# Raport: Sortowanie

## Bartłomiej Kręgielewski – WDI Grupa 3A – 13:50 Poniedziałek

Kod Programu:

```
import random
import time

numberOfSort = 8
randRange = 10001
numberOfElements = 10
numberOfTests = 10
step = 1000
displayResults = 1

def fill(tab, size):
    for i in range(0, size):
        tab.append(random.randrange(1, randRange))

def selectionSort(tab):
    for i in range(0, len(tab)):
        minimum = i
        for j in range(i, len(tab)):
            if (tab[j] < tab[minimum]):
                minimum = j
        tmp = tab[i]
        tab[i] = tab[minimum]
        tab[minimum] = tmp

def insertionSort(tab):
    for i in range(1, len(tab)):
            if (tab[i] < tab[i - 1]):
                    for j in range(i - 1, -1, -1):
                            if (tab[j] < tab[i]):
                                    tab.insert(j+1, tab.pop(i))
                                    break
                            elif (j == 0):
                                    tab.insert(j, tab.pop(i))

def bubbleSort(tab):
    for i in range(0, len(tab)):
            for j in range(0, len(tab) - i - 1):
                    if (tab[j] > tab[j + 1]):
                            tab[j], tab[j+1] = tab[j+1], tab[j]


def partition(tab, s, f):
```

```
        counter = s
        for i in range(s+1, f+1):
                if (tab[counter] > tab[i]):
                        tab.insert(counter, tab.pop(i))
                        counter += 1
        return counter

def quickSort(tab, s, f):
        if (s < f):
                p = partition(tab, s, f)
                quickSort(tab, s, p-1)
                quickSort(tab, p+1, f)

def mergeSort(tab):
    if (len(tab) > 1):
        middle = len(tab)//2
        L = tab[:middle]
        R = tab[middle:]
        mergeSort(L)
        mergeSort(R)
        i = 0
        j = 0
        k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                tab[k] = L[i]
                i+=1
            else:
                tab[k] = R[j]
                j+=1
            k+=1
        while i < len(L):
            tab[k] = L[i]
            i+=1
            k+=1
        while j < len(R):
            tab[k] = R[j]
            j+=1
            k+=1

def makeHeap(tab, n, i):
        largest = i
        l = 2 * i + 1
        r = 2 * i + 2
        if l < n and tab[i] < tab[l]:
                largest = l
        if r < n and tab[largest] < tab[r]:
                largest = r
```

```
        if largest != i:
                tab[i],tab[largest] = tab[largest],tab[i]
                makeHeap(tab, n, largest)

def heapSort(tab):
        length = len(tab)
        for i in range(length, -1, -1):
                makeHeap(tab, length, i)
        for i in range(length-1, 0, -1):
                tab[i], tab[0] = tab[0], tab[i]
                makeHeap(tab, i, 0)

def bucketSort(tab, min, max):
        buckets = []
        numberOfBuckets = int(len(tab)/2)
        if (numberOfBuckets == 0):
                numberOfBuckets = 1
        delta = ((max - min)/numberOfBuckets)
        for i in range(0, numberOfBuckets):
                buckets.append([])
        for i in range(0, len(tab)):
                buckets[int((tab[i] - min)/delta)].append(tab[i])
        tab.clear()
        for i in range(0, numberOfBuckets):
                bubbleSort(buckets[i])
                tab += buckets[i]

def radixCompare(tab, position):
    n = len(tab)
    output = [0] * (n)
    count = [0] * (10)
    for i in range(0, n):
        index = (tab[i]/position)
        count[ int((index)%10) ] += 1
    for i in range(1,10):
        count[i] += count[i-1]
    i = n-1
    while i>=0:
        index = (tab[i]/position)
        output[ count[ int((index)%10) ] - 1] = tab[i]
        count[ int((index)%10) ] -= 1
        i -= 1
    i = 0
    for i in range(0,len(tab)):
        tab[i] = output[i]

def radixSort(tab):
    maximum = max(tab)
```

```
    position = 1
    while maximum/position > 0:
        radixCompare(tab,position)
        position *= 10

#####################################
numberOfElements = int(input("Choose number of elements in array at the
beginning: "))
while (numberOfElements < 1):
        numberOfElements = int(input("You must choose not less than 1\nChose
number of elements in array at the beginning: "))
numberOfTests = int(input("Choose number of tests: "))
while (numberOfTests < 1):
        numberOfTests = int(input("You must choose not less than 1\nChose
number of tests: "))
step = int(input("Choose step: "))
while (step < 0):
        step = int(input("You must choose not less than 0\nChoose step: "))
displayResults = int(input("Display results?\n1.Yes\n2.No\n"))
while (displayResults != 1 and displayResults != 2):
        displayResults = int(input("Select number\nDisplay
results?\n1.Yes\n2.No\n"))
results = []
for i in range(0, numberOfSort):
        results.append([])
testingStart = time.time()
for i in range(0, numberOfTests):
        T = []
        fill(T, numberOfElements + i * step)
        Tabs = []
        for i in range(0, numberOfSort):
                Tabs.append(T.copy())

        sortStart = time.time()
        bubbleSort(Tabs[0])
        results[0].append(time.time() - sortStart)

        sortStart = time.time()
        insertionSort(Tabs[1])
        results[1].append(time.time() - sortStart)

        sortStart = time.time()
        selectionSort(Tabs[2])
        results[2].append(time.time() - sortStart)

        sortStart = time.time()
        quickSort(Tabs[3],0,len(Tabs[3]) - 1)
        results[3].append(time.time() - sortStart)
```

```
        sortStart = time.time()
        mergeSort(Tabs[4])
        results[4].append(time.time() - sortStart)

        sortStart = time.time()
        heapSort(Tabs[5])
        results[5].append(time.time() - sortStart)

        sortStart = time.time()
        bucketSort(Tabs[6], 0, randRange)
        results[6].append(time.time() - sortStart)

        sortStart = time.time()
        radixSort(Tabs[7])
        results[7].append(time.time() - sortStart)

        if (displayResults == 1):
                print("\nTest number #" + str(i))
                print("Original Tab")
                print(T)
                print("Sorted Tabs")
                for i in range(0, numberOfSort):
                        print(Tabs[i])
                print()

print("Time for all tests: " + str(time.time() - testingStart))

f = open("bubbleSort.txt","w+")
for i in range(0, len(results[0])):
        f.write(str(results[0][i]) + "\t" + str(numberOfElements + i * step) +
"\n")
f.close()

f = open("insertionSort.txt","w+")
for i in range(0, len(results[1])):
        f.write(str(results[1][i]) + "\t" + str(numberOfElements + i * step) +
"\n")
f.close()

f = open("selectionSort.txt","w+")
for i in range(0, len(results[2])):
        f.write(str(results[2][i]) + "\t" + str(numberOfElements + i * step) +
"\n")
f.close()

f = open("quickSort.txt","w+")
for i in range(0, len(results[3])):
```

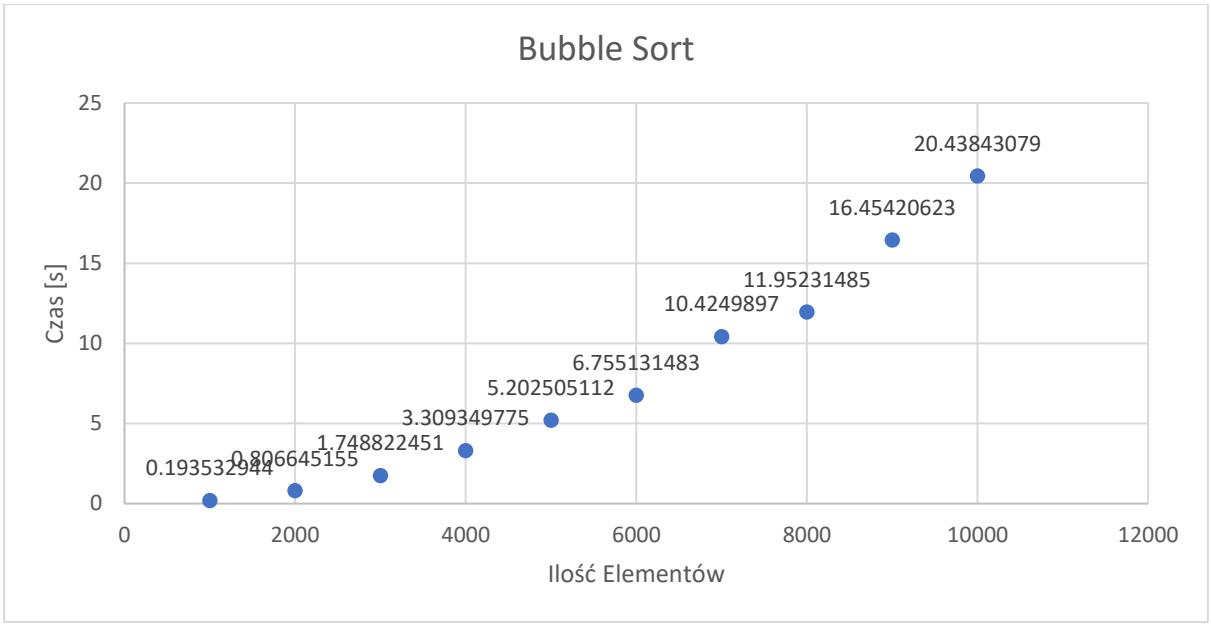
```
        f.write(str(results[3][i]) + "\t" + str(numberOfElements + i * step) +
"\n")
f.close()

f = open("mergeSort.txt","w+")
for i in range(0, len(results[4])):
        f.write(str(results[4][i]) + "\t" + str(numberOfElements + i * step) +
"\n")
f.close()

f = open("heapSort.txt","w+")
for i in range(0, len(results[5])):
        f.write(str(results[5][i]) + "\t" + str(numberOfElements + i * step) +
"\n")
f.close()

f = open("bucketSort.txt","w+")
for i in range(0, len(results[6])):
        f.write(str(results[6][i]) + "\t" + str(numberOfElements + i * step) +
"\n")
f.close()

f = open("radixtSort.txt","w+")
for i in range(0, len(results[7])):
        f.write(str(results[7][i]) + "\t" + str(numberOfElements + i * step) +
"\n")
f.close()
#######################################
```

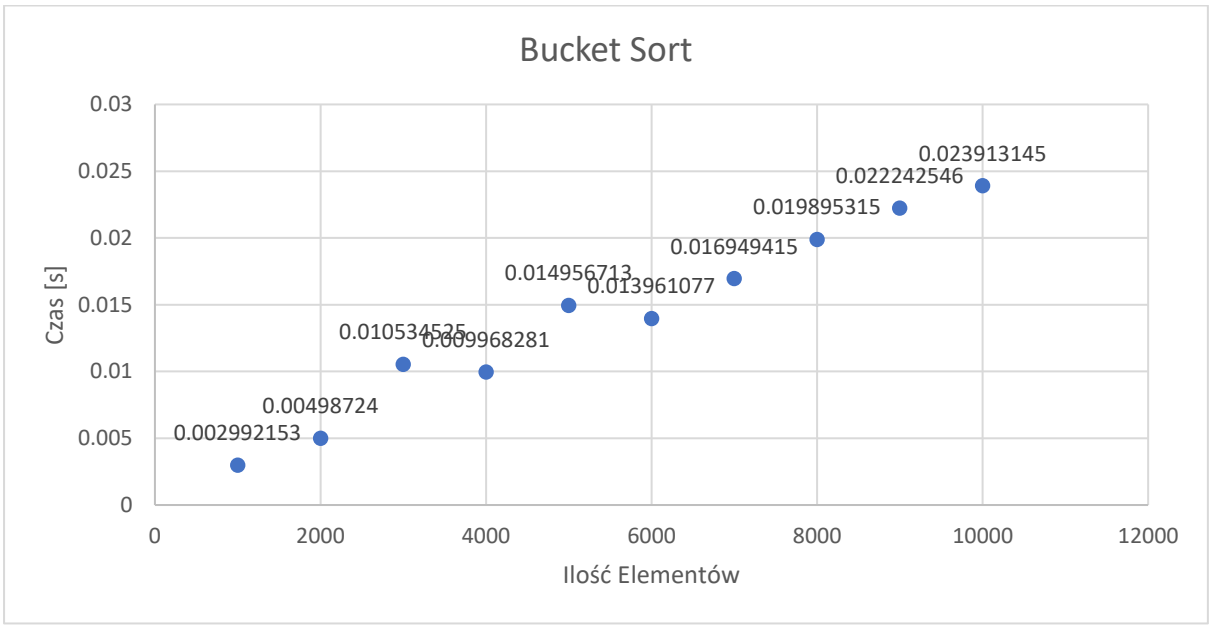
Uwaga: Program automatycznie utworzy 8 plików tekstowych z zapisanymi czasami wykonywania się sortowań

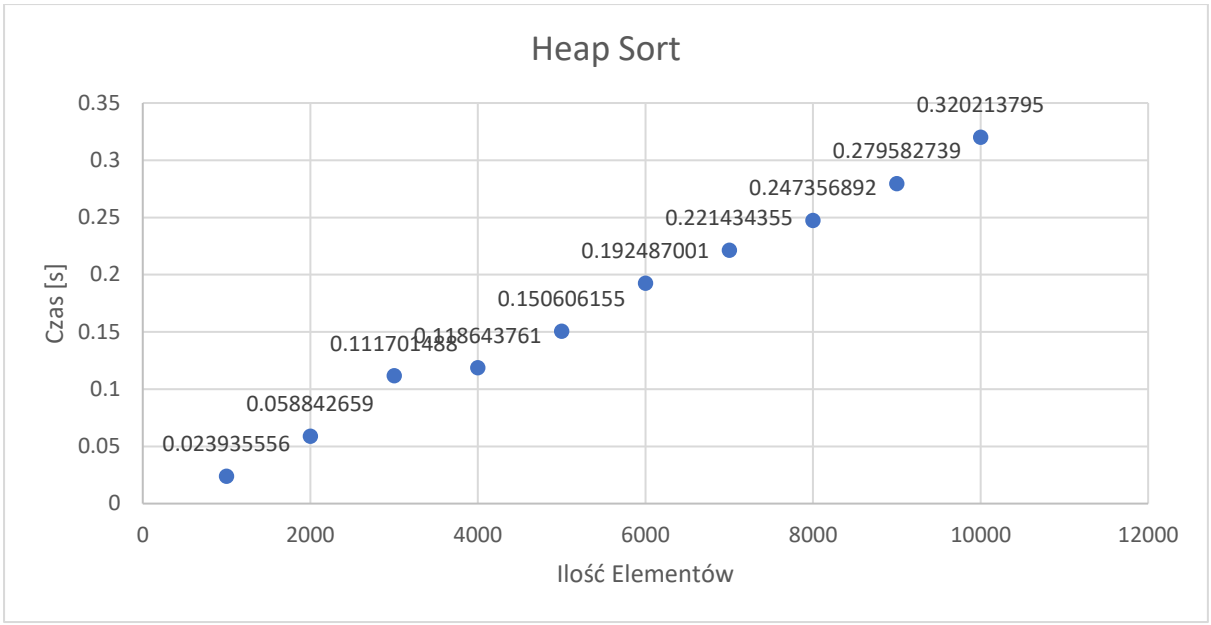
# Zestawienie wyników

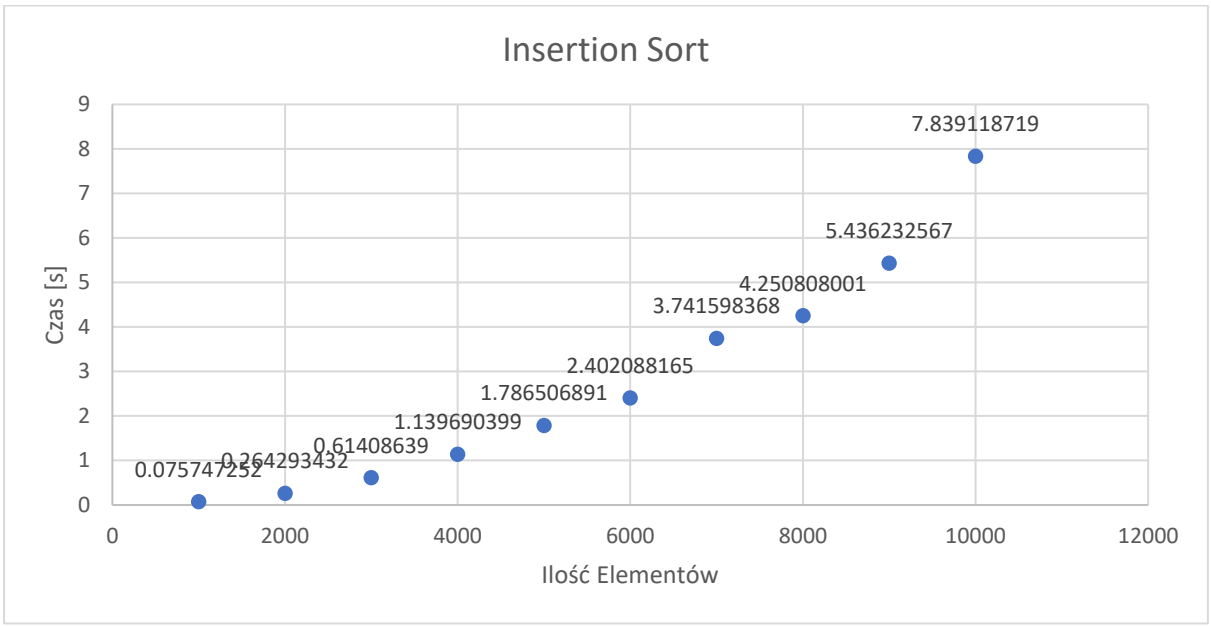
| Liczba Elementów | BubbleSort | BucketSort | HeapSort | InsertionSort | MergeSort | QuickSort | RadixSort | SelectionSort |
|---|---|---|---|---|---|---|---|---|
| 1000 | 0.193533 | 0.002992 | 0.023936 | 0.075747 | 0.012019 | 0.009939 | 0.791961 | 0.092786 |
| 2000 | 0.806645 | 0.004987 | 0.058843 | 0.264293 | 0.02992 | 0.022939 | 1.895026 | 0.378537 |
| 3000 | 1.748822 | 0.010535 | 0.111701 | 0.614086 | 0.041921 | 0.04439 | 2.74897 | 0.935005 |
| 4000 | 3.30935 | 0.009968 | 0.118644 | 1.13969 | 0.05097 | 0.054886 | 3.476905 | 1.515785 |
| 5000 | 5.202505 | 0.014957 | 0.150606 | 1.786507 | 0.069821 | 0.086101 | 4.162771 | 2.46198 |
| 6000 | 6.755131 | 0.013961 | 0.192487 | 2.402088 | 0.084748 | 0.118673 | 5.300311 | 3.312725 |
| 7000 | 10.42499 | 0.016949 | 0.221434 | 3.741598 | 0.102205 | 0.151633 | 5.677788 | 4.802887 |
| 8000 | 11.95231 | 0.019895 | 0.247357 | 4.250808 | 0.113729 | 0.196472 | 7.085837 | 5.744335 |
| 9000 | 16.45421 | 0.022243 | 0.279583 | 5.436233 | 0.130776 | 0.247324 | 7.510419 | 7.334436 |
| 10000 | 20.43843 | 0.023913 | 0.320214 | 7.839119 | 0.143614 | 0.275434 | 8.546176 | 10.14972 |

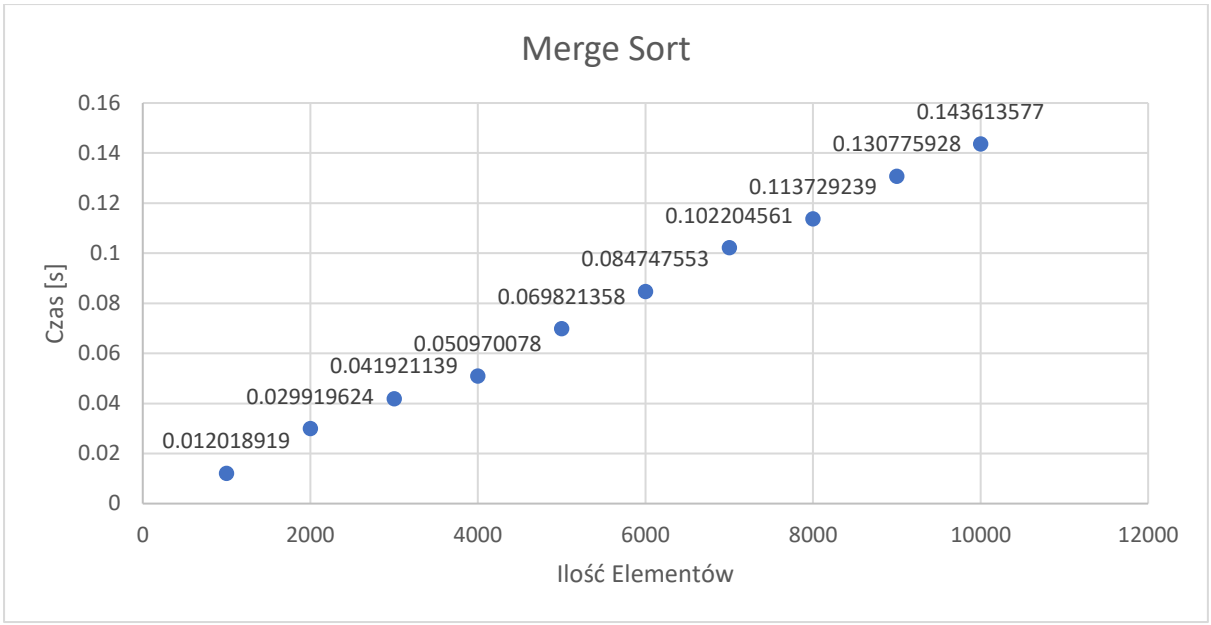
Czasy wykonywania się wszystkich testów: 192.96452689170837 sekund

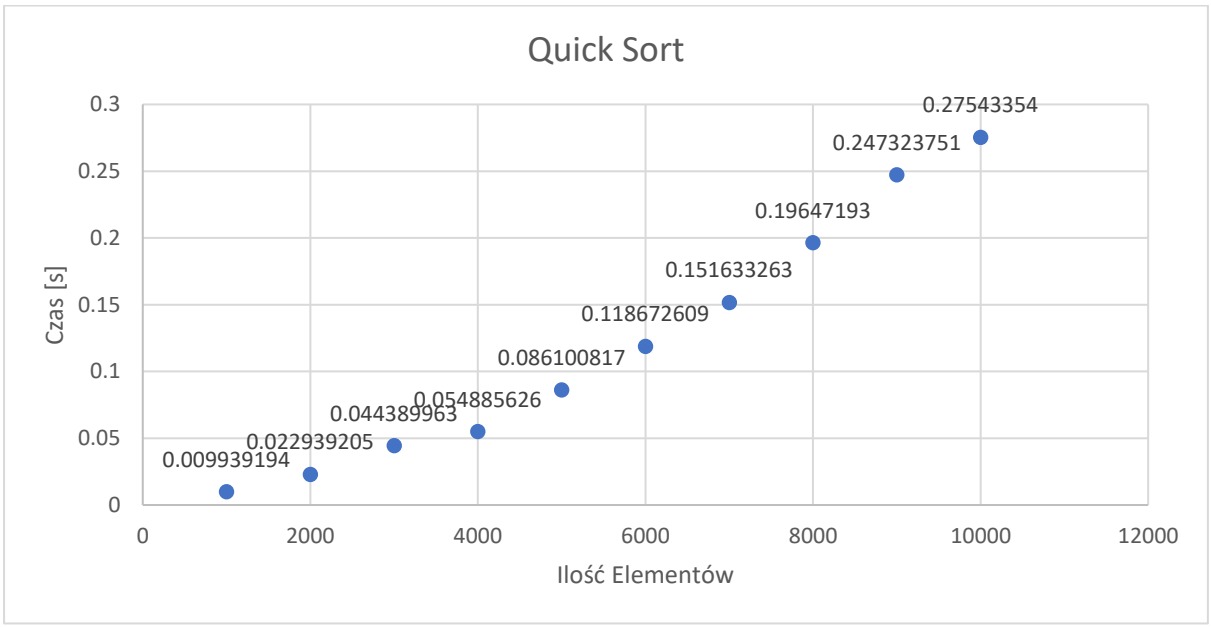

Bubble Sort
Czas [s]
Ilość Elementów
0
5
10
15
20
25
0
2000
4000
6000
8000
10000
12000
20.43843079
16.45420623
11.95231485
10.4249897
6.755131483
5.202505112
3.309349775
1.748822451
0.806645155
0.193532944

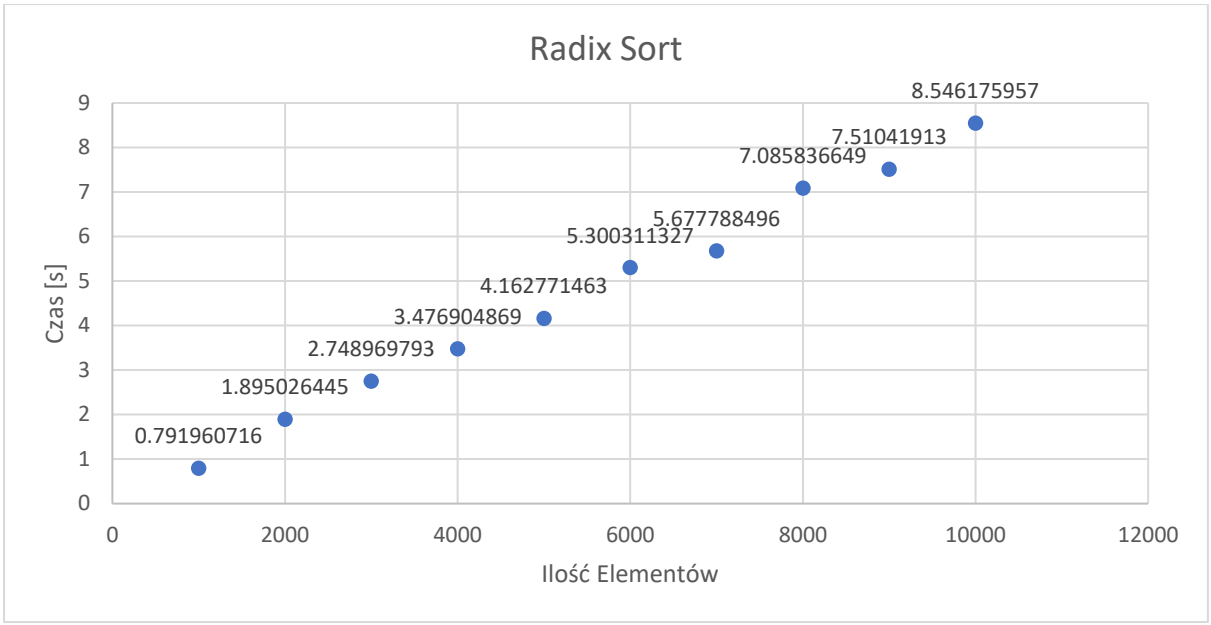

Bucket Sort
Czas [s]
Ilość Elementów
0
0.005
0.01
0.015
0.02
0.025
0.03
0
2000
4000
6000
8000
10000
12000
0.023913145
0.022242546
0.019895315
0.016949415
0.014956713
0.013961077
0.010534525
0.009968281
0.00498724
0.002992153

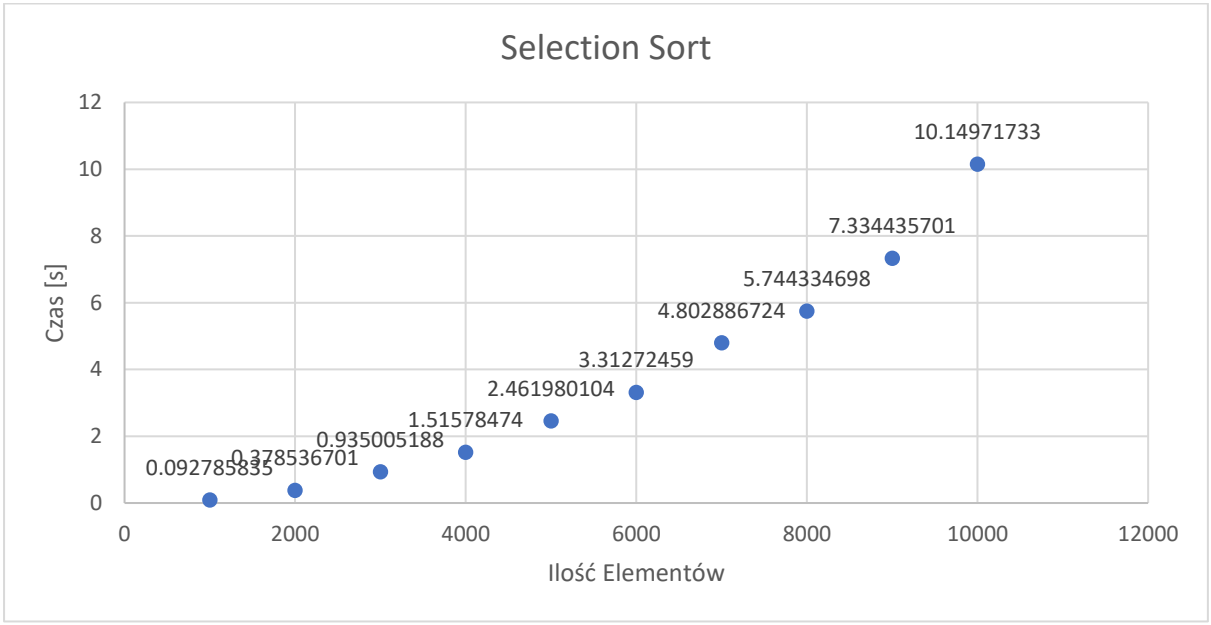

Heap Sort
Czas [s]
Ilość Elementów
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
2000
4000
6000
8000
10000
12000
0.320213795
0.279582739
0.247356892
0.221434355
0.192487001
0.150606155
0.118643761
0.111701488
0.058842659
0.023935556


Insertion Sort
Czas [s]
Ilość Elementów
9
8
7
6
5
4
3
2
1
0
0
2000
4000
6000
8000
10000
12000
7.839118719
5.436232567
4.250808001
3.741598368
2.402088165
1.786506891
1.139690399
0.61408639
0.264293432
0.075747252


Merge Sort
Czas [s]
Ilość Elementów
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
2000
4000
6000
8000
10000
12000
0.143613577
0.130775928
0.113729239
0.102204561
0.084747553
0.069821358
0.050970078
0.041921139
0.029919624
0.012018919


Quick Sort
Czas [s]
Ilość Elementów
0.3
0.25
0.2
0.15
0.1
0.05
0
0
2000
4000
6000
8000
10000
12000
0.27543354
0.247323751
0.19647193
0.151633263
0.118672609
0.086100817
0.054885626
0.044389963
0.022939205
0.009939194


Radix Sort
Czas [s]
Ilość Elementów
0
1
2
3
4
5
6
7
8
9
0
2000
4000
6000
8000
10000
12000
0.791960716
1.895026445
2.748969793
3.476904869
4.162771463
5.300311327
5.677788496
7.085836649
7.51041913
8.546175957


Selection Sort
Czas [s]
Ilość Elementów
0
2
4
6
8
10
12
0
2000
4000
6000
8000
10000
12000
0.092785835
0.378536701
0.935005188
1.51578474
2.461980104
3.31272459
4.802886724
5.744334698
7.334435701
10.14971733

Zestawienie algorytmów ze sobą (punkty połączono aby zwiększyć czytelność)