 455 399 619 857 269
245 970 126 432 275 226 422 609 653 913 165 53 451 107 192 175 857 894 367 696 986 196 209 806 784 600 508 259 15 132 405 295 347 459 717 81 143 787 653 714 895 469 33
1 34 464 343 605 583 350 761 557 947 151 320 238
Chislo porivnyan: 26969

Chislo perestanovok: 3775

Sort array:
2 3 3 4 7 9 11 11 12 13 14 14 15 16 16 19 19 25 26 28 29 29 29 32 32 33 34 35 37 38 40 40 42 43 44 46 47 47 49 49 50 52 53 53 54 54 57 58 58 60 60 60 62 65 68
70 73 73 73 74 77 77 79 79 80 81 81 82 83 83 84 85 87 88 89 91 93 94 95 95 96 97 99 100 101 102 102 103 105 107 107 108 109 109 111 112 113 113 114 114 115 115 117 118
118 119 119 120 122 122 123 123 123 124 125 126 126 127 127 128 128 129 130 130 131 131 132 132 133 133 133 134 135 138 138 138 142 142 142 143 143 144 144 144 145 148
148 148 149 149 151 151 152 153 154 155 158 158 159 159 164 164 165 165 165 166 167 167 168 169 169 170 174 174 175 176 180 182 182 183 183 186 189 189 190 190
191 192 192 193 193 194 196 199 201 202 205 205 206 207 208 208 208 209 210 210 211 214 215 216 219 220 222 223 223 224 224 224 225 225 226 227 227 229 229 231 231 231 2
32 235 238 242 243 243 245 245 247 247 249 252 253 253 253 254 254 255 256 257 258 259 259 261 264 264 264 264 264 265 269 269 270 270 272 272 274 275 276 277 278 280 28
3 283 283 286 286 288 290 290 290 292 293 293 294 295 297 297 297 301 303 304 304 305 305 308 308 310 312 312 312 314 314 315 316 318 318 318 320 323 324 327 330 331 332
3 333 334 336 336 337 337 340 340 343 345 346 346 346 348 348 349 350 352 352 353 353 353 355 355 360 361 362 363 363 363 364 366 367 367 369 370 370 372 372
372 372 373 373 375 375 376 376 380 380 382 385 387 390 390 392 394 394 394 395 395 395 396 397 397 398 399 400 400 400 401 401 401 401 402 405 405 407 410 412 412 413
414 415 416 416 417 417 418 420 421 421 422 422 423 426 426 428 429 431 432 433 434 436 436 438 439 439 439 440 440 441 441 443 444 447 447 448 448 449 449 451 452 453 4
54 455 458 459 461 461 461 462 464 464 465 465 465 466 466 466 466 467 469 469 470 471 472 473 473 475 475 475 477 477 480 483 483 483 485 485 485 486 487 489 490 490 49
0 491 493 495 495 498 498 498 501 502 502 502 503 503 503 503 506 506 507 508 509 510 510 510 511 511 511 512 513 513 514 514 514 515 516 516 517 517 519 519 519 520 521 522 522 524
525 525 526 527 528 529 531 531 532 532 534 534 535 535 540 540 540 541 541 541 543 544 544 544 544 545 546 550 550 550 550 550 551 553 553 556 556 557 559 559 562 563 563 565
566 568 569 569 571 571 572 573 575 575 577 583 583 583 584 585 586 586 587 588 588 590 592 594 595 597 597 598 600 601 603 603 604 604 605 605 605 609 609 611 614
616 616 617 617 617 619 620 622 624 625 625 625 630 630 631 631 632 632 632 633 636 637 638 640 640 641 643 643 646 646 649 652 652 653 653 653 654 657 658 660 660 661 6
64 664 666 666 666 667 667 668 670 670 670 671 671 673 676 679 680 680 682 684 684 685 685 687 689 689 692 692 694 694 696 696 696 696 699 698 698 700 701 701 701 70
2 702 702 702 703 704 704 705 706 706 706 707 707 708 708 709 710 711 711 712 713 713 714 714 715 716 717 717 718 718 719 721 722 722 723 723 723 724 724 725 728 728
728 728 729 729 730 731 732 732 733 734 735 736 737 738 738 740 741 744 745 755 755 756 757 758 758 759 760 761 761 761 762 762 764 765 766 766 767 768 769 769 770
770 772 773 773 774 774 774 775 777 779 780 780 781 782 782 783 784 784 784 785 786 787 787 787 790 793 795 798 798 799 800 801 801 802 802 803 803 804 804 806 806 807
807 808 811 812 814 814 818 819 819 819 820 823 824 826 828 829 829 830 830 831 831 832 834 835 835 837 839 839 841 842 843 843 845 845 848 849 857 857 859 8
59 859 862 862 863 863 863 864 865 865 865 866 866 869 870 872 875 876 876 877 878 879 881 881 883 884 885 885 890 890 891 894 895 895 896 896 898 898 899 900 901 90
1 903 904 905 906 907 908 910 911 911 912 912 913 913 914 915 915 915 916 916 917 919 919 921 922 922 922 923 923 931 932 933 934 936 937 937 937 938 940 942 944 945 945

2.7 Тестування алгоритму

Додатково протестуємо алгоритм сортування гребінцем на випадково згенерованому масиві з розмірністю 20 елементів (елементи можуть повторюватись).

Код:

Результат роботи:

Source.cpp Header.h Lab1.cpp ▾ ×

Lab1 (Глобальная область) ▾ main()

1 #include <iostream>
2 #include "Header.h"
3 using namespace std;
4 int main()
5 {
6 srand(time(NULL));
7 int r = 20;
8 int* A = new int[r];
9 for (int i = 0; i < r; i++)
10 {
11 A[i] = rand()%20+1;
12 }
13 cout << "Start array:" << endl;
14 for (int i = 0; i < r; i++)
15 {
16 cout<<" "<<A[i];
17 }
18 cout<<endl<<"Chislo perestanovok: "<<sort(A, r)<<endl;
19 cout << endl<<"Sort array:" << endl;
20 for (int i = 0; i < r; i++)
21 {
22 cout << " " << A[i];
23 }
24 delete[] A;
25 }

Консоль отладки Microsoft Visual Studio

Start array:
9 19 11 6 1 9 20 18 17 6 12 13 2 11 2 12 17 6 12 7
Chislo porivnyan: 189

Chislo perestanovok: 19

Sort array:
1 2 2 6 6 6 7 9 9 11 11 12 12 12 13 17 17 18 19 20

Як бачимо, алгоритм коректно відсортував випадково згенерований масив. Отже, наш алгоритм сортування гребінцем працює правильно.

2.8 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	2	0
100	20	0
1000	199	0
5000	991	0
10000	1981	0
20000	3962	0
50000	9904	0

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

Розмірність масиву	Число порівнянь	Число перестановок
10	67	7
100	1610	108
1000	26969	1474
5000	169778	8732
10000	369539	18812
20000	799052	40786
50000	2247599	108878

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	67	10
100	1610	218
1000	26969	3943
5000	169778	25311
10000	369539	56774
20000	799052	124309
50000	2247599	340961

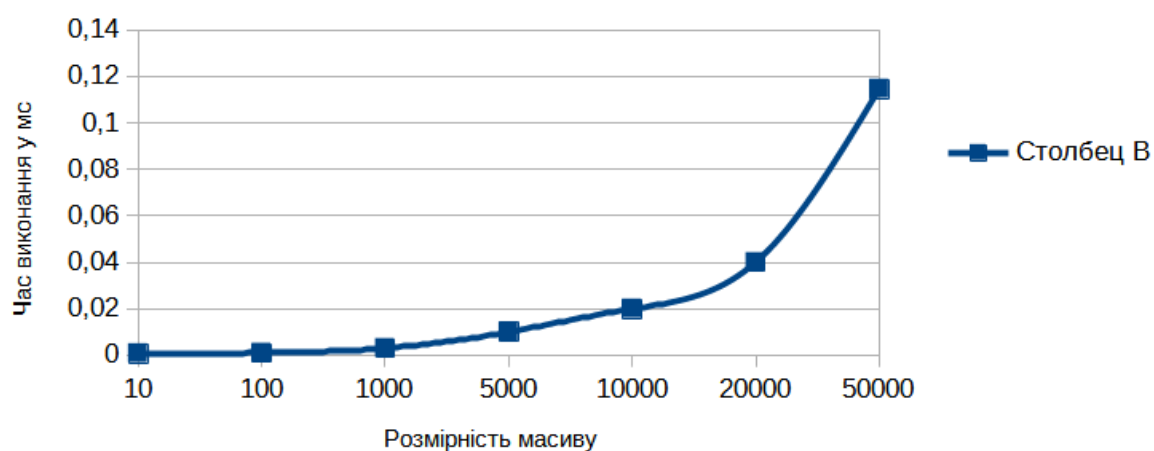
1.5.3 Графіки залежності часових характеристик оцінювання від розмірності масиву

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

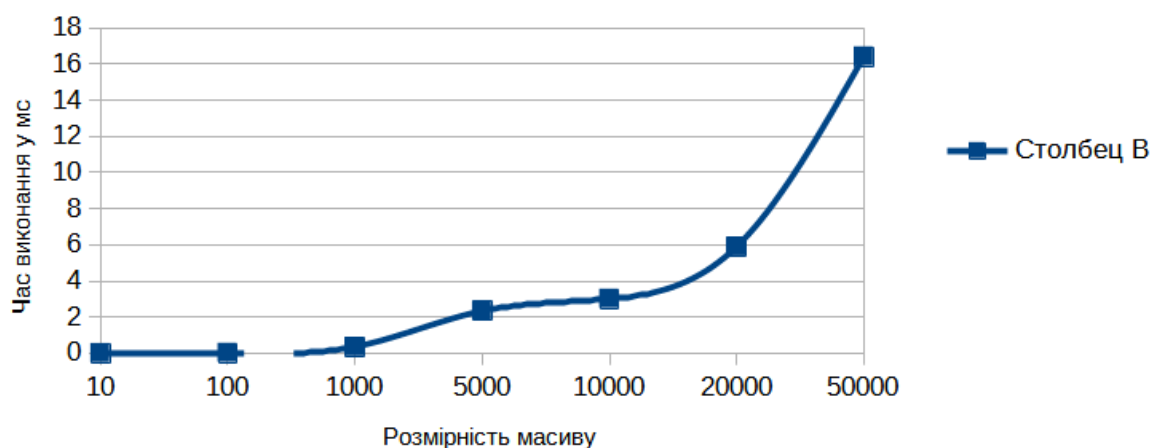
Графік залежності часових характеристик оцінювання від
розмірності масиву

(упорядкований масив)



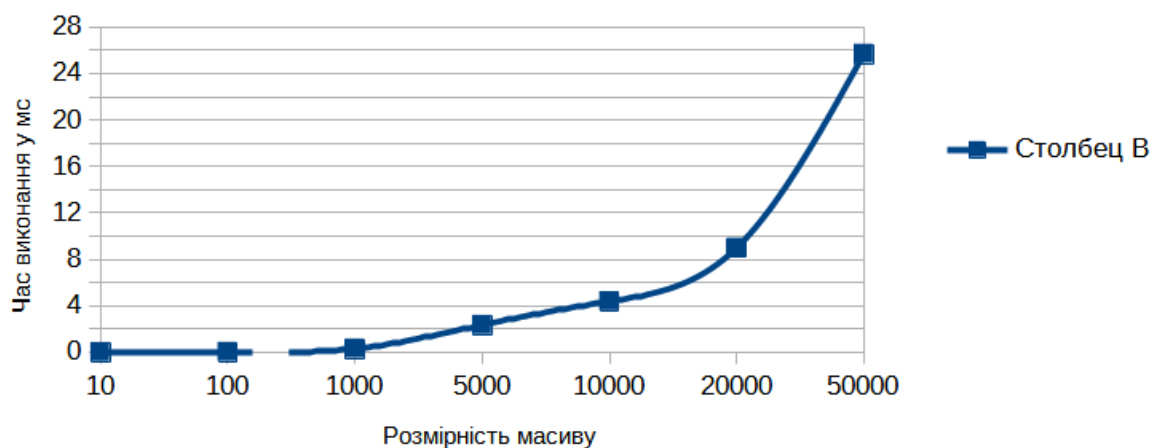
Графік залежності часових характеристик оцінювання від
розмірності масиву

(зворотньо упорядкований масив)

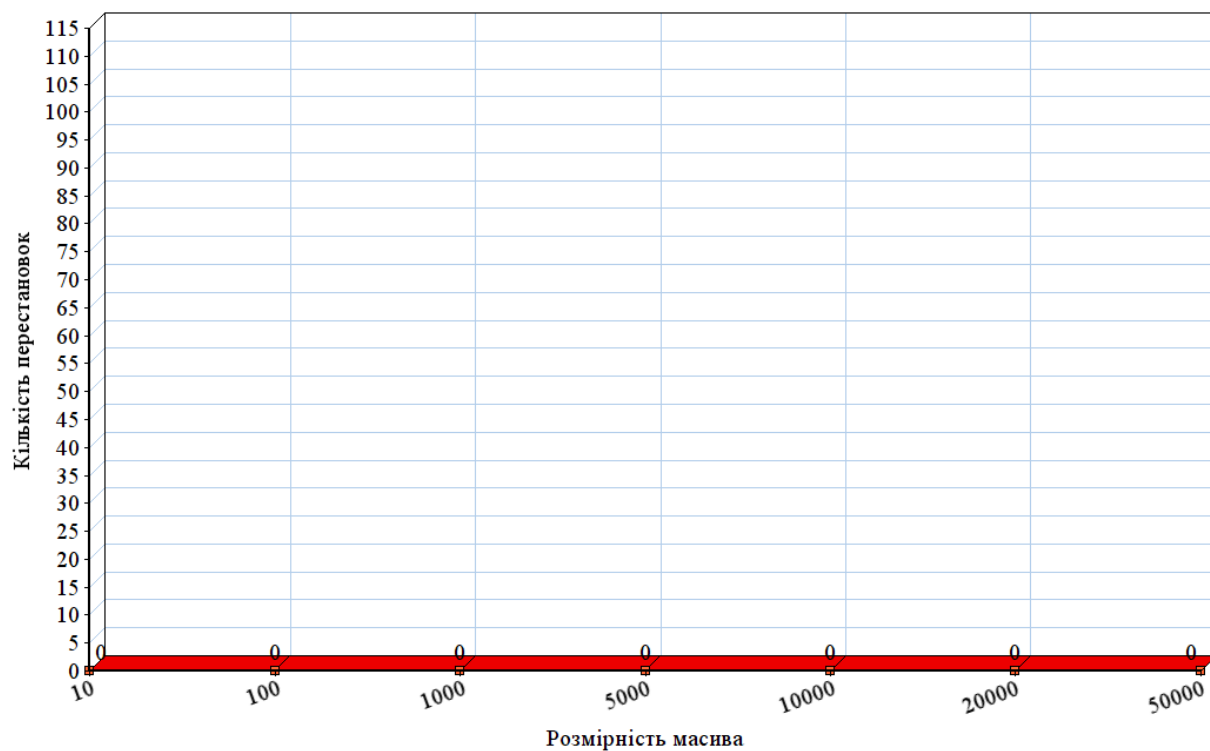


Графік залежності часових характеристик оцінювання від
розмірності масиву

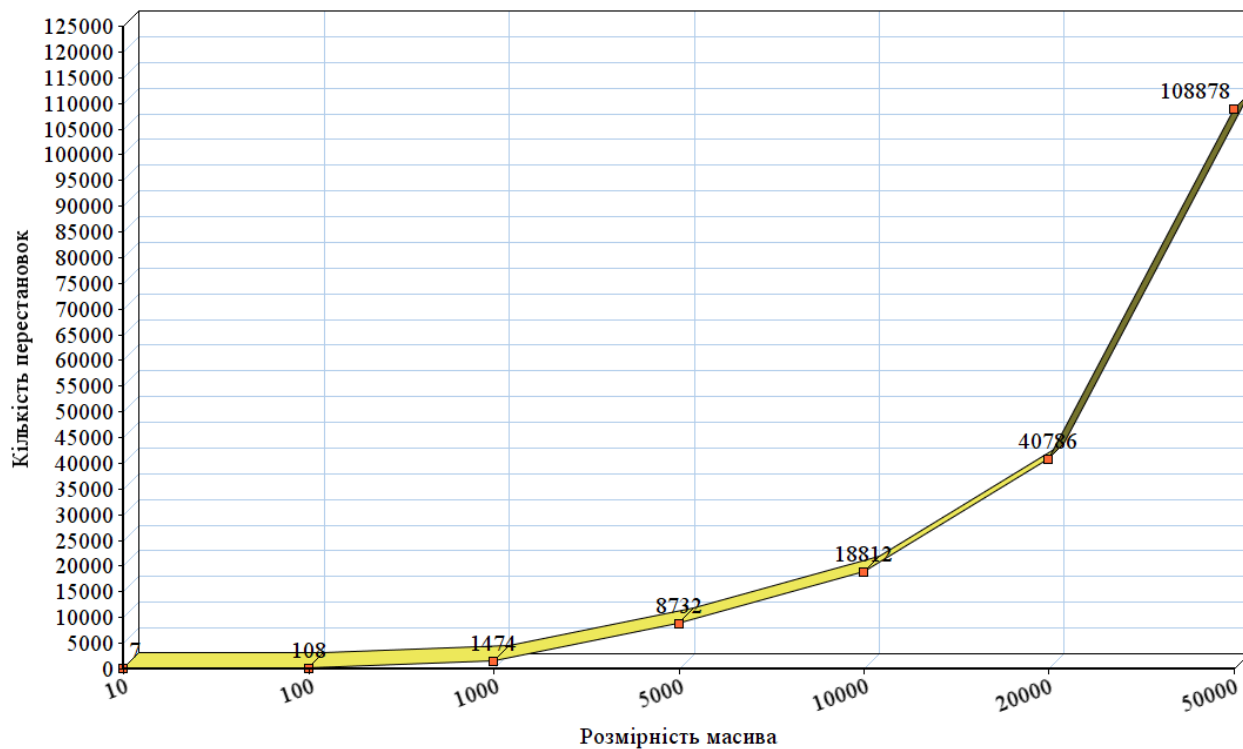
(зворотньо упорядкований масив)



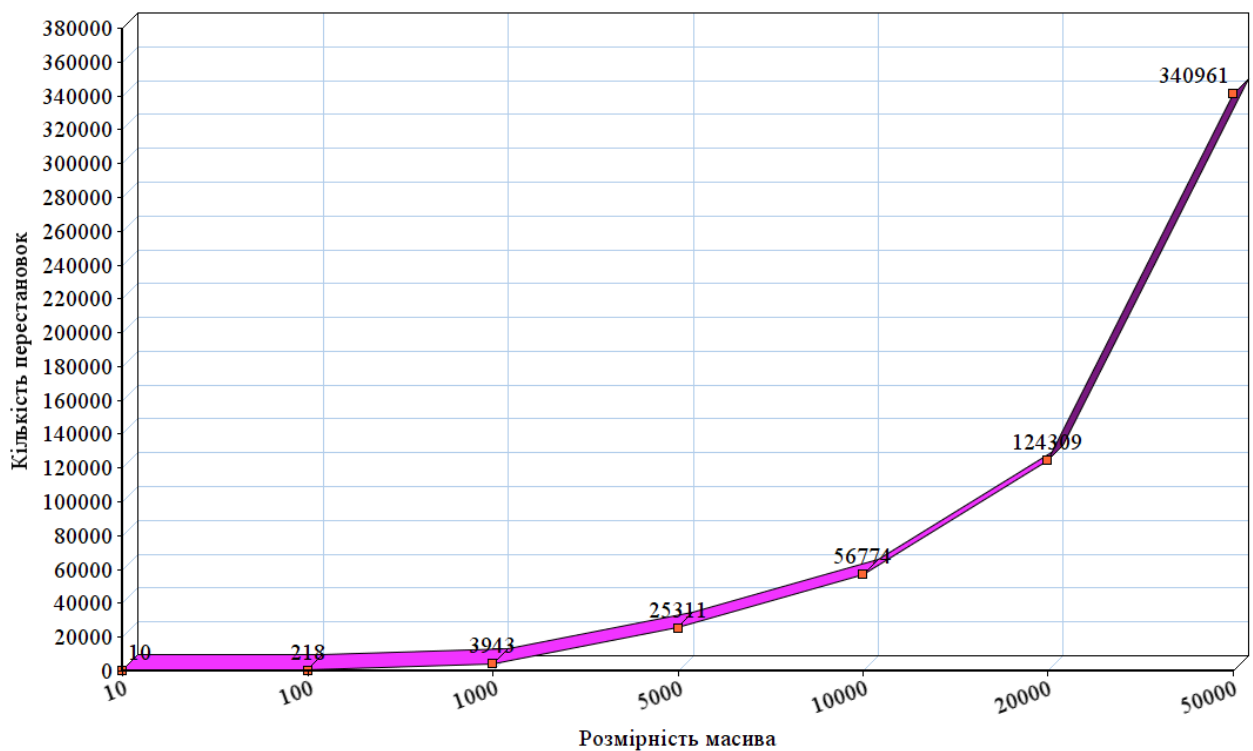
Графік залежності часових характеристик оцінювання від розмірності масива (упорядкований)



Графік залежності часових характеристик оцінювання від розмірності масива (зворотньо упорядкований)

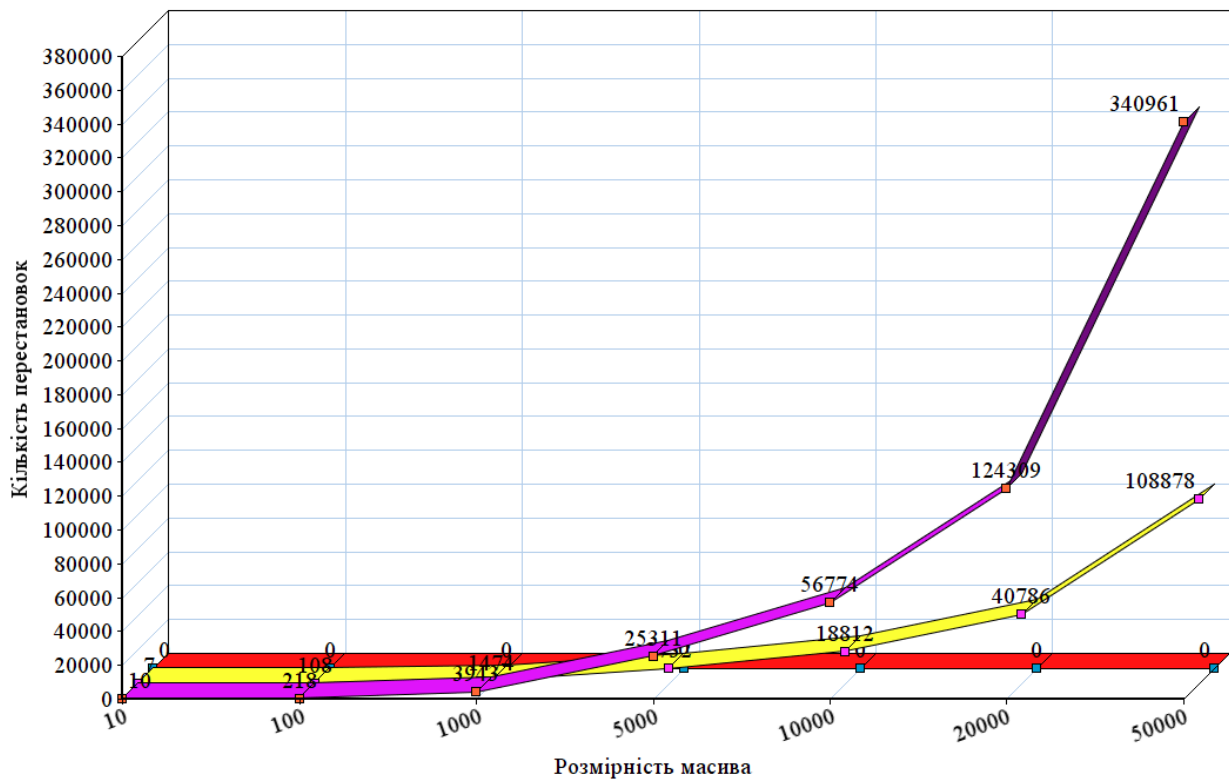


Графік залежності часових характеристик оцінювання від розмірності масива (випадково упорядкований)

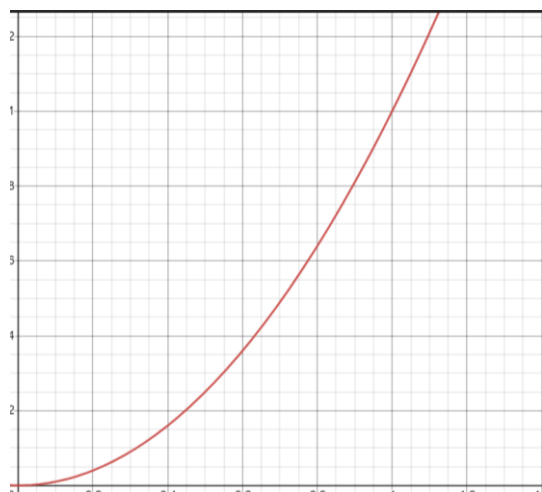


Графіки залежності часових характеристик оцінювання від розмірності масива

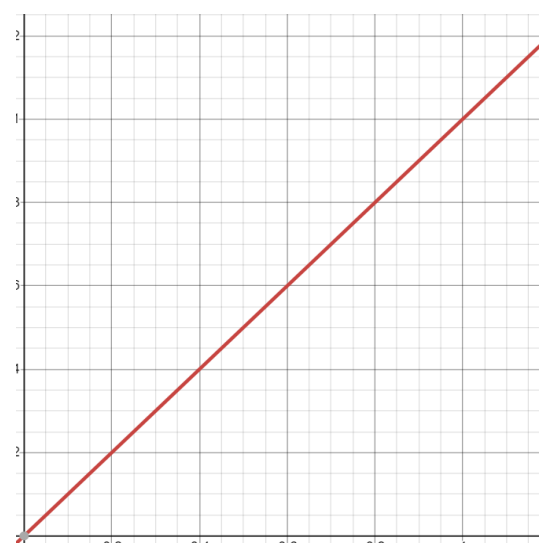
— випадково упорядкований — зворотно упорядкований — упорядкований



Асимптотична оцінка в гіршому випадку



Асимптотична оцінка в кращому випадку



ВИСНОВОК

У ході лабораторної роботи ми вивчили основні методи аналізу обчислювальної складності алгоритмів внутрішнього сортування, застосували ці знання на практичному завданні і оцінили поріг ефективності кожного методу. При виконанні порівняльного аналізу алгоритмів сортування бульбашкою і гребінцем, ми зробили висновок, що сортування гребінцем є більш швидким алгоритмом. Переконатися у цьому можна, подивившись на графіки залежності часових характеристик оцінювання від розмірності масиву.

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%.