PROJEKT 3

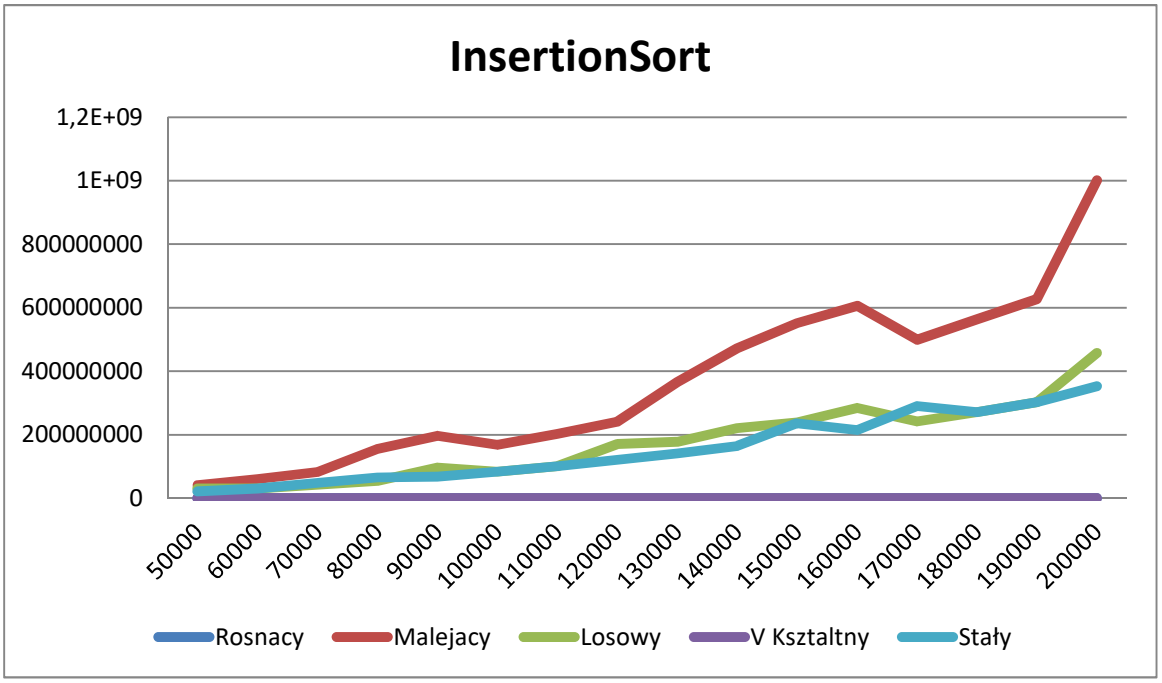
Piotr Waszak grupa k28 indeks 96053

Projekt ma na celu przeprowadzić porównanie, różnych metod sortowania, oraz wykazać ich skuteczność dla poszczególnych ciągów danych.

I część projektu

Należy porównać szybkość działania 4 metod sortowania: Insertion Sort, Selection Sort, Heap Sort, Coctail Sort dla liczb całkowitych rzędu 50 000 – 200 000 elementów generowanych w postaci: losowej, rosnącej, malejącej, stałej oraz v-kształtnej.

1. Algorytm InsertionSort jest stabliny, a jego złożoność to O (n^2). W przpadku większych tablic, zmniejsza liczbe porównań (O (log n)).



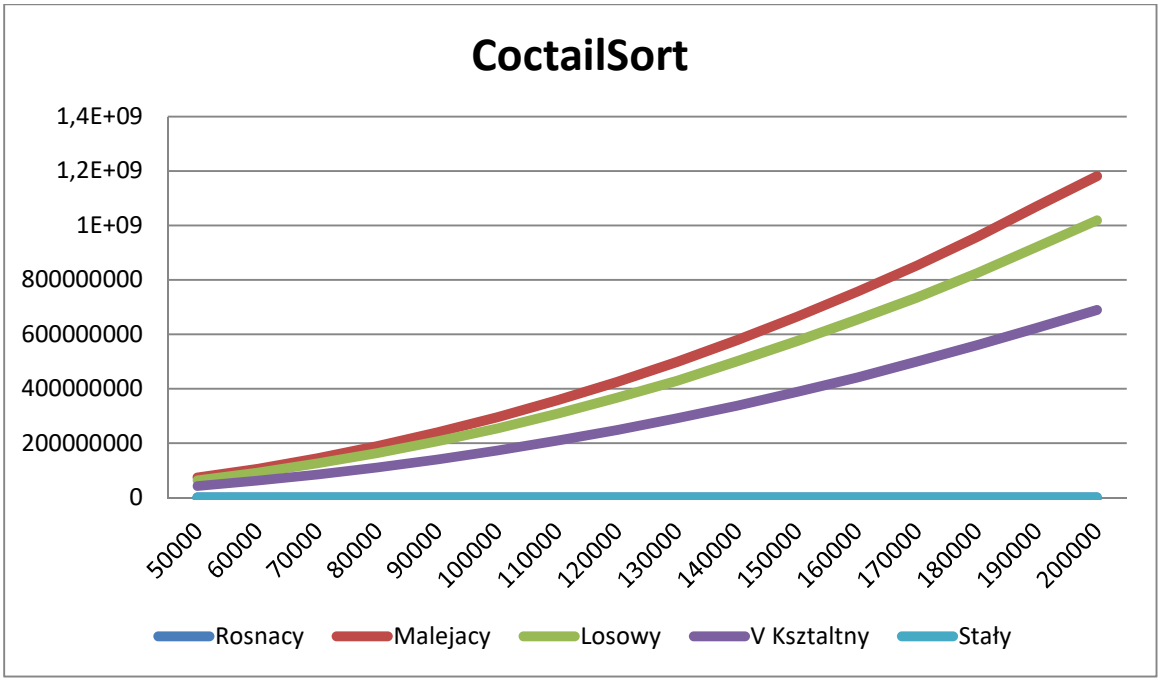
Legenda: oś Y: czas, oś X: Wielkość tablicy

	Rosnacy	Malejacy	Losowy	V Ksztaltny	Stały
50000	3543	41853303	30630468	88741	21229341
60000	2812	60977188	30370610	103926	30444854
70000	3428	82944380	42681784	122599	48712073
80000	3883	155151417	54791011	141856	65213512
90000	4365	196313850	97054494	161083	67932205
100000	4859	168337867	84022145	180245	83840218
110000	5527	202769530	101371170	199251	101354665
120000	5941	241121281	170592019	218244	120613891
130000	6370	366005474	177434037	237303	141540218
140000	6916	472031164	220788284	257666	164192517
150000	7355	551012158	238487278	277397	235807909
160000	7834	606162765	284351713	297592	214421820
170000	8320	498628224	241977105	318740	290612402
180000	8823	563135312	271621573	338137	271232137
190000	9295	626588883	302914728	358220	302247829
200000	9770	1000847720	457375133	401189	352734239

Najgorzej poradził sobie z: tablicą liczb malejących.

Najlepiej poradził sobie z: tablicą liczb rosnących.

2. Algorytm CoctailSort jest stabliny, a jego złożoność to O (n^2).

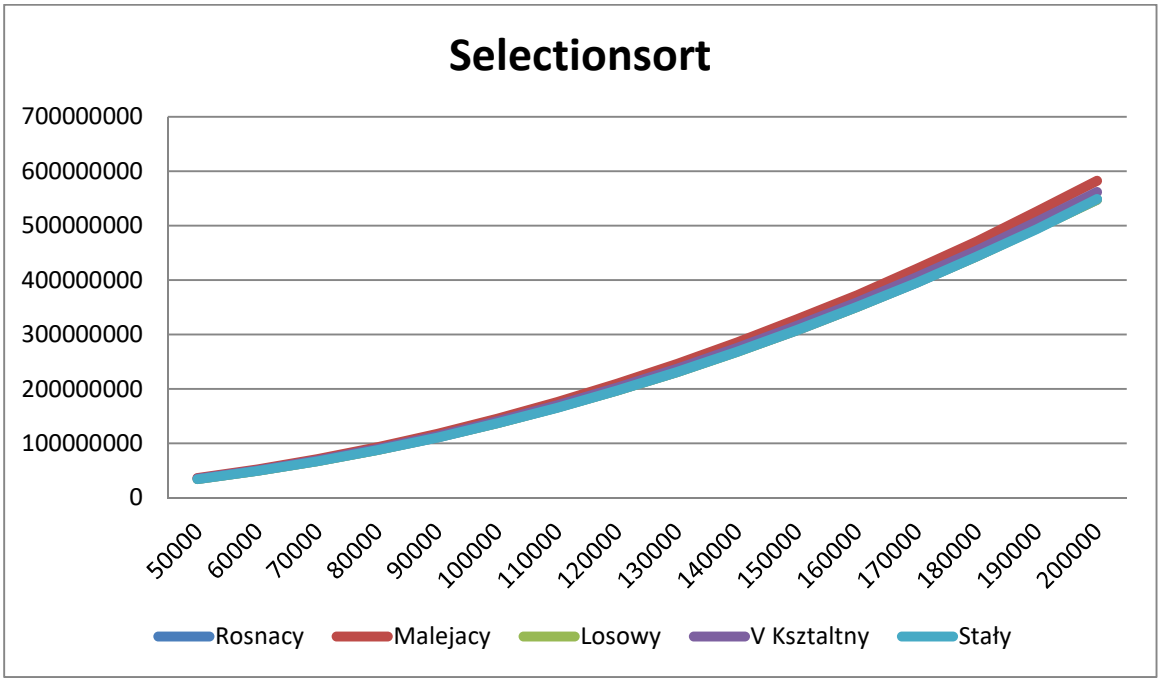


	Rosnacy	Malejacy	Losowy	V Ksztaltny	Stały
50000	3762	73757616	63436262	42971550	1456
60000	1778	106278830	91734859	62146562	1827
70000	2232	144685895	125864145	84364275	2036
80000	2327	188958397	163487416	110178374	2353
90000	2740	239880136	206825535	139707677	2619
100000	3049	294979522	254040780	172607694	2928
110000	3245	357253090	308065690	209288917	3338
120000	3498	425234782	366982082	248164499	3513
130000	3870	498883510	429897594	291440086	3849
140000	4105	578442033	501407369	337930554	4102
150000	4412	664607764	575214645	387951260	4415
160000	4674	756331304	654130922	441493804	4824
170000	5036	853123857	735269730	499660664	5077
180000	5720	957067722	824762918	560086806	5651
190000	5557	1070264127	920575428	623006805	5580
200000	5940	1181386232	1019075732	689324799	5907

Najgorzej poradził sobie z: tablicą liczb malejących.

Najlepiej poradził sobie z: tablicą liczb rosnących a także z tablica liczb stałych.

3. Algorytm SelectionSort nie jest stabliny, wydajność jest podobna jak w metodzie prostego wstawiania. Złożoność algorytmu to O (n^2).

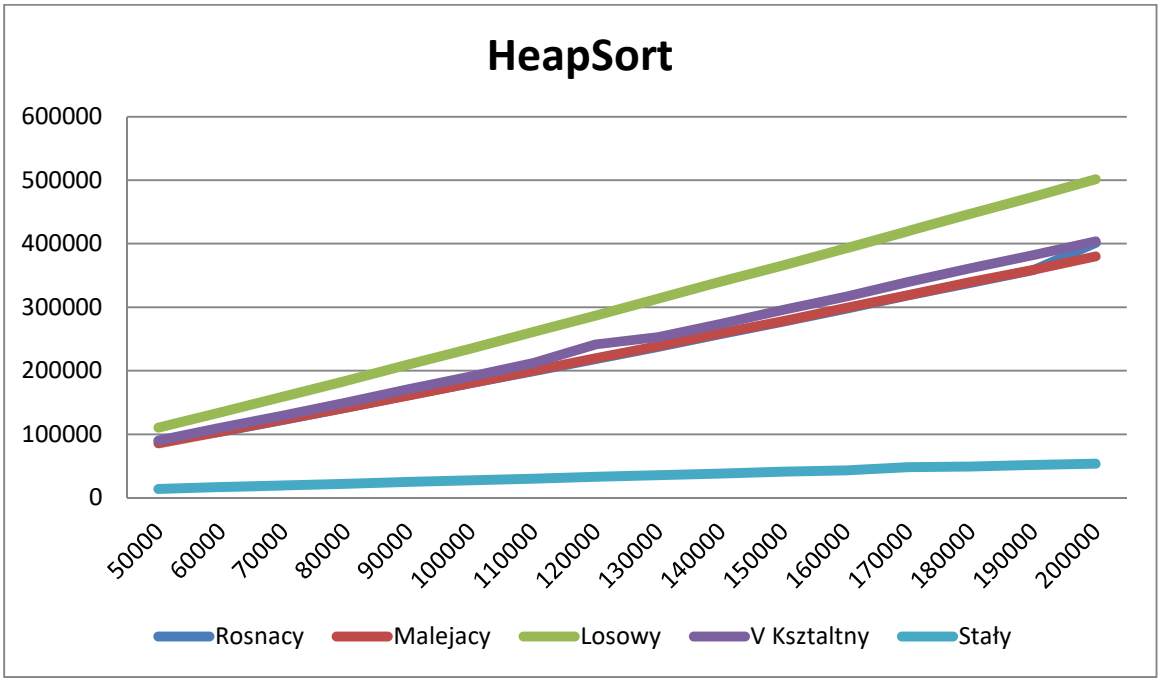


	Rosnacy	Malejacy	Losowy	V Ksztaltny	Stały
50000	34217275	36343355	34259367	35115642	34235196
60000	49269514	52382163	49316295	50538413	49306840
70000	67058432	71227622	67138464	68788253	67156848
80000	87576736	93033552	87664193	89843190	87598212
90000	110854596	117751564	110935427	113733264	110852025
100000	136848489	145502223	136925975	140366233	136854890
110000	165584335	175915190	165699121	169962237	165569988
120000	197063100	209424696	197165782	202521693	197045503
130000	231240133	245947675	231383009	237246692	231231939
140000	268531224	285001532	268353742	275133426	268229270
150000	307819999	328309351	308106046	315823102	307928334
160000	350295820	372273867	350547017	359374347	350295068
170000	395197707	420791196	395672923	405733224	395443618
180000	443310095	471177037	443695733	454795438	443404575
190000	495591887	525761774	494419397	507095090	493527579
200000	547780673	582340842	547224758	561799759	548653324

Najgorzej poradził sobie z: tablicą liczb malejących.

Najlepiej poradził sobie z: tablicą liczb losowych.

4. Algorytm HeapSort nie jest stabilny, a jego złożoność to O (n $\log_2$n).



	Rosnacy	Malejacy	Losowy	V Ksztaltny	Stały
50000	88741	85791	110261	90294	13616
60000	103926	104457	134547	110296	16424
70000	122599	123180	159422	129511	19069
80000	141856	141891	184221	149890	21589
90000	161083	161048	209758	170973	24551
100000	180245	180317	234809	190995	27262
110000	199251	200279	260935	212146	29820
120000	218244	219654	286938	241483	32714
130000	237303	238984	313643	252821	35267
140000	257666	258943	340083	273971	37946
150000	277397	278396	365941	296131	40888
160000	297592	299133	392475	316732	43055
170000	318740	318999	419744	339774	47802
180000	338137	339570	447026	361180	49160
190000	358220	358999	473964	381935	51636
200000	401189	380283	501739	403975	53710

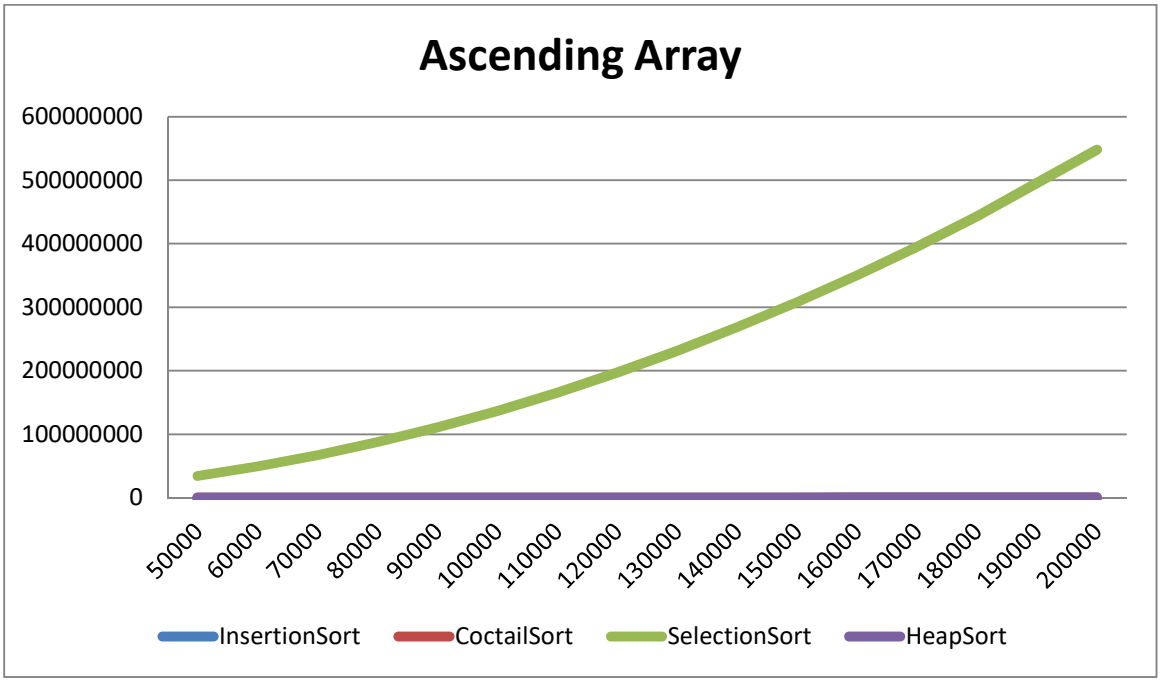
Najgorzej poradził sobie z: tablicą liczb losowych.

Najlepiej poradził sobie z: tablicą liczb stałych.

II część projektu

Należy porównać szybkość działania na wygenerowanych tablicach z poprzedniej części, algorytmy sortowania Insertion Sort, Selection Sort, Coctail Sort oraz Heap Sort.

1. Tablica wygenerowana rosnąco.

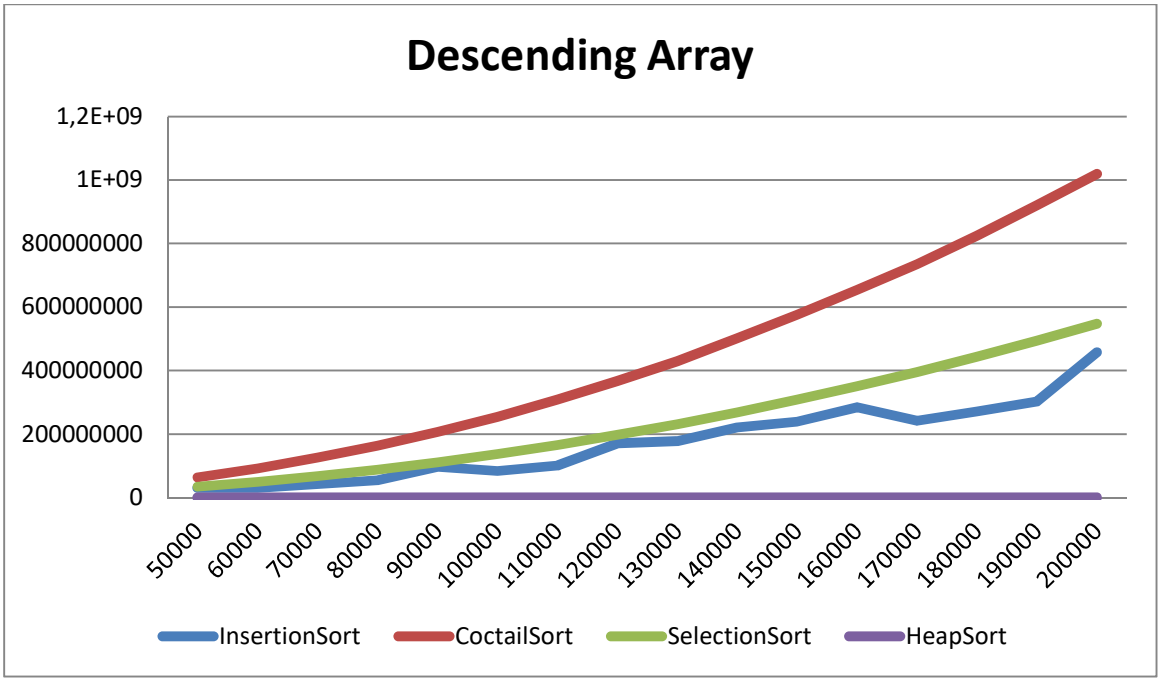


	InsertionSort	CoctailSort	SelectionSort	HeapSort
50000	3543	3762	34217275	88741
60000	2812	1778	49269514	103926
70000	3428	2232	67058432	122599
80000	3883	2327	87576736	141856
90000	4365	2740	110854596	161083
100000	4859	3049	136848489	180245
110000	5527	3245	165584335	199251
120000	5941	3498	197063100	218244
130000	6370	3870	231240133	237303
140000	6916	4105	268531224	257666
150000	7355	4412	307819999	277397
160000	7834	4674	350295820	297592
170000	8320	5036	395197707	318740
180000	8823	5720	443310095	338137
190000	9295	5557	495591887	358220
200000	9770	5940	547780673	401189

Najwolniejszy Algorytm: SelectionSort

Najszybszy Algorytm: CoctailSort

2. Tablica wygenerowana malejąco.

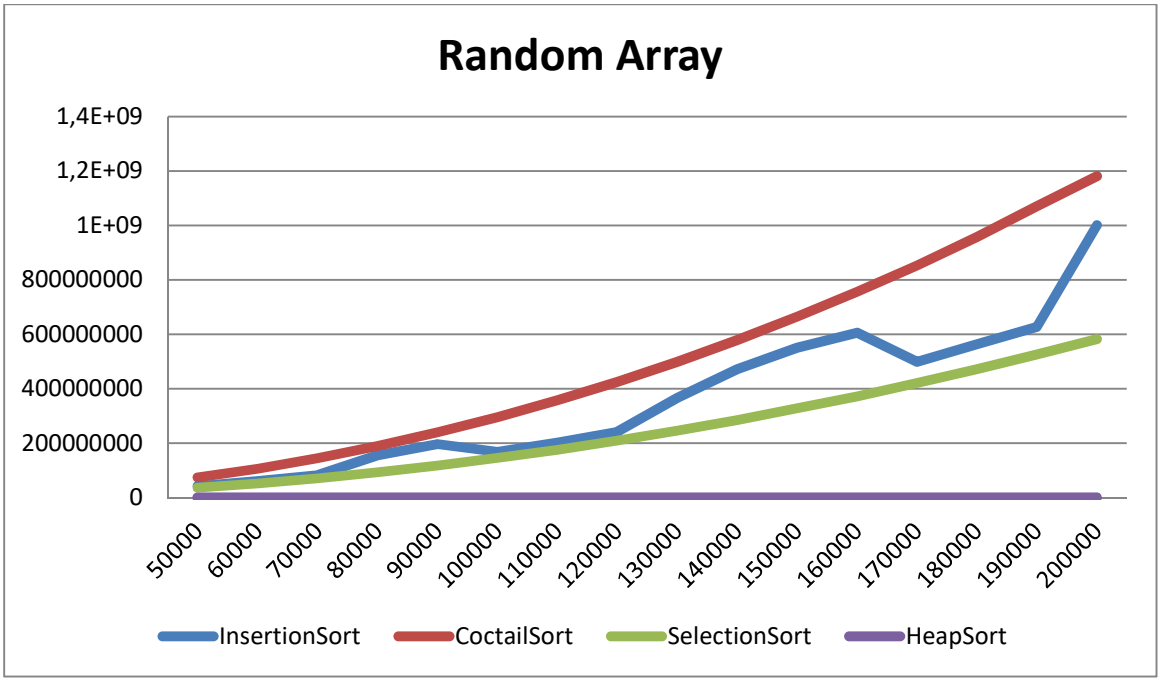


	InsertionSort	CoctailSort	SelectionSort	HeapSort
50000	30630468	63436262	34259367	110261
60000	30370610	91734859	49316295	134547
70000	42681784	125864145	67138464	159422
80000	54791011	163487416	87664193	184221
90000	97054494	206825535	110935427	209758
100000	84022145	254040780	136925975	234809
110000	101371170	308065690	165699121	260935
120000	170592019	366982082	197165782	286938
130000	177434037	429897594	231383009	313643
140000	220788284	501407369	268353742	340083
150000	238487278	575214645	308106046	365941
160000	284351713	654130922	350547017	392475
170000	241977105	735269730	395672923	419744
180000	271621573	824762918	443695733	447026
190000	302914728	920575428	494419397	473964
200000	457375133	1019075732	547224758	501739

Najwolniejszy Algorytm: CoctailSort

Najszybszy Algorytm: HeapSort

3. Tablica wygenerowana losowo.

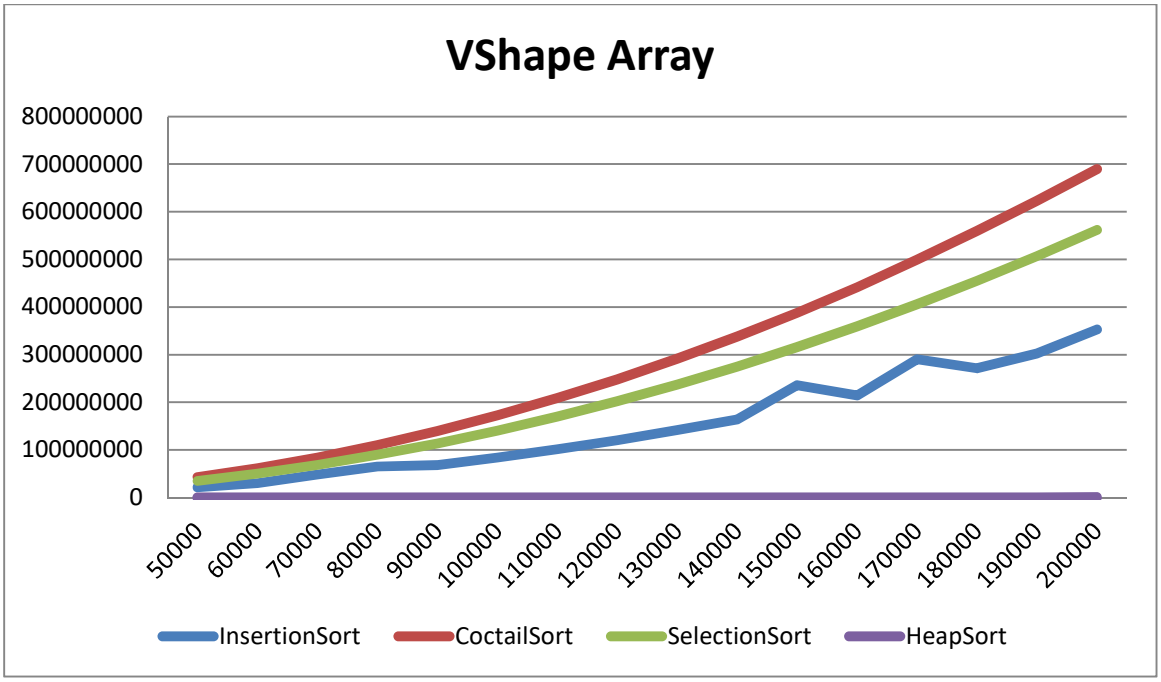


	InsertionSort	CoctailSort	SelectionSort	HeapSort
50000	41853303	73757616	36343355	85791
60000	60977188	106278830	52382163	104457
70000	82944380	144685895	71227622	123180
80000	155151417	188958397	93033552	141891
90000	196313850	239880136	117751564	161048
100000	168337867	294979522	145502223	180317
110000	202769530	357253090	175915190	200279
120000	241121281	425234782	209424696	219654
130000	366005474	498883510	245947675	238984
140000	472031164	578442033	285001532	258943
150000	551012158	664607764	328309351	278396
160000	606162765	756331304	372273867	299133
170000	498628224	853123857	420791196	318999
180000	563135312	957067722	471177037	339570
190000	626588883	1070264127	525761774	358999
200000	1000847720	1181386232	582340842	380283

Najwolniejszy Algorytm: CoctailSort

Najszybszy Algorytm: HeapSort

4. Tablica wygenerowana na kształt litery V.

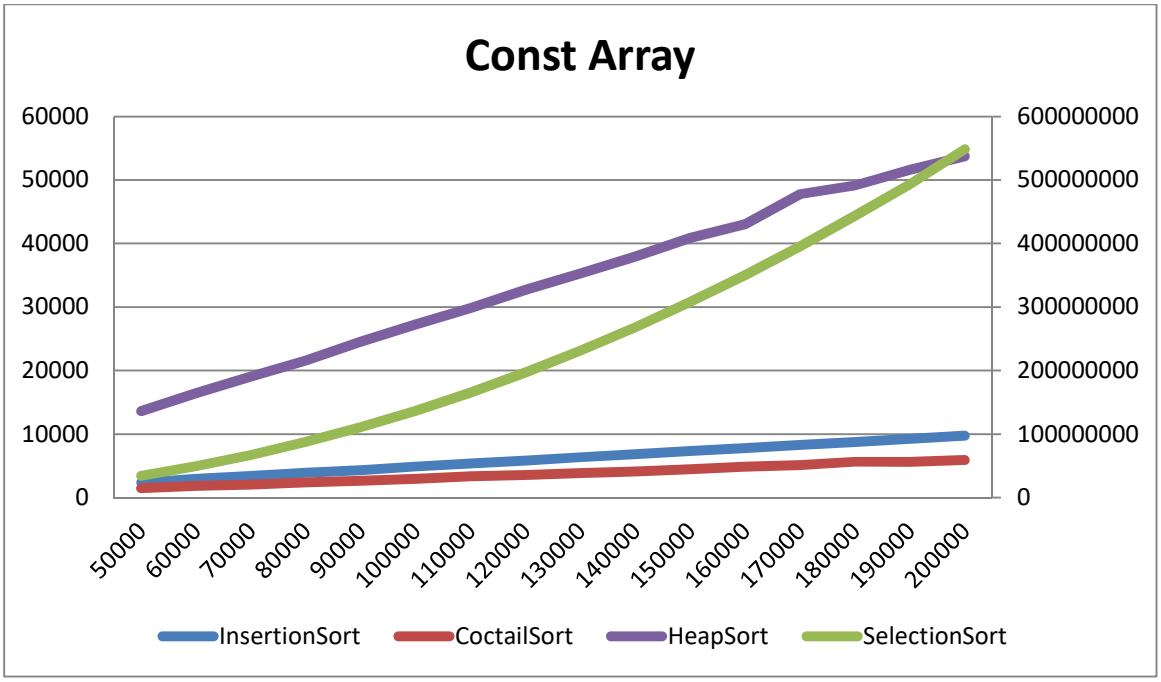


	InsertionSort	CoctailSort	SelectionSort	HeapSort
50000	21229341	42971550	35115642	90294
60000	30444854	62146562	50538413	110296
70000	48712073	84364275	68788253	129511
80000	65213512	110178374	89843190	149890
90000	67932205	139707677	113733264	170973
100000	83840218	172607694	140366233	190995
110000	101354665	209288917	169962237	212146
120000	120613891	248164499	202521693	241483
130000	141540218	291440086	237246692	252821
140000	164192517	337930554	275133426	273971
150000	235807909	387951260	315823102	296131
160000	214421820	441493804	359374347	316732
170000	290612402	499660664	405733224	339774
180000	271232137	560086806	454795438	361180
190000	302247829	623006805	507095090	381935
200000	352734239	689324799	561799759	403975

Najwolniejszy Algorytm: CoctailSort

Najszybszy Algorytm: HeapSort

5. Tablica wygenerowana dla stalej liczby.



* Wyłącznie SelectionSort korzysta z danych po stronie prawej. Został wstawiony pomocniczy Axis, ponieważ pozostałe wyniki były niewidoczne.

	InsertionSort	CoctailSort	SelectionSort	HeapSort
50000	2356	1456	34235196	13616
60000	2960	1827	49306840	16424
70000	3400	2036	67156848	19069
80000	3902	2353	87598212	21589
90000	4269	2619	110852025	24551
100000	4869	2928	136854890	27262
110000	5349	3338	165569988	29820
120000	5781	3513	197045503	32714
130000	6336	3849	231231939	35267
140000	6829	4102	268229270	37946
150000	7288	4415	307928334	40888
160000	7756	4824	350295068	43055
170000	8281	5077	395443618	47802
180000	8738	5651	443404575	49160
190000	9228	5580	493527579	51636
200000	9718	5907	548653324	53710

Najwolniejszy Algorytm: SelectionSort

Najszybszy Algorytm: CoctailSort

Algorytm InsertionSort, mimo prostoty ani razu nie okazał się być najgorszy. Algorytm jest stabliny, a jego złożoność to O (n^2). W przpadku większych tablic, zmniejsza liczbe porównań (O (log n)).

Algorytm CoctailSort w testach wypadł dość ciekawie, ponieważ w trzech przypadkach na pięć wypadł najgorzej, jednak w pozostałych wypadł najlepiej. Algorytm jest stabliny, a jego złożoność to O (n^2).

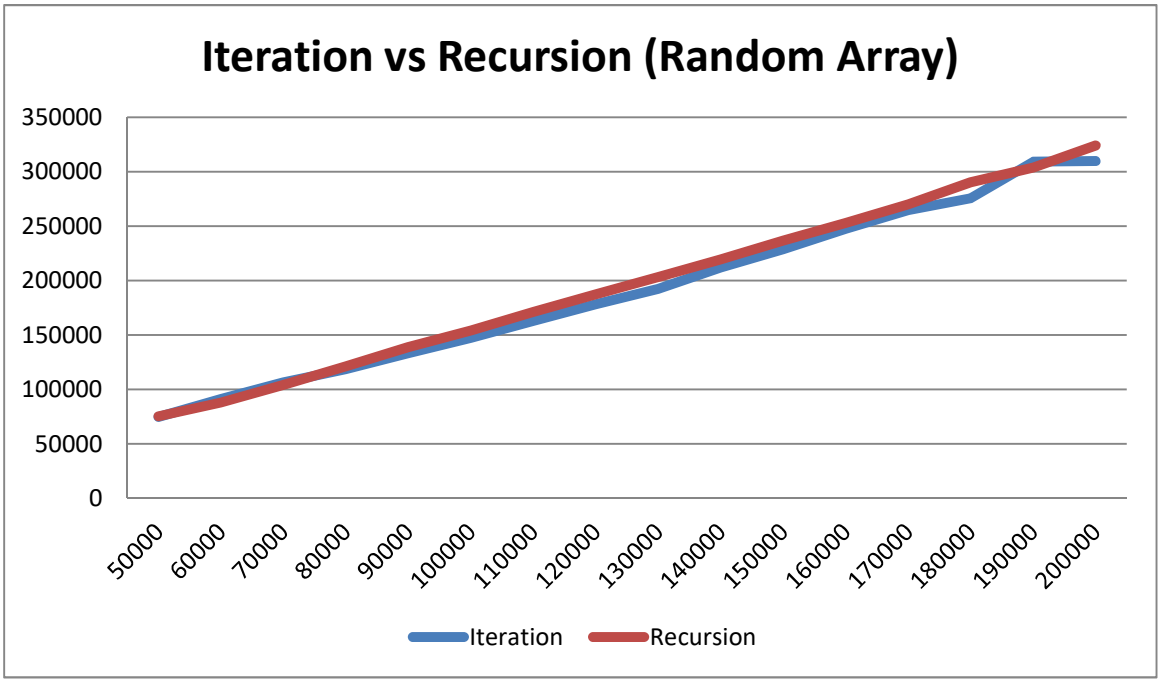
Algorytm SelectionSort, w wielu testach był zdecydowanie najwolniejszy, w sortowaniu liczb losowych zanotował bardzo ładny wynik. Jest więc niestabliny, a jego złożność to O (n^2).

Algorytm HeapSort mimo że nie zawsze okazywał się najszybszy, praktycznie w kazdym teście był efektywny. Algorytm nie jest stabliny, a jego złożoność to O ($n \log_2 n$).

III część projektu

Należy porównać szybkość sortowania algorytmu QuickSort, w wersji iteracyjnej, oraz rekurencyjnej na tablicy liczb losowych.
Wersje rekurencyjną należy, również poddać eksperymentowi zmieniając położenie pivota w miejsce losowe, środkowe, oraz w ostatni indeks tablicy. Tablica do drugiego eksperemntu ma generować liczby w kształt litery A.

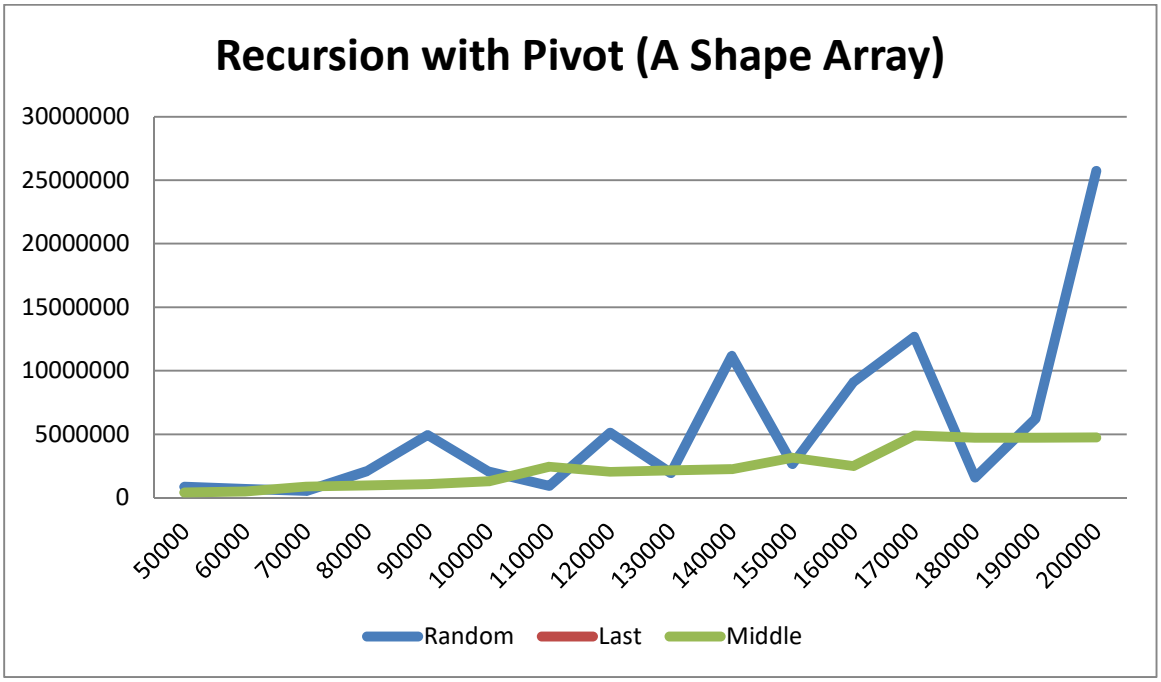
1. Algorytm QuickSort jest prosty w implementacji dla wersji rekurencyjnej. Złożoność średnia to $O(n \log_2 n)$, pesymistycznej $O(n^2)$. Algorytm nie jest stabliny.



	Iteration	Recursion
50000	74607	75247
60000	91154	87877
70000	106356	104119
80000	118468	121192
90000	133067	138781
100000	147447	154189
110000	162991	171150
120000	178339	187211
130000	192446	203105
140000	212042	219272
150000	228749	236691
160000	247607	252953
170000	264719	269878
180000	275647	290067
190000	308941	303882
200000	309654	323894

Aglorytm Iteracyjny jest nieznacznie szybszy, od algorytmu rekurencyjnego.

2. Algorytm sortowania QuickSort rekurencyjny, w trzech wariantach umieszczenia pivota.



	Random	Last	Middle
50000	846516	413725	411455
60000	686773	486577	487930
70000	542734	878089	879772
80000	2102206	954775	955489
90000	4937766	1050092	1052861
100000	2062511	1287549	1290437
110000	928711	2435448	2436264
120000	5124831	2035642	2035834
130000	1951494	2139667	2142888
140000	11177983	2240897	2241001
150000	2638508	3146720	3143334
160000	9090481	2504118	2508037
170000	12678971	4884400	4888352
180000	1594626	4712223	4714582
190000	6227291	4705532	4710736
200000	25742701	4741240	4747030

Najwolniej wypadł pivot w losowym miejscu.

Najszybciej wypadł pivot na ostatnim miejscu.

Kod w C# dla I i II części projektu.

```
using
System;
          using System.Diagnostics;
          namespace Sortowanie
          {
              class Program
              {
                  static void Main(string[] args)
                  {
                      // InsertionSort
                      for (int i = 50000; i <= 200000; i += 10000)
                      {
                          int[] tablica = tabAscending(i);
                          long start = Stopwatch.GetTimestamp();
                          InsertionSort(tablica);
                          long stop = Stopwatch.GetTimestamp();
                          Console.WriteLine("InsertionSort: Rosnacy:{0}; {1}", i, stop -
          start);
                      }
                      for (int i = 50000; i <= 200000; i += 10000)
                      {
                          int[] tablica = tabDescending(i);
                          long start = Stopwatch.GetTimestamp();
                          InsertionSort(tablica);
                          long stop = Stopwatch.GetTimestamp();
                          Console.WriteLine("InsertionSort: Malejacy: {0}; {1}", i, stop -
          start);
                      }
                      for (int i = 50000; i <= 200000; i += 10000)
                      {
                          int[] tablica = tabRandom(i);
                          long start = Stopwatch.GetTimestamp();
                          InsertionSort(tablica);
                          long stop = Stopwatch.GetTimestamp();
                          Console.WriteLine("InsertionSort: Losowy: {0}; {1}", i, stop -
          start);
                      }
                      for (int i = 50000; i <= 200000; i += 10000)
                      {
                          int[] tablica = tabVShape(i);
                          long start = Stopwatch.GetTimestamp();
                          InsertionSort(tablica);
                          long stop = Stopwatch.GetTimestamp();
                          Console.WriteLine("InsertionSort: V Ksztaltny: {0}; {1}", i, stop
```

```
- start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabConst(i);
                long start = Stopwatch.GetTimestamp();
                InsertionSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("InsertionSort: Staly: {0}; {1}", i, stop -
start);
            }

            //CoctailSort
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabAscending(i);
                long start = Stopwatch.GetTimestamp();
                CocktailSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("CocktailSort: Rosnacy:{0}; {1}", i, stop -
start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabDescending(i);
                long start = Stopwatch.GetTimestamp();
                CocktailSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("CocktailSort: Malejacy: {0}; {1}", i, stop -
start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabRandom(i);
                long start = Stopwatch.GetTimestamp();
                CocktailSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("CocktailSort: Losowy: {0}; {1}", i, stop -
start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabVShape(i);
                long start = Stopwatch.GetTimestamp();
                CocktailSort(tablica);
                long stop = Stopwatch.GetTimestamp();
```

```
                Console.WriteLine("CocktailSort: V Ksztaltny: {0}; {1}", i, stop
- start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabConst(i);
                long start = Stopwatch.GetTimestamp();
                CocktailSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("CocktailSort: Staly: {0}; {1}", i, stop -
start);
            }
            // SelectionSort
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabAscending(i);
                long start = Stopwatch.GetTimestamp();
                SelectionSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("SelectionSort: Rosnacy:{0}; {1}", i, stop -
start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabDescending(i);
                long start = Stopwatch.GetTimestamp();
                SelectionSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("SelectionSort: Malejacy: {0}; {1}", i, stop -
start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabRandom(i);
                long start = Stopwatch.GetTimestamp();
                SelectionSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("SelectionSort: Losowy: {0}; {1}", i, stop -
start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabVShape(i);
                long start = Stopwatch.GetTimestamp();
                SelectionSort(tablica);
                long stop = Stopwatch.GetTimestamp();
```

```
                Console.WriteLine("SelectionSort: V Ksztaltny: {0}; {1}", i, stop
- start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabConst(i);
                long start = Stopwatch.GetTimestamp();
                SelectionSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("SelectionSort: Staly: {0}; {1}", i, stop -
start);
            }
            // HeapSort
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabAscending(i);
                long start = Stopwatch.GetTimestamp();
                HeapSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("HeapSort: Rosnacy:{0}; {1}", i, stop - start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabDescending(i);
                long start = Stopwatch.GetTimestamp();
                HeapSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("HeapSort: Malejacy: {0}; {1}", i, stop -
start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabRandom(i);
                long start = Stopwatch.GetTimestamp();
                HeapSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("HeapSort: Losowy: {0}; {1}", i, stop - start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabVShape(i);
                long start = Stopwatch.GetTimestamp();
                HeapSort(tablica);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("HeapSort: V Ksztaltny: {0}; {1}", i, stop -
start);
```

```
    }
    for (int i = 50000; i <= 200000; i += 10000)
    {
        int[] tablica = tabConst(i);
        long start = Stopwatch.GetTimestamp();
        HeapSort(tablica);
        long stop = Stopwatch.GetTimestamp();
        Console.WriteLine("HeapSort: Staly: {0}; {1}", i, stop - start);
    }
    Console.ReadKey();
}
// GENERATORY TABLIC
public static int[] tabAscending(int rozmiar)
{
    int[] tab = new int[rozmiar];
    for (int i = 0; i < rozmiar; i++)
    {
        tab[i] = i + 1;
    }
    return tab;
}
public static int[] tabDescending(int rozmiar)
{
    int[] tab = new int[rozmiar];
    for (int i = 0; i < rozmiar; i++)
    {
        tab[i] = rozmiar - i;
    }
    return tab;
}
public static int[] tabRandom(int rozmiar)
{
    int[] tablica = new int[rozmiar];
    Random rand = new Random();
    for (int i = 0; i < rozmiar; i++)
    {
        tablica[i] = rand.Next(1, rozmiar + 1);
    }
    return tablica;
}
public static int[] tabVShape(int rozmiar)
{
    int[] tablica = new int[rozmiar];
    int srodek = (int)rozmiar / 2;
    for (int i = 0; i < srodek; i++)
    {
```

```
            tablica[i] = (srodek - i) * 2;
        }
        for (int i = srodek; i < rozmiar; i++)
        {
            tablica[i] = (i - srodek) * 2 + 1;
        }
        return tablica;
    }
    public static int[] tabConst(int rozmiar)
    {
        int[] tab = new int[rozmiar];
        for (int i = 0; i < rozmiar; i++)
        {
            tab[i] = 23;
        }
        return tab;
    }
    // ALGORYTMY SORTOWANIA
    public static void InsertionSort(int[] arr)
    {
        for (int i = 1; i < arr.Length; i++)
        {
            int j = i;
            int tmp = arr[j];
            while ((j > 0) && (arr[j - 1] > tmp))
            {
                arr[j] = arr[j - 1];
                j--;
            }
            arr[j] = tmp;
        }
    }
    public static void CocktailSort(int[] arr)
    {
        int left = 1, right = arr.Length - 1, k = arr.Length - 1;
        do
        {
            for (int j = right; j >= left; j--)
            {
                if (arr[j - 1] > arr[j])
                {
                    int tmp = arr[j - 1];
                    arr[j - 1] = arr[j];
                    arr[j] = tmp;
                    k = j;
                }
```

```
        }
        left = k + 1;
        for (int j = left; j <= right; j++)
        {
            if (arr[j - 1] > arr[j])
            {
                int tmp = arr[j - 1];
                arr[j - 1] = arr[j];
                arr[j] = tmp;
                k = j;
            }
        }
        right = k - 1;
    } while (left <= right);
}
public static void SelectionSort(int[] arr)
{
    int k;
    for (int i = 0; i < (arr.Length - 1); i++)
    {
        int tmp = arr[i];
        k = i;
        for (int j = i + 1; j < arr.Length; j++)
            if (arr[j] < tmp)
            {
                k = j;
                tmp = arr[j];
            }
        arr[k] = arr[i];
        arr[i] = tmp;
    }
}
public static void Heapify(int[] t, int left, int right)
{
    int 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;
        }
        public static void HeapSort(int[] arr)
        {
            int left = (int)arr.Length / 2;
            int right = (int)arr.Length - 1;
            while (left > 0)
            {
                left--;
                Heapify(arr, left, right);
            }
            while (right > 0)
            {
                int buf = arr[left];
                arr[left] = arr[right];
                arr[right] = buf;
                right--;
                Heapify(arr, left, right);
            }
        }
    }
}
```

Kod dla III części projektu:

```
using
System;
          using System.Diagnostics;
          using System.Threading;
          namespace QuickSort
          {
              class Program
              {
                  static void Tester()
                  {
                      for (int i = 50000; i <= 200000; i += 10000)
                      {
                          int[] tablica = tabRandom(i);
                          long start = Stopwatch.GetTimestamp();
                          QuickSortIteration(tablica);
                          long stop = Stopwatch.GetTimestamp();
                          Console.WriteLine("QuickSortIteration Losowy {0} {1}", i, stop -
          start);
                      }
                      for (int i = 50000; i <= 200000; i += 10000)
                      {
                          int[] tablica = tabRandom(i);
                          long start = Stopwatch.GetTimestamp();
                          QuickSortRecursion(tablica, 0, tablica.Length - 1);
                          long stop = Stopwatch.GetTimestamp();
                          Console.WriteLine("QuickSortRecursion Losowy {0} {1}", i, stop -
          start);
                      }
                      for (int i = 50000; i <= 200000; i += 10000)
                      {
                          int[] tablica = tabAShape(i);
                          long start = Stopwatch.GetTimestamp();
                          QuickSortRecursionRandomPivot(tablica, 0, tablica.Length - 1);
                          long stop = Stopwatch.GetTimestamp();
                          Console.WriteLine("RadnomPivot Aksztaltna {0} {1}", i, stop -
          start);
                      }
                      for (int i = 50000; i <= 200000; i += 10000)
                      {
                          int[] tablica = tabAShape(i);
                          long start = Stopwatch.GetTimestamp();
                          QuickSortRecursionLastPivot(tablica, 0, tablica.Length - 1);
                          long stop = Stopwatch.GetTimestamp();
                          Console.WriteLine("LastPivot Aksztaltna {0} {1}", i, stop -
```

```
start);
            }
            for (int i = 50000; i <= 200000; i += 10000)
            {
                int[] tablica = tabAShape(i);
                long start = Stopwatch.GetTimestamp();
                QuickSortRecursion(tablica, 0, tablica.Length - 1);
                long stop = Stopwatch.GetTimestamp();
                Console.WriteLine("MiddlePivot Aksztaltna {0} {1}", i, stop -
start);
            }
        }
        static void Main(string[] args)
        {
            Thread TesterThread = new Thread(Program.Tester, 8 * 1024 * 1024); //
utworzenie wątku
            TesterThread.Start(); // uruchomienie wątku
            TesterThread.Join(); // oczekiwanie na zakończenie wątku
        }
        // Tablice
        public static int[] tabAShape (int size)
        {
            int[] arr = new int[size];
            int center = (int)size / 2;
            for (int i = 0; i < center; i++)
            {
                arr[i] = i * 2 + 2;
            }
            for (int i = center; i < size; i++)
            {
                arr[i] = (size - i) * 2 - 1;
            }
            return arr;
        }
        public static int[] tabRandom(int rozmiar)
        {
            int[] tablica = new int[rozmiar];
            Random rand = new Random();
            for (int i = 0; i < rozmiar; i++)
            {
                tablica[i] = rand.Next(1, rozmiar + 1);
            }
            return tablica;
        }
        // Algorytmy Sortowania
        // sorotwanie szybkie iterracyjne
```

```
        public static void QuickSortIteration(int[] t)
        {
            int i, j, l, p, sp;
            int[] stos_l = new int[t.Length],
            stos_p = new int[t.Length]; // przechowywanie żądań podziału
            sp = 0;
            stos_l[sp] = 0;
            stos_p[sp] = t.Length - 1; // rozpoczynamy od całej tablicy
            do
            {
                l = stos_l[sp]; p = stos_p[sp]; sp--; // pobieramy żądanie
podziału
                do
                {
                    int x;
                    i = l; j = p; x = t[(l + p) / 2]; // analogicznie do wersji
rekurencyjnej
                    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; } //
ewentualnie dodajemy żądanie podziału
                    p = j;
                } while (l < p);
            } while (sp >= 0); // dopóki stos żądań nie będzie pusty
        }
        // Sortowanie szybkie rekurencyjne srodkowe polozenie pivota
        public static void QuickSortRecursion(int[] t, int l, int p)
        {
            int i, j, x;
            i = l;
            j = p;
            x = t[(l + p) / 2]; // (pseudo)mediana
            do
            {
                while (t[i] < x) i++; // przesuwamy indeksy z lewej
                while (x < t[j]) j--; // przesuwamy indeksy z prawej
                if (i <= j) // jeśli nie minęliśmy się indeksami (koniec kroku)
                { // zamieniamy elementy
```

```
                int buf = t[i]; t[i] = t[j]; t[j] = buf;
                i++; j--;
            }
        }
        while (i <= j);
        if (l < j) QuickSortRecursion(t, l, j); // sortujemy lewą część
(jeśli jest)
        if (i < p) QuickSortRecursion(t, i, p); // sortujemy prawą część
(jeśli jest)
    }
    // Sortowanie szybkie pivot na ostatnim miejscu
    public static void QuickSortRecursionLastPivot(int[] t, int l, int p)
    {
        int i, j, x;
        i = l;
        j = p;
        x = t[p]; // (pseudo)mediana
        do
        {
            while (t[i] < x) i++; // przesuwamy indeksy z lewej
            while (x < t[j]) j--; // przesuwamy indeksy z prawej
            if (i <= j) // jeśli nie minęliśmy się indeksami (koniec kroku)
            { // zamieniamy elementy
                int buf = t[i]; t[i] = t[j]; t[j] = buf;
                i++; j--;
            }
        }
        while (i <= j);
        if (l < j) QuickSortRecursion(t, l, j); // sortujemy lewą część
(jeśli jest)
        if (i < p) QuickSortRecursion(t, i, p); // sortujemy prawą część
(jeśli jest)
    }
    // Sortowanie szybkie pivot losowy
    public static void QuickSortRecursionRandomPivot(int[] t, int l, int p)
    {
        int i, j, x;
        i = l;
        j = p;
        x = t[new Random().Next(l, p)]; // (pseudo)mediana
        do
        {
            while (t[i] < x) i++; // przesuwamy indeksy z lewej
            while (x < t[j]) j--; // przesuwamy indeksy z prawej
            if (i <= j) // jeśli nie minęliśmy się indeksami (koniec kroku)
            { // zamieniamy elementy
```

```
                int buf = t[i]; t[i] = t[j]; t[j] = buf;
                i++; j--;
            }
        }
        while (i <= j);
        if (l < j) QuickSortRecursion(t, l, j); // sortujemy lewą część
(jeśli jest)
        if (i < p) QuickSortRecursion(t, i, p); // sortujemy prawą część
(jeśli jest)
    }
}
```