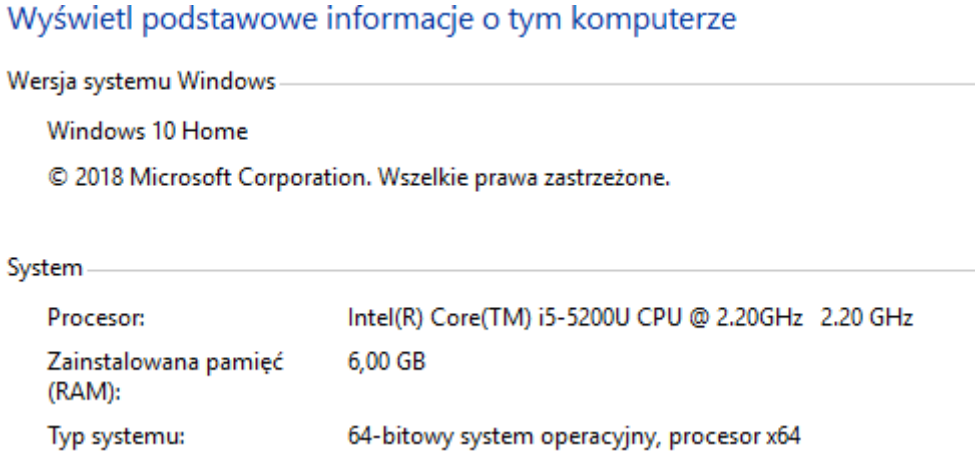
KOMPUTER UŻYWANY DO TESTÓW



WERSJA VISUAL STUDIO 2017

I.Porównaj szybkość działania 4 metod sortowania: Insertion Sort, Selection Sort, Heap Sort, Cocktail Sort dla tablicy liczb całkowitych (rzędu 50k - 200k elementów) generowanych w postaci: losowej, rosnącej, malejącej, stałej, v- kształtnej.

Użyty kod

```
static void Main(string[] args)
{
    Stopwatch stopwatch = new Stopwatch();
    Console.WriteLine("rozmiar\tlosowa\t\trosnaca\t\tmalejaca\tstala\t\tvksztaltna");
    for (int i = 50000; i <= 200000; i += 10000)
    {
        int[] tab = new int[i];
        Console.Write(i + "\t");
        losowa(tab);//metoda zwracająca tablicę losowa
        stopwatch.Start();//rozpoczęcie pomiaru czasu
        InsertionSort(tab);//zaimplementowanie tablicy losowej do metody sortującej
        stopwatch.Stop();//zakończenie pomiaru czasu
        Console.Write(stopwatch.Elapsed.TotalSeconds.ToString("F8") + "\t");
        stopwatch.Reset();//reset pomiaru
        rosnaca(tab);//metoda zwracająca tablicę rosnaca
        stopwatch.Start();//rozpoczęcie pomiaru czasu
        InsertionSort(tab);//zaimplementowanie tablicy rosnącej do metody sortującej
        stopwatch.Stop();//zakończenie pomiaru czasu
        Console.Write(stopwatch.Elapsed.TotalSeconds.ToString("F8") + "\t");
        stopwatch.Reset();//reset pomiaru
        malejaca(tab);//metoda zwracająca tablicę malejaca
        stopwatch.Start();//rozpoczęcie pomiaru czasu
        InsertionSort(tab);//zaimplementowanie tablicy malejącej do metody sortującej
        stopwatch.Stop();//zakończnie pomiaru czasu
        Console.Write(stopwatch.Elapsed.TotalSeconds.ToString("F8") + "\t");
        stopwatch.Reset();//reset pomiaru
        stala(tab);//metoda zwracająca tablicę stala
        stopwatch.Start();//rozpoczęcie pomiaru czasu
        InsertionSort(tab);//zaimplementowanie tablicy stałej do metody sortującej
        stopwatch.Stop();//zakończenie pomiaru czasu
        Console.Write(stopwatch.Elapsed.TotalSeconds.ToString("F8") + "\t");
        stopwatch.Reset();//reset pomiaru
        vksztaltna(tab);//metoda zwracająca tablicę vksztaltna
        stopwatch.Start();//rozpoczęcie pomiaru czasu
        InsertionSort(tab);//zaimplementowanie tablicy vksztaltnej do metody sortującej
        stopwatch.Stop();//zakończenie pomiaru czasu
        Console.Write(stopwatch.Elapsed.TotalSeconds.ToString("F8") + "\n");
        stopwatch.Reset();//reset pomiaru
```

W zależności od pomiaru, który miałem wykonać zmieniała się tylko nazwa metody.

Użyte metody sortowania:

1.InsertionSort

```
static public int[] InsertionSort(int[] tab)//metoda sortowania przez wstawianie
{
    for (uint i = 1; i < tab.Length; i++)
    {
        uint j = i;
        int temp = tab[j];

        while ((j > 0) && (tab[j - 1] > temp))
        {
            tab[j] = tab[j - 1];
            j--;
        }

        tab[j] = temp;
    }
    return tab;
}
```

2.SelectionSort

```
public static int[] SelectionSort(int[] tab)
{
    uint k;
    for (uint i = 0; i < (tab.Length - 1); i++)
    {
        int temp = tab[i];
        k = i;
        for (uint j = i + 1; j < tab.Length; j++)
            if (tab[j] < temp)
            {
                k = j;
                temp = tab[j];
            }

        tab[k] = tab[i];
        tab[i] = temp;
    }
    return tab;
}
```

3.CocktailSort

```
public static int[] CocktailSort(int[] tab)//funkcja sortowania koktajlowego
{
    int left = 1, right = tab.Length - 1, k = tab.Length - 1;
    do
    {
        for (int j = right; j >= left; j--)
            if (tab[j - 1] > tab[j])
            {
                int temp = tab[j - 1];
                tab[j - 1] = tab[j];
                tab[j] = temp;
                k = j;
            }

        left = k + 1;

        for (int j = left; j <= right; j++)
            if (tab[j - 1] > tab[j])
            {
                int temp = tab[j - 1];
                tab[j - 1] = tab[j];
                tab[j] = temp;
                k = j;
            }

        right = k - 1;
    } while
        (left <= right);
    return tab;
}
```

4.HeapSort

```
static int[] heapify(int[] t, uint left, uint right)
{
    uint i = left, j = 2 * i + 1;
    int buf = t[i];
    while (j <= right)
    {
        if (j < right)
            if (t[j] < t[j + 1])
                j++;
        if (buf >= t[j]) break;

        t[i] = t[j];
        i = j;
        j = 2 * i + 1;
    }
    t[i] = buf;
    return t;
}
public static void HeapSort(int[] tab)//metoda sortująca przez kopce
{
    uint left = (uint)tab.Length / 2;
    uint right = (uint)tab.Length - 1;
    while (left > 0)
    {
        left--;
        heapify(tab, left, right);
    }
    while (right > 0)
    {
        int buf = tab[left];
        tab[left] = tab[right];
        tab[right] = buf;
        right--;
        heapify(tab, left, right);
    }
}
```

Metody zwracające tablicę wypełnioną odpowiednimi wartościami:

1.Losowa

```
public static int[] losowa(int[] tab)//zwracanie tablicy wypełnionej losowo
{
    Random rand = new Random();
    for (int i = 0; i < tab.Length; i++)
    {
        tab[i] = rand.Next(50000, 200000);
    }
    return tab;
}
```

2.Rosnąca

```
public static int[] rosnaca(int[] tab)//zwracanie tablicy wypełnionej rosnąco
{
    for (int i = 0; i < tab.Length; i++)
    {
        tab[i] = i + 1;
    }
    return tab;
}
```

3.Malejąca

```
public static int[] malejaca(int[] tab)//zwracanie tablicy wypełnionej malejąco
{
    int x = tab.Length;
    for (int i = 0; i < tab.Length; i++)
    {
        tab[i] = x;
        x--;
    }
    return tab;
}
```

4.Stała

```
public static int[] stala(int[] tab)//zwracanie tablicy wypełnionej stałymi wartościami
{
    for (int i = 0; i < tab.Length; i++)
    {
        tab[i] = tab.Length;
    }
    return tab;
}
```

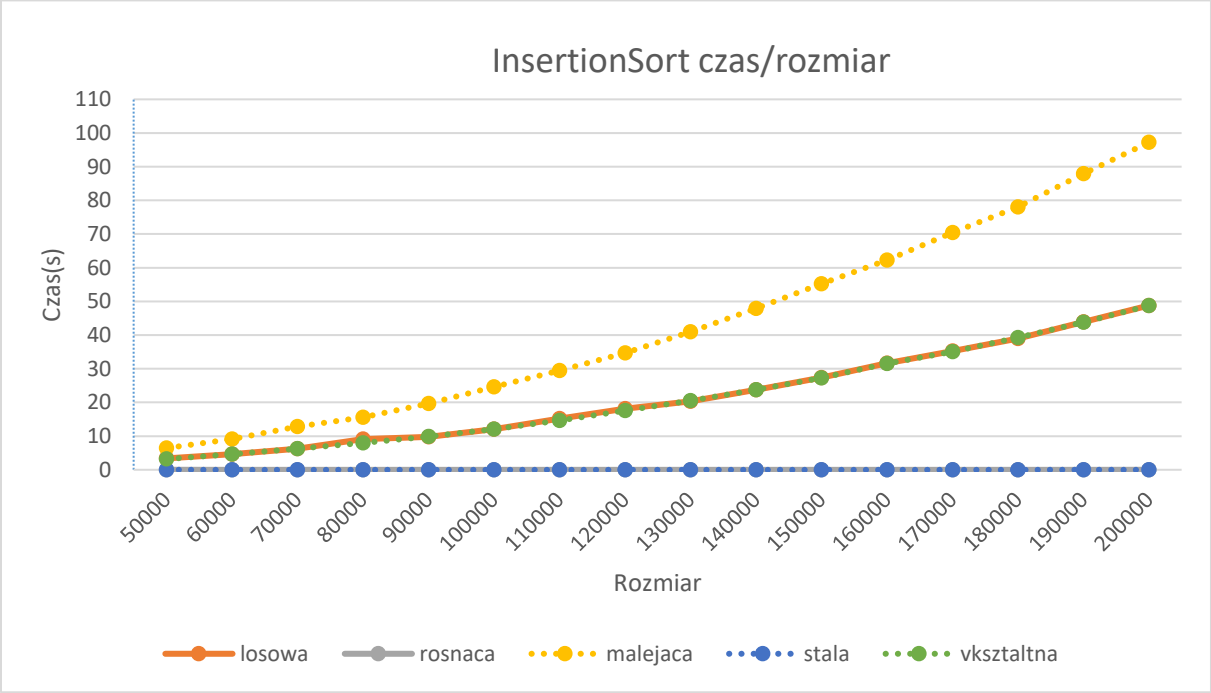
5.V-kształtna

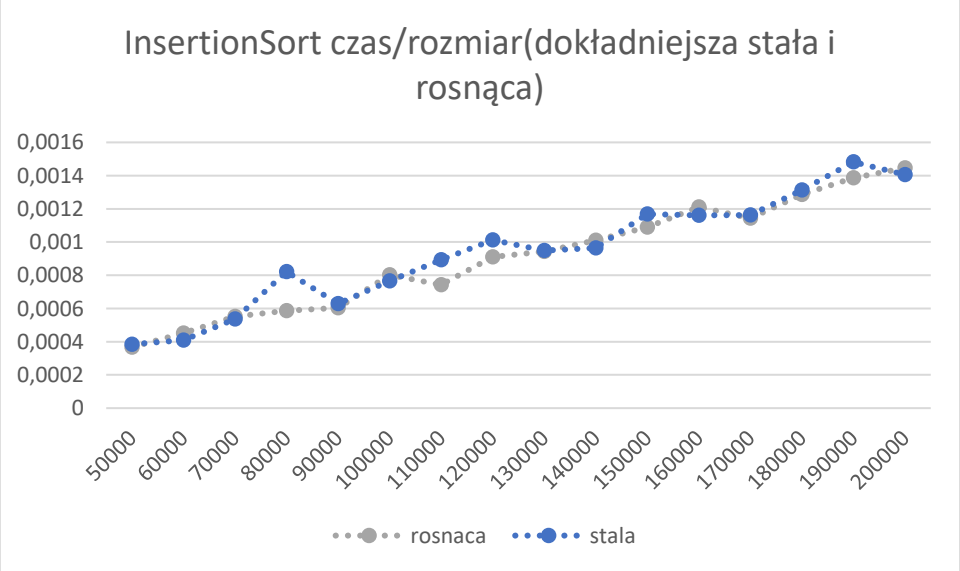
```
public static int[] vksztaltna(int[] tab)//zwracanie tablicy wypełnionej v-ksztaltnie
{
    int x = tab.Length;
    int y = 2;
    for (int i = 0; i < tab.Length; i++, x -= 2)
    {
        if (i == tab.Length / 2)
        {
            tab[i] = 1;
        }
        else if (i < tab.Length / 2)
        {
            tab[i] = x;
        }
        else
        {
            tab[i] = y;
            y += 2;
        }
    }
    return tab;
}
```

Insertion Sort

rozmiar	losowa	rosnaca	malejaca	stala	vksztaltna
50000	3,37828	0,000367	6,445355	0,000385	3,182591
60000	4,644208	0,000451	9,134337	0,00041	4,642068
70000	6,282491	0,000551	12,8122	0,000537	6,246745
80000	9,118548	0,000587	15,53327	0,000821	7,995688
90000	9,803106	0,000603	19,61814	0,000628	9,843123
100000	12,09686	0,000803	24,59147	0,000765	12,08662
110000	15,15885	0,000743	29,43585	0,000893	14,64825
120000	18,13819	0,00091	34,68765	0,001011	17,56357
130000	20,34681	0,000944	40,92292	0,00095	20,52604
140000	23,71873	0,00101	47,92417	0,000964	23,73064
150000	27,39687	0,00109	55,24488	0,001169	27,30439

160000	31,67395	0,00121	62,22785	0,001161	31,50588
170000	35,29276	0,001144	70,39745	0,001163	35,02633
180000	38,93988	0,001286	78,00324	0,001312	39,28196
190000	43,89687	0,001387	87,90522	0,001482	43,83687
200000	48,78219	0,001446	97,21661	0,001405	48,72306

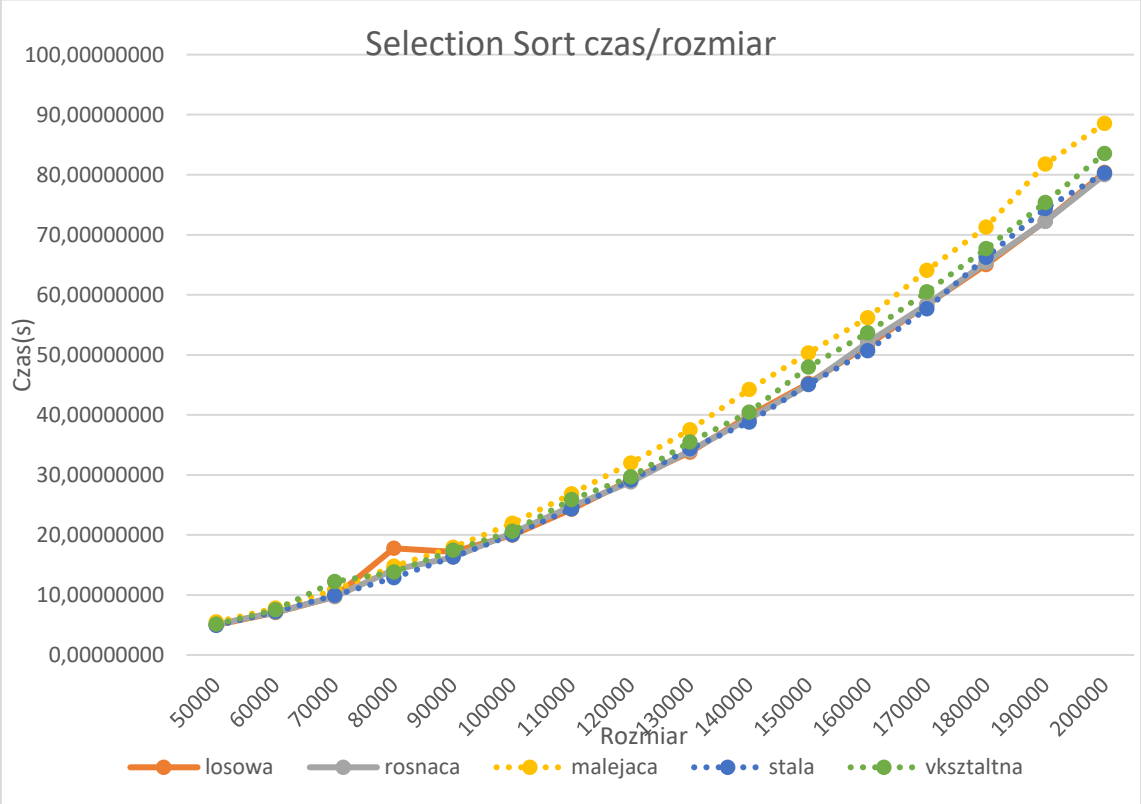




Selection Sort

rozmiar	losowa	rosnaca	malejaca	stala	vksztaltna
50000	4,99336040	5,10718480	5,48045050	4,89498030	5,11809030
60000	7,05467170	7,07719920	7,78329550	7,22965020	7,51313170
70000	9,73893410	9,73722200	10,83249210	9,91833940	12,22505700
80000	17,77214620	14,17970830	14,79681550	12,84794890	13,80703310
90000	17,19271090	16,28060920	17,90730710	16,33355350	17,42230520

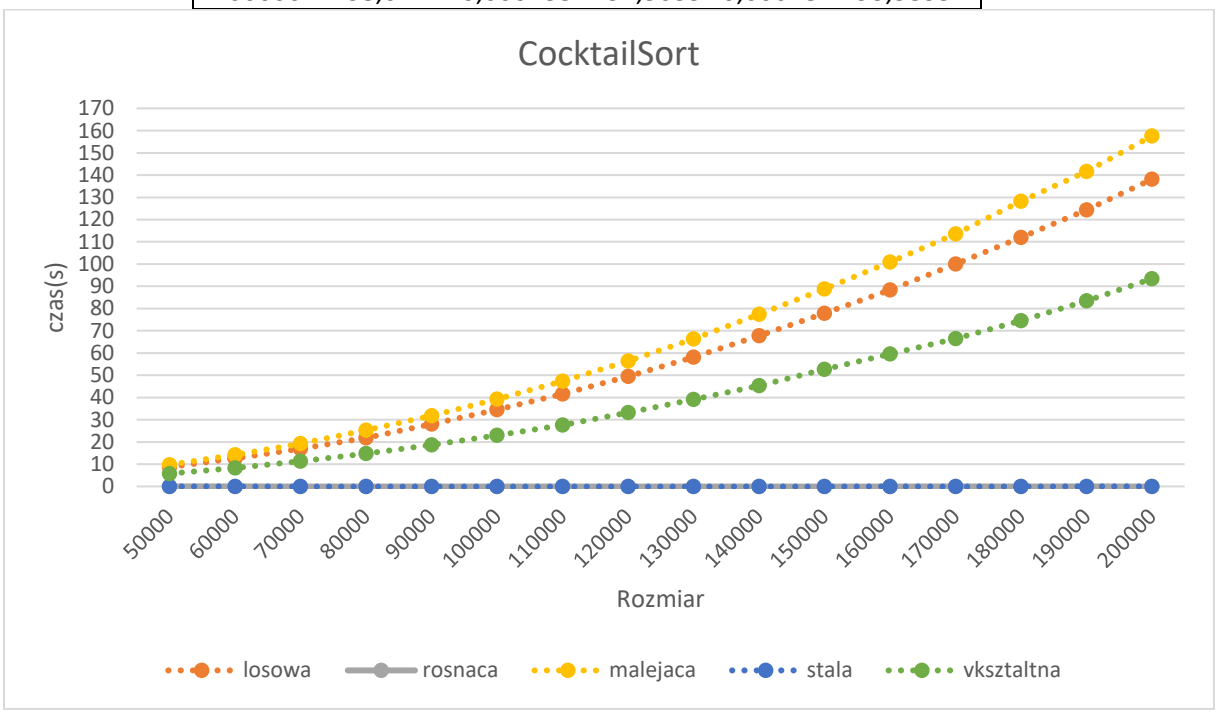
100000	19,94608060	20,32235940	21,94922180	19,98329950	20,56849700
110000	24,27714450	24,73485260	26,83720600	24,33181530	25,85322920
120000	29,17480550	28,78769620	31,94844900	29,10708690	29,66250020
130000	33,78835490	34,03475400	37,49991790	34,36746340	35,44990610
140000	39,82707660	39,38263160	44,23926830	38,80540910	40,44877100
150000	45,25685670	44,99409270	50,27941220	45,09662370	47,94556220
160000	51,58401210	52,07130220	56,18531160	50,69653830	53,66831380
170000	58,36383970	58,36434170	64,05623830	57,63673400	60,48559710
180000	64,99692770	65,36616190	71,25281020	66,25600040	67,65978220
190000	72,28263640	72,20847630	81,76733110	74,37271240	75,36054410
200000	80,35679130	79,99772570	88,53216330	80,27001740	83,53505330

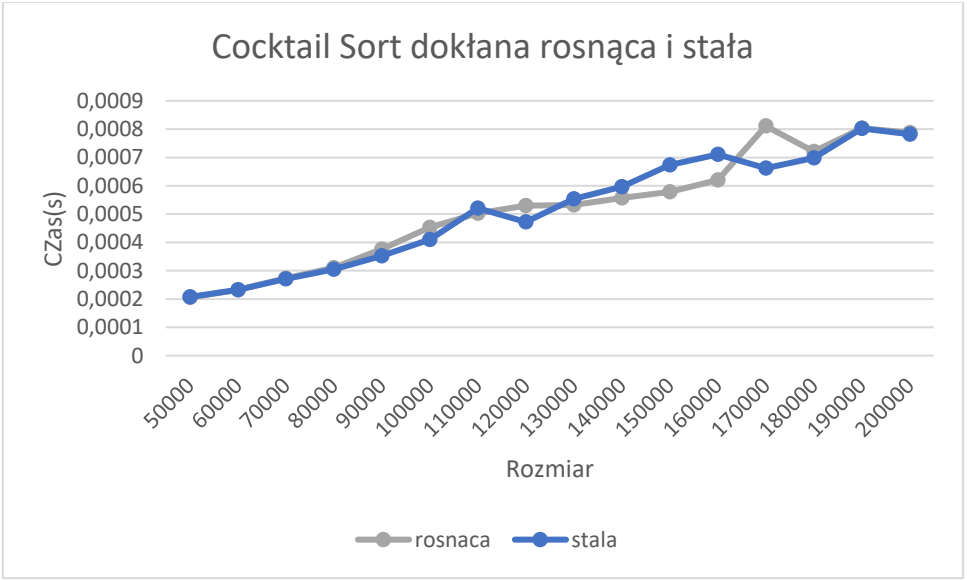


Cocktail Sort

rozmiar	losowa	rosnaca	malejaca	stala	vksztaltna
50000	8,681554	0,000205	9,777496	0,000207	5,758571
60000	12,55443	0,000233	14,13243	0,000233	8,241791
70000	16,98603	0,000273	19,16336	0,000271	11,23984
80000	21,80834	0,00031	25,33498	0,000305	14,74526
90000	28,04835	0,000376	31,70568	0,000352	18,63396
100000	34,43645	0,000453	39,21574	0,00041	22,90709
110000	41,54766	0,000502	47,29774	0,000521	27,63541
120000	49,51968	0,00053	56,34882	0,000472	33,16107

130000	58,04906	0,000532	66,3465	0,000553	39,1535
140000	67,69467	0,000557	77,32234	0,000597	45,22241
150000	77,82636	0,000579	88,81509	0,000673	52,67628
160000	88,38126	0,00062	100,8882	0,000711	59,49697
170000	99,98203	0,000811	113,4855	0,000662	66,48476
180000	112,0039	0,000721	128,1885	0,000698	74,55846
190000	124,3095	0,000804	141,6588	0,000803	83,36614
200000	138,0724	0,000788	157,5635	0,000782	93,38352

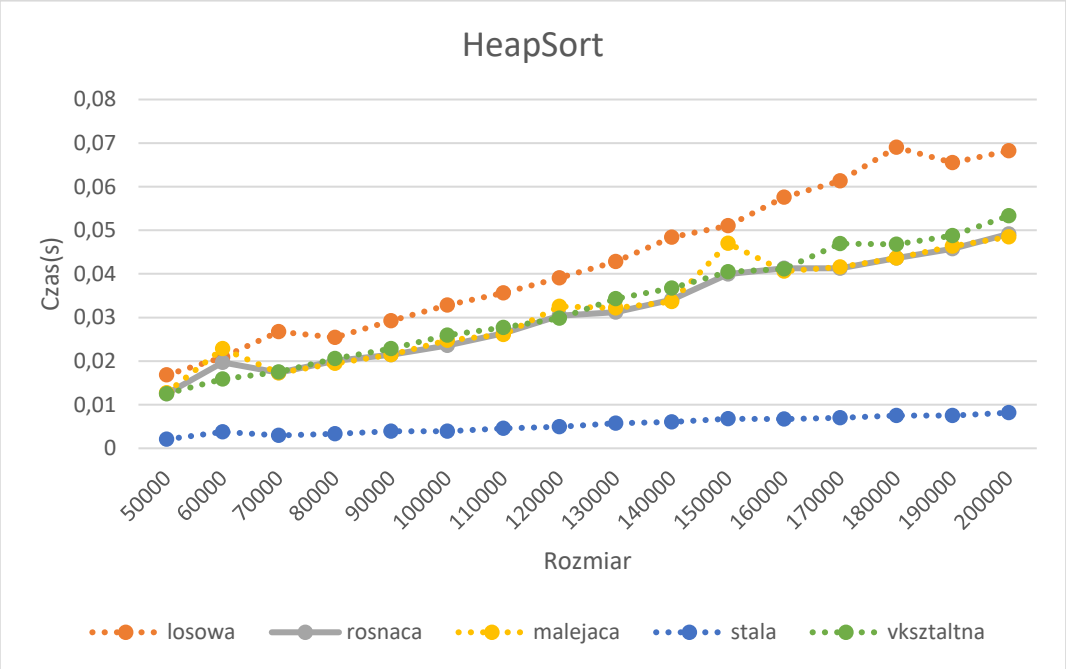




Heap Sort

rozmiar	losowa	rosnaca	malejaca	stala	vksztaltna
50000	0,0168	0,012471	0,01273	0,002059	0,012523
60000	0,020851	0,019709	0,022837	0,003731	0,015869

70000	0,026739	0,017354	0,017278	0,002962	0,01752
80000	0,025423	0,02009	0,019447	0,003335	0,020591
90000	0,029275	0,021444	0,021471	0,003895	0,022888
100000	0,03287	0,023559	0,024735	0,00392	0,02594
110000	0,035663	0,026415	0,02606	0,004569	0,027696
120000	0,039111	0,030435	0,032576	0,004926	0,029803
130000	0,042824	0,031189	0,032161	0,005764	0,03428
140000	0,048383	0,034063	0,033635	0,006036	0,036713
150000	0,05105	0,03993	0,046986	0,006762	0,040445
160000	0,057579	0,041248	0,040647	0,006721	0,041169
170000	0,061326	0,04131	0,041571	0,007021	0,04693
180000	0,069026	0,043603	0,043669	0,007515	0,046788
190000	0,065538	0,045794	0,046342	0,007537	0,048752
200000	0,068246	0,049161	0,048478	0,008167	0,053345



Wnioski:

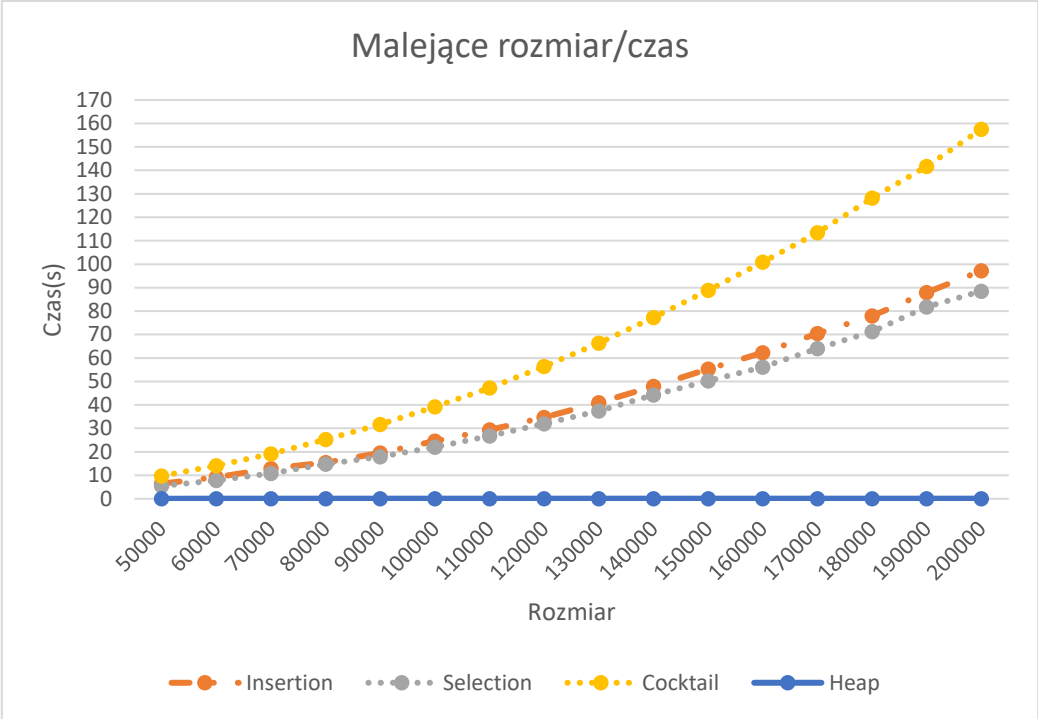
- Insertion Sort - Jest efektywny dla niewielkiej liczby elementów, jego złożoność wynosi $O(n^2)$
- Selection Sort - algorytm o złożoności $O(n^2)$, jest niestabilny
- Cocktail Sort - algorytm o złożoności $O(n^2)$
- Heap Sort – algorytm o złożoności $O(n \log_2 n)$, podstawową zaletą algorytmu sortowania stogowego jest pesymistyczna złożoność O(n log2n), w przypadku średnim sortowanie szybkie jest efektywniejsze niż sortowanie stogowe Najlepszą metodą sortowania okazał się Heap Sort, początkowo inne metody „dawały sobie radę", lecz im nasze tablice miały większy rozmiar tym Heap Sort udowadniał swoją przydatność.

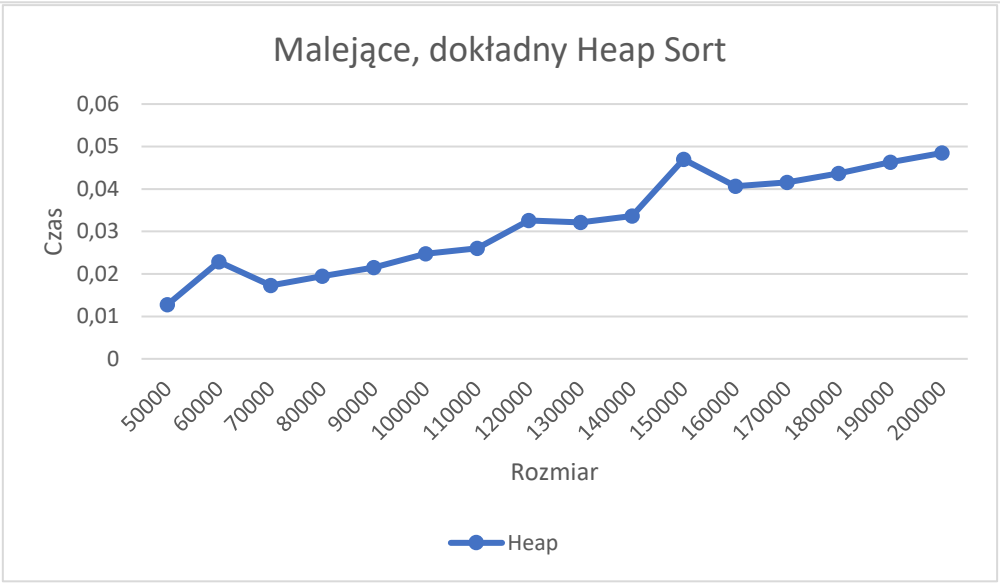
II. Dla różnych typów danych wejściowych porównaj efektywność działania powyższych algorytmów.

Przedstaw wykresy t = f(n) dla każdego typu danych wejściowych i różnych metod sortowania (5 wykresów). Liczbę elementów należy dobrać w taki sposób, aby możliwe było wykonanie pomiarów. Sformułować wnioski odnośnie złożoności obliczeniowej i efektywności wykonywania oraz zachowania się algorytmów dla poszczególnych typów danych wejściowych.

1.Tablica z wartościami malejącymi

rozmiar	Insertion	Selection	Cocktail	Heap
50000	6,4453548	5,48045050	9,777496	0,01273
60000	9,1343367	7,78329550	14,13243	0,022837
70000	12,8122038	10,83249210	19,16336	0,017278
80000	15,5332722	14,79681550	25,33498	0,019447
90000	19,6181364	17,90730710	31,70568	0,021471
100000	24,5914664	21,94922180	39,21574	0,024735
110000	29,4358542	26,83720600	47,29774	0,02606
120000	34,6876511	31,94844900	56,34882	0,032576
130000	40,9229158	37,49991790	66,3465	0,032161
140000	47,9241674	44,23926830	77,32234	0,033635
150000	55,2448786	50,27941220	88,81509	0,046986
160000	62,2278467	56,18531160	100,8882	0,040647
170000	70,3974527	64,05623830	113,4855	0,041571
180000	78,0032353	71,25281020	128,1885	0,043669
190000	87,9052221	81,76733110	141,6588	0,046342
200000	97,2166093	88,53216330	157,5635	0,048478

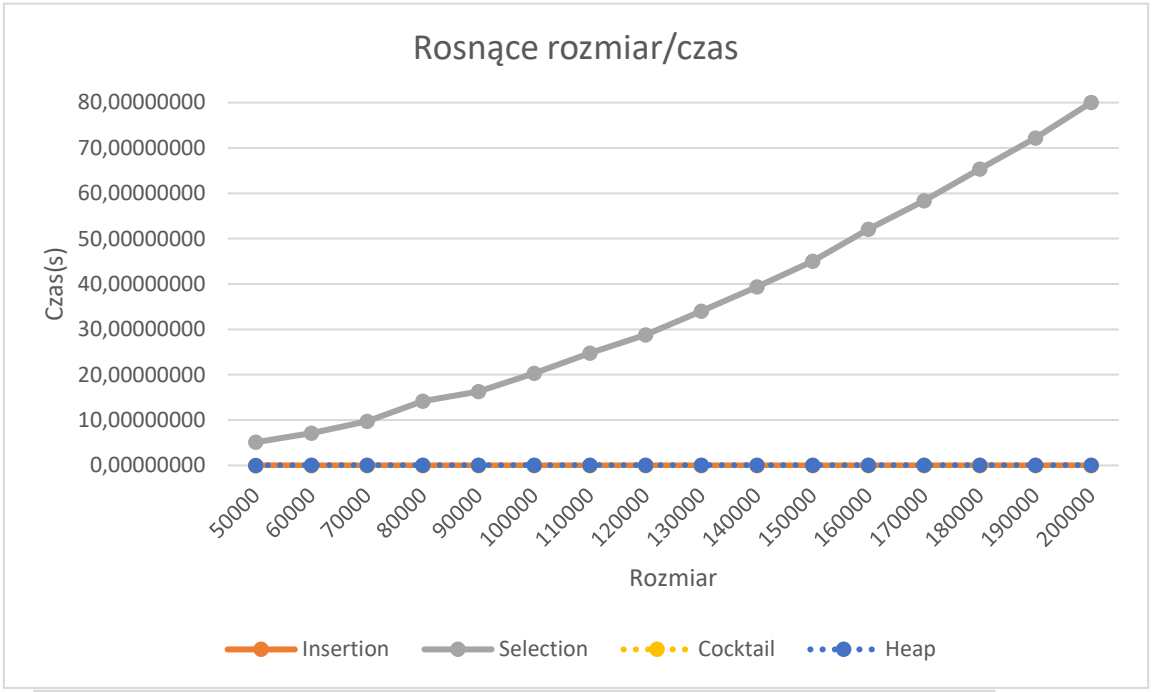


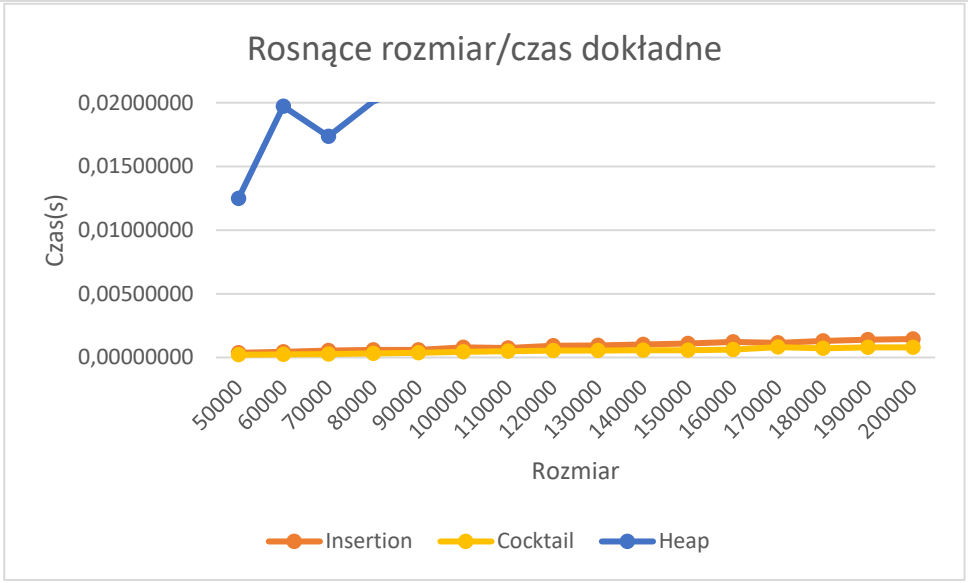


2.Tablica z wartościami rosnącymi

rozmiar	Insertion	Selection	Cocktail	Heap
50000	0,00036730	5,10718480	0,00020500	0,012471
60000	0,00045070	7,07719920	0,00023300	0,019709
70000	0,00055090	9,73722200	0,00027310	0,017354
80000	0,00058680	14,17970830	0,00031040	0,02009
90000	0,00060310	16,28060920	0,00037610	0,021444
100000	0,00080260	20,32235940	0,00045300	0,023559
110000	0,00074300	24,73485260	0,00050240	0,026415
120000	0,00091030	28,78769620	0,00052990	0,030435
130000	0,00094390	34,03475400	0,00053230	0,031189

140000	0,00101010	39,38263160	0,00055740	0,034063
150000	0,00109020	44,99409270	0,00057890	0,03993
160000	0,00121000	52,07130220	0,00061990	0,041248
170000	0,00114380	58,36434170	0,00081100	0,04131
180000	0,00128600	65,36616190	0,00072110	0,043603
190000	0,00138670	72,20847630	0,00080360	0,045794
200000	0,00144590	79,99772570	0,00078770	0,049161

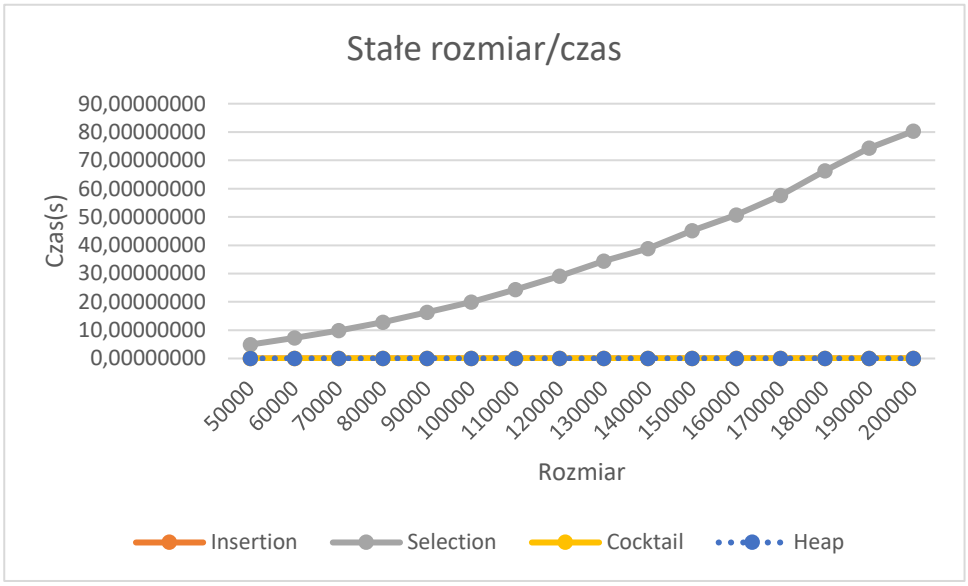


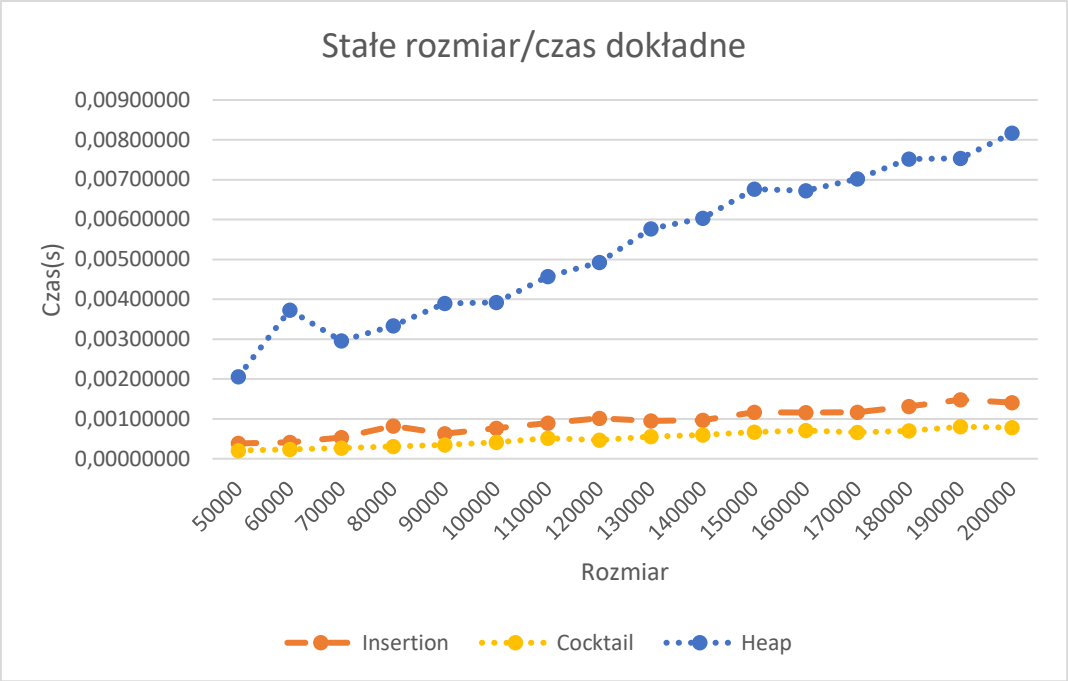


3.Tablica z wartościami stałymi

rozmiar	Insertion	Selection	Cocktail	Heap
50000	0,00038500	4,89498030	0,00020740	0,00205930
60000	0,00040970	7,22965020	0,00023250	0,00373080

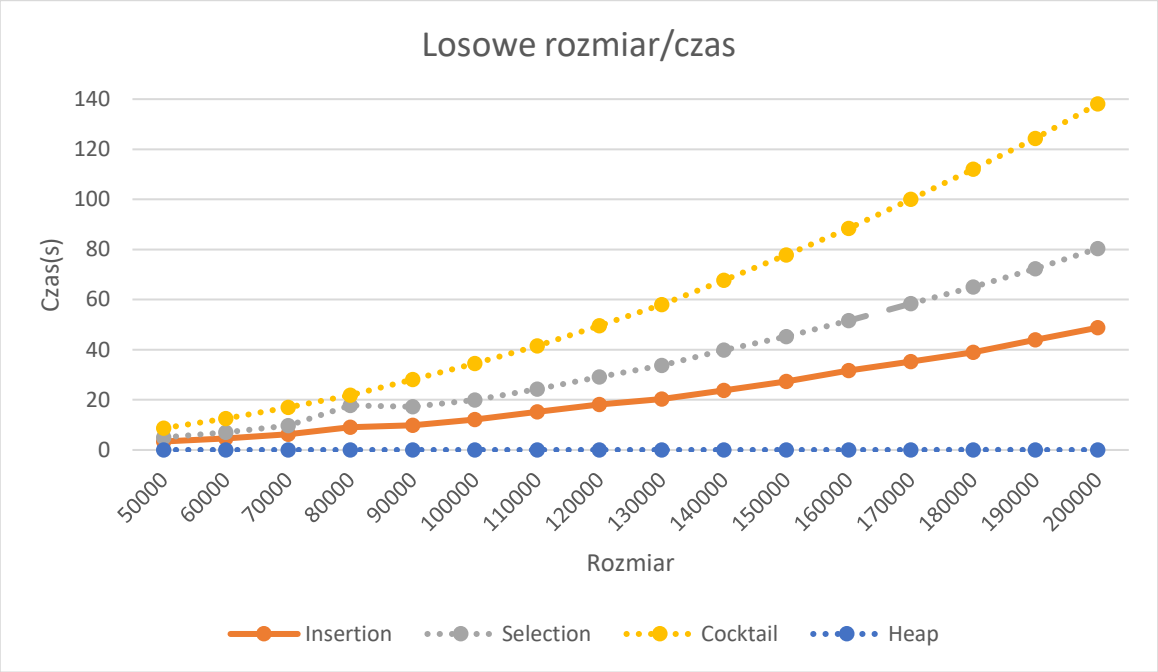
70000	0,00053650	9,91833940	0,00027120	0,00296170
80000	0,00082080	12,84794890	0,00030480	0,00333460
90000	0,00062830	16,33355350	0,00035190	0,00389540
100000	0,00076530	19,98329950	0,00041010	0,00392010
110000	0,00089260	24,33181530	0,00052060	0,00456940
120000	0,00101140	29,10708690	0,00047210	0,00492550
130000	0,00094950	34,36746340	0,00055320	0,00576410
140000	0,00096440	38,80540910	0,00059710	0,00603580
150000	0,00116850	45,09662370	0,00067300	0,00676160
160000	0,00116110	50,69653830	0,00071080	0,00672110
170000	0,00116340	57,63673400	0,00066230	0,00702120
180000	0,00131210	66,25600040	0,00069820	0,00751490
190000	0,00148180	74,37271240	0,00080260	0,00753680
200000	0,00140530	80,27001740	0,00078210	0,00816650

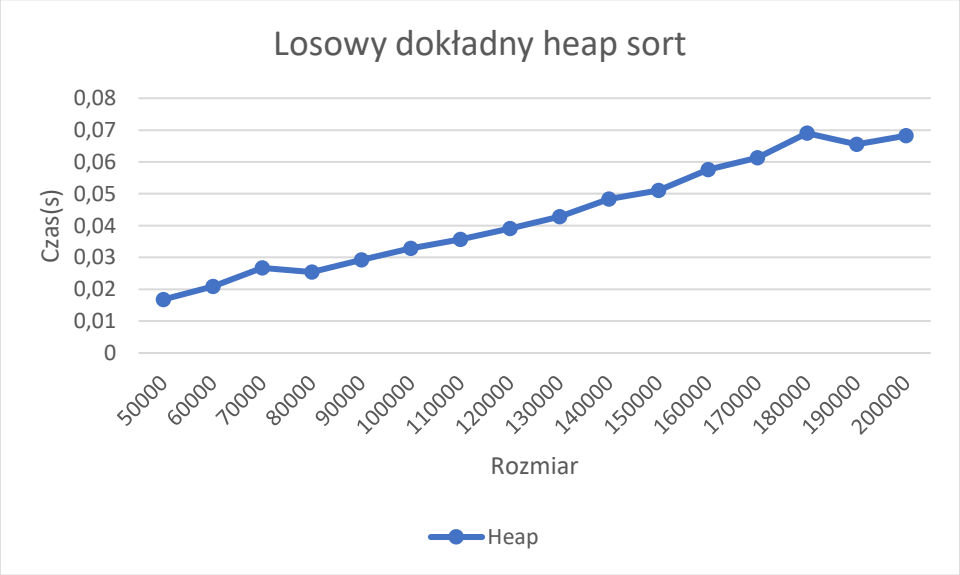




4.Tablica z wartościami losowymi

rozmiar	Insertion	Selection	Cocktail	Heap
50000	3,37828	4,99336040	8,681554	0,0168
60000	4,644208	7,05467170	12,55443	0,020851
70000	6,282491	9,73893410	16,98603	0,026739
80000	9,118548	17,77214620	21,80834	0,025423
90000	9,803106	17,19271090	28,04835	0,029275
100000	12,09686	19,94608060	34,43645	0,03287
110000	15,15885	24,27714450	41,54766	0,035663
120000	18,13819	29,17480550	49,51968	0,039111
130000	20,34681	33,78835490	58,04906	0,042824
140000	23,71873	39,82707660	67,69467	0,048383
150000	27,39687	45,25685670	77,82636	0,05105
160000	31,67395	51,58401210	88,38126	0,057579
170000	35,29276	58,36383970	99,98203	0,061326
180000	38,93988	64,99692770	112,0039	0,069026
190000	43,89687	72,28263640	124,3095	0,065538
200000	48,78219	80,35679130	138,0724	0,068246

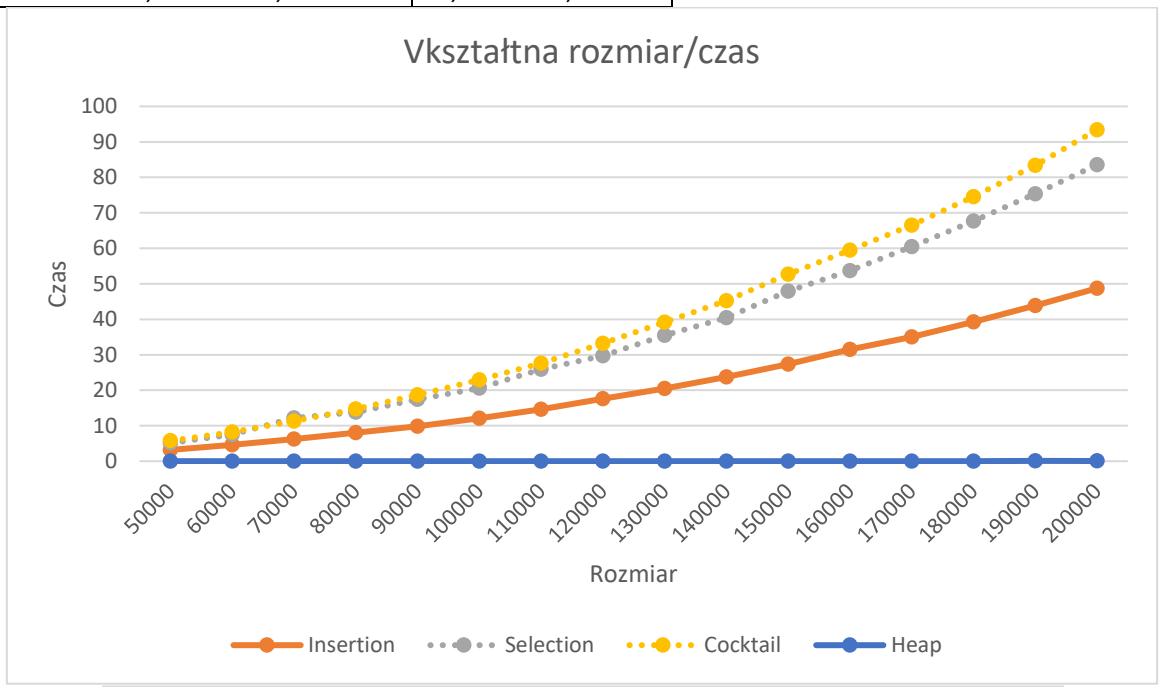




5.Tablica z wartościami v-ksztaltnymi

rozmiar	Insertion	Selection	Cocktail	Heap
50000	3,182591	5,11809030	5,758571	0,012523
60000	4,642068	7,51313170	8,241791	0,015869
70000	6,246745	12,22505700	11,23984	0,01752
80000	7,995688	13,80703310	14,74526	0,020591
90000	9,843123	17,42230520	18,63396	0,022888
100000	12,08662	20,56849700	22,90709	0,02594
110000	14,64825	25,85322920	27,63541	0,027696
120000	17,56357	29,66250020	33,16107	0,029803
130000	20,52604	35,44990610	39,1535	0,03428

140000	23,73064	40,44877100	45,22241	0,036713
150000	27,30439	47,94556220	52,67628	0,040445
160000	31,50588	53,66831380	59,49697	0,041169
170000	35,02633	60,48559710	66,48476	0,04693
180000	39,28196	67,65978220	74,55846	0,046788
190000	43,83687	75,36054410	83,36614	0,048752
200000	48,72306	83,53505330	93,38352	0,053345



Wnioski:

- W przypadku, gdy tablica jest malejąca to najlepiej działa Heap Sort
- Z tablicami rosnącymi najlepiej radzi sobie sortowanie koktajlowe
- Gdy tablica ma stałe wartości to metoda Cocktail Sort nie ma sobie równych

- W przypadku tablicy z wartościami losowymi powinniśmy użyć metody Heap Sort ponieważ wtedy jest najszybsza.
- Z tablicą V-ksztaltna najlepiej sobie radzi Heap Sort.

III. Zaimplementuj algorytm Quicksort w dwóch wersjach: rekurencyjnie oraz iteracyjnie (z własną implementacją stosu). Porównaj obie wersje na wspólnym wykresie przy sortowaniu ciągu losowego. Następnie porównaj różne sposoby wyboru klucza do porównania: skrajnie prawego, środkowego co do położenia, losowo wybranego. Utwórz wykres porównujący efektywność działania algorytmu (iteracyjnego lub rekurencyjnego) w zależności od wyboru różnego klucza dla A-kształtnego ciągu wejściowego (przynajmniej 15 punktów pomiarowych). Sformułuj wnioski dotyczące złożoność badanych algorytmów. Jak wybór klucza wpływa na działanie algorytmu (najgorsze, najlepsze przypadki)?

1.Quick Sort rekurencyjny i iteracyjny.

```
using System;
using System.Numerics;
using System.Diagnostics;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp24
{
class Program
{
private void rekurencyjny(int[] tab, int left, int right) //metoda sortujaca quicksort rekurencyjnie
{
int i, j, x;
i = left;
j = right;
x = tab[(left + right) / 2];

do
{
while (tab[i] < x) i++;
while (x < tab[j]) j--;
```

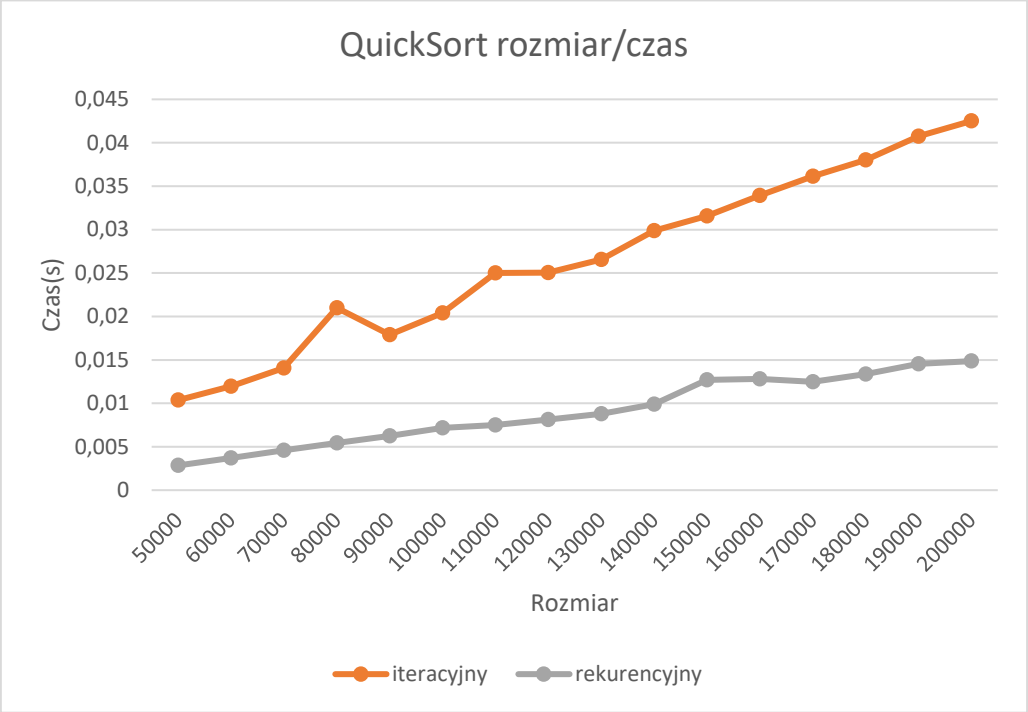
```
            if (i <= j)
                {
            int buf = tab[i];
            tab[i] = tab[j];
              tab[j] = buf;
                i++;
                j--;
                }
            } while (i <= j);

        if (left < j) rekurencyjny(tab, left, j);
       if (i < right) rekurencyjny(tab, i, right);
                }
```

```
public static int[] iteracyjny(int[] t)//metoda sortujaca quicksort iteracyjnie
                {
                int i, j, l, p, sp;
            int[] stos_l = new int[t.Length],
               stos_p = new int[t.Length];
                      sp = 0;
                 stos_l[sp] = 0;
              stos_p[sp] = t.Length - 1;

                      do
                       {
                 l = stos_l[sp];
                 p = stos_p[sp];
                      sp--;

                       do
                        {
                      int x;
                      i = l;
                      j = p;
                 x = t[(l + p) / 2];
                       do
                        {
                while (t[i] < x) i++;
                while (x < t[j]) j--;
                     if (i <= j)
                        {
                   int buf = t[i];
                    t[i] = t[j];
                    t[j] = buf;
                        i++;
                        j--;
                        }
                  } while (i <= j);

                     if (i < p)
                        {
                       sp++;
                  stos_l[sp] = i;
                  stos_p[sp] = p;
                        }
                     p = j;
                } while (l < p);
               } while (sp >= 0);
```

```
            return t;

        }
    static public int[] losowa(int[] tab)
        {
     for (int i = 0; i < tab.Length; i++)
        {
            Random rand = new Random();
      tab[i] = rand.Next(50000, 200000);

        }
      return tab;

        }
     static void Main(string[] args)
        {
Console.WriteLine("rozmiar\titeracyjny\trekurencyjny");
         Stopwatch stopwatch = new Stopwatch();
        int lp = 1;
  for (int i = 50000; i <= 200000; i += 10000)
        {

        int[] tab = new int[i];
    losowa(tab);//zwracanie tablicy losowej
int[] tab2 = tab;//powielenie tablicy by wartosci wylosowane dla obu pomiarow byly
                                    takie same
        stopwatch.Start();//stoper start
  iteracyjny(tab);//zaimplementowana tablica do metody
        stopwatch.Stop();//stoper stop
Console.Write(tab.Length + "\t" + stopwatch.Elapsed.TotalSeconds.ToString("F8"));
       stopwatch.Reset();//reset pomiaru

        stopwatch.Start();//stoper start
  iteracyjny(tab2);//zaimplementowana tablica do metody
        stopwatch.Stop();//stoper stop
Console.Write("\t" + stopwatch.Elapsed.TotalSeconds.ToString("F8"));
       stopwatch.Reset();//reset pomiaru
             Console.Write("\n");
        }


        }
        }
        }
```

rozmiar	iteracyjny	rekurencyjny
50000	0,0103629	0,0028475
60000	0,0119799	0,0037234
70000	0,0140682	0,004582
80000	0,0209883	0,0054532
90000	0,0178928	0,0062353
100000	0,0203987	0,0071643
110000	0,0250222	0,0074841
120000	0,0250408	0,0081185
130000	0,0265534	0,0087743
140000	0,0298797	0,0099131
150000	0,0315867	0,0126991
160000	0,0339369	0,0128054

170000	0,0361505	0,0124908
180000	0,0380323	0,0133452
190000	0,0407643	0,0145352
200000	0,0425048	0,0148578



2.Quick Sort rekurencyjny(różna mediana).

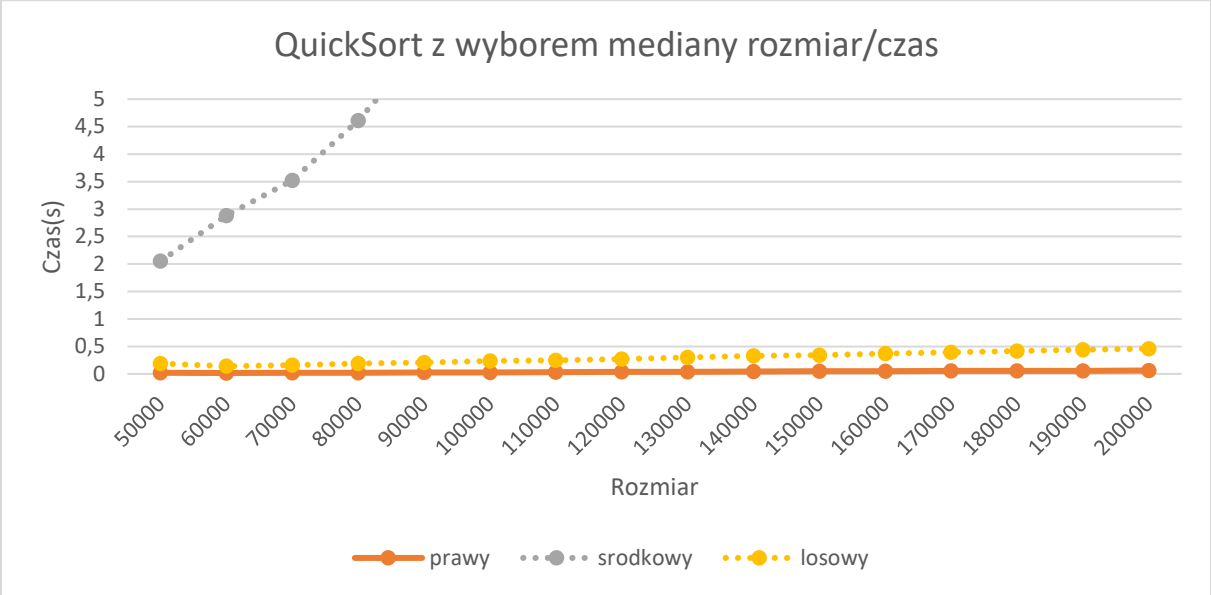
```
using System;
using System.Numerics;
using System.Diagnostics;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;

namespace ConsoleApp24
{
    class Program
    {
        static void rekurencyjnyprawy(int[] tab, int left, int right)//metoda
sortująca quicksort rekurencyjnie z medianą z prawej
        {
            int i, j, x;
            i = left;
            j = right;
            x = tab[right];

            do
            {
                while (tab[i] < x) i++;
                while (x < tab[j]) j--;
                if (i <= j)
                {
                    int buf = tab[i];
```

```
                tab[i] = tab[j];
                tab[j] = buf;
                i++;
                j--;
            }
        } while (i <= j);

        if (left < j) rekurencyjnyprawy(tab, left, j);
        if (i < right) rekurencyjnyprawy(tab, i, right);
    }
    static void rekurencyjnysrodkowy(int[] tab, int left, int right)//metoda
sortująca quicksort rekurencyjnie z medianą na środku
    {
        int i, j, x;
        i = left;
        j = right;
        x = tab[(left + right) / 2];

        do
        {
            while (tab[i] < x) i++;
            while (x < tab[j]) j--;
            if (i <= j)
            {
                int buf = tab[i];
                tab[i] = tab[j];
                tab[j] = buf;
                i++;
                j--;
            }
        } while (i <= j);

        if (left < j) rekurencyjnysrodkowy(tab, left, j);
        if (i < right) rekurencyjnysrodkowy(tab, i, right);
    }
    static void rekurencyjnylosowy(int[] tab, int left, int right)//metoda
sortująca quicksort rekurencyjnie z medianą losową
    {
        int i, j, x;
        i = left;
        j = right;
        Random rand = new Random(Guid.NewGuid().GetHashCode());
        x = tab[rand.Next(right - left) + left];

        do
        {
            while (tab[i] < x) i++;
            while (x < tab[j]) j--;
            if (i <= j)
            {
                int buf = tab[i];
                tab[i] = tab[j];
                tab[j] = buf;
                i++;
                j--;
            }
        } while (i <= j);

        if (left < j) rekurencyjnylosowy(tab, left, j);
        if (i < right) rekurencyjnylosowy(tab, i, right);
    }
```

```
        static public int[] aksztaltna(int[] tab)//metoda zwaraca tablice aksztaltna
        {
            int a = tab.Length - 1;
            int b = 1;
            for (int i = 0; i < tab.Length; i++)
            {
                if (i == tab.Length / 2)
                {
                    tab[i] = tab.Length;
                }
                else if (i < tab.Length / 2)
                {
                    tab[i] = b;
                    b += 2;
                }
                else
                {
                    tab[i] = a;
                    a -= 2;
                }
            }
            return tab;


        }
        static void sw()
        {
            Console.WriteLine("rozmiar\tprawy\tsrodkowy\tlosowy");
            Stopwatch stopwatch = new Stopwatch();
            int lp = 1;
            for (int i = 50000; i <= 200000; i += 10000)
            {

                int[] tab = new int[i];
                aksztaltna(tab);//zwraca tablice aksztalta

                stopwatch.Start();//stoper start
                rekurencyjnyprawy(tab, 0, tab.Length - 1);//zaimplementowana tablica
do metody
                stopwatch.Stop();//stoper stop
                Console.Write(tab.Length + "\t" +
stopwatch.Elapsed.TotalSeconds.ToString("F8"));
                stopwatch.Reset();//reset pomiaru
                aksztaltna(tab);//zwraca tablice aksztaltna
                stopwatch.Start();//stoper start
                rekurencyjnysrodkowy(tab, 0, tab.Length - 1);//zaimplementowana
tablica do metody
                stopwatch.Stop();//stoper stop
                Console.Write("\t" + stopwatch.Elapsed.TotalSeconds.ToString("F8"));
                stopwatch.Reset();//reset pomiaru
                aksztaltna(tab);//zwraca tablice aksztaltna
                stopwatch.Start();//stoper start
                rekurencyjnylosowy(tab, 0, tab.Length - 1);//zaimplementowana tablica
do metody
                stopwatch.Stop();//stoper stop
                Console.Write("\t" + stopwatch.Elapsed.TotalSeconds.ToString("F8"));
```

```
                stopwatch.Reset();//reset pomiaru
                Console.Write("\n");
            }
        }
        static void Main(string[] args)
        {
            ThreadStart operation = new ThreadStart(sw);
            Thread thr = new Thread(operation, 40000000);
            thr.Start();
            Console.ReadKey();









        }
    }
}
```

prawy	srodkowy	losowy
0,022533	2,0511622	0,189431
0,016873	2,8814094	0,140985
0,018982	3,5232043	0,161161
0,022074	4,6090031	0,190088
0,02709	5,9793505	0,20566
0,029175	7,289343	0,23767
0,031996	8,7568562	0,249924
0,035505	10,3097865	0,272761
0,038374	12,0825681	0,298202
0,041108	14,0710407	0,327065
0,048222	16,1821778	0,341614
0,05013	18,3492559	0,367407
0,054163	20,9796833	0,395579
0,057056	23,2803207	0,414217
0,057488	25,9404045	0,438611
0,062909	28,6813831	0,459318



Wnioski:

- Quick Sort rekurencyjny jest bardziej wydajny od iteracyjnego
- Quick Sort rekurencyjny najlepiej sobie radzi, gdy mediana jest skrajnie prawym elementem tablicy
- Quick Sort rekurencyjny najgorzej sobie radzi, gdy mediana jest środkiem.