

Algorytmy i struktury danych

Sprawozdanie z zadania w zespołach nr. 1  
prowadząca: dr hab. inż. Małgorzata Sterna, prof PP

## Algorytmy sortujące

autorzy:

Piotr Więtczak nr indeksu 132339,  
Tomasz Chudziak nr indeksu 136691

24 marca 2018

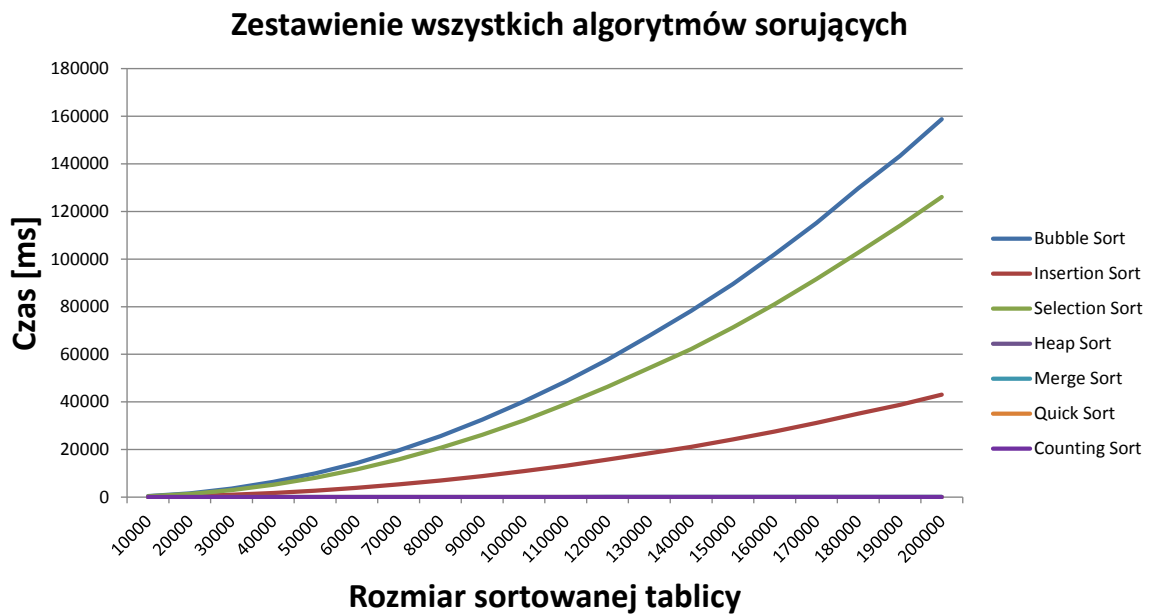
# 1 Implementacja algorytmów sortujących

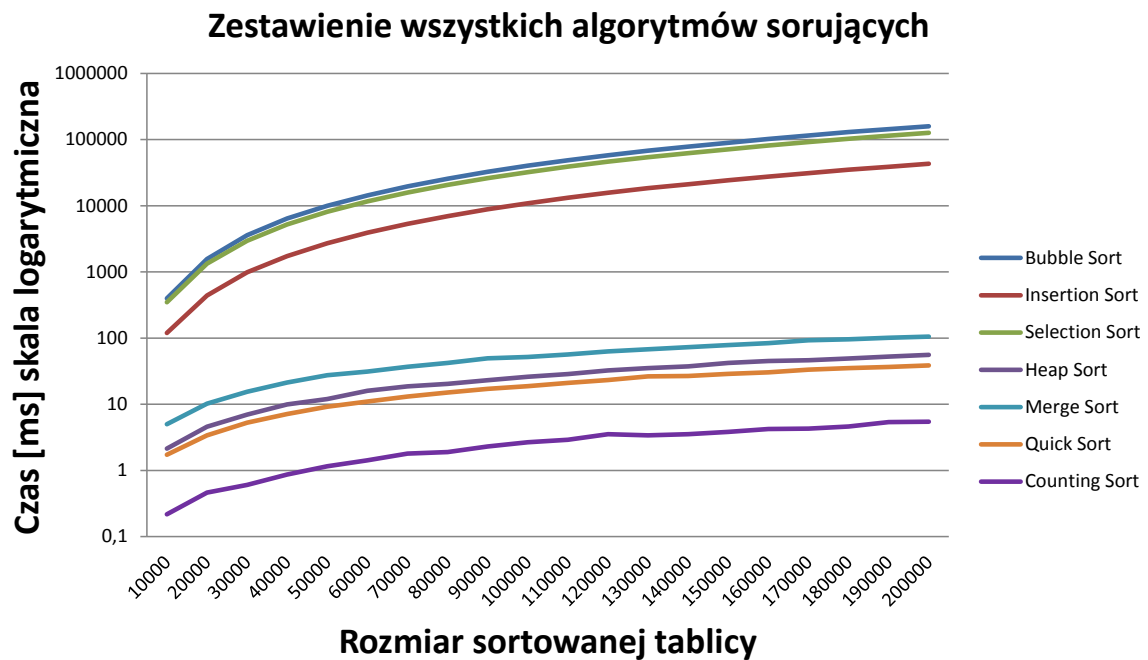
Do implementacji metod sortowania posłużyliśmy się językiem `C++`, każda metoda została napisana w odrębnej funkcji, która za parametry przyjmuje kolejno: wskaźnik na tablicę, rozmiar sortowanej tablicy oraz jako ostatni wartość opcjonalną "reverse" typu `bool`, która odpowiada za to czy tablica będzie posortowana rosnąco czy malejąco. Do mierzenia czasu poszczególnych metod użyliśmy klasy `std::chrono::high_resolution_clock` z biblioteki `chrono`.

## 2 Badana zależność czasu obliczeń $t[s]$ od liczby sortowanych elementów $n$ .

### 2.1 Podział metod sortowania

W celu zachowania przejrzystości otrzymanych danych podzieliliśmy metody na dwie grupy, "wolne" (Insertion Sort, Selection Sort, Bubble Sort) i "szybkie" (Counting Sort, Quick Sort, Merge Sort, Heap Sort). Różnice w zależności czasu obliczeń  $t[s]$  od liczby sortowanych elementów  $n$  dla algorytmów "wolnych" i "szybkich" przedstawiają poniższe wykresy.





## Wnioski do podziału metod sortowania

Jak widać na [wykresie](#) przedstawiającym zależność czasu obliczeń od liczby sortowanych elementów, linie przedstawiające metody "szybkie" zlewają się ze sobą i leżą przy samej osi OX, [wykres](#) pokazuje także jak znacząca jest różnica szybkości wykonywania sortowań między grupami. Dopiero przedstawienie danych na [wykresie](#) ze skalą logarytmiczną pozwala rozróżnić metody "szybkie".

## 2.2 Metody "wolne"

### 2.2.1 Opis algorytmów "wolnych"

#### Insert Sort

Zalety:

- działa w miejscu
- stabilny

Wady:

- wolniejszy od metod "szybkich"
- mało wydajne dla dużej ilości elementów do posortowania

Inne cechy:

- zachowanie naturalne

**Tabela przedstawiająca złożoność obliczeniową dla przypadków optymistycznego, średniego i pesymistycznego**

	złożoność obliczeniowa dla przypadku optymistycznego	złożoność obliczeniowa dla przypadku średniego	złożoność obliczeniowa dla przypadku pesymistycznego
Insert Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$

Tablica 1: Tablica złożoności obliczeniowej dla metody Insert Sort

### **Selection Sort**

Zalety:

- działa w miejscu
- stabilny

Wady:

- wolniejszy od metod "szybkich"
- mało wydajne dla dużej ilości elementów do posortowania

Inne cechy:

- zachowanie naturalne

**Tabela przedstawiająca złożoność obliczeniową dla przypadków optymistycznego, średniego i pesymistycznego**

	złożoność obliczeniowa dla przypadku optymistycznego	złożoność obliczeniowa dla przypadku średniego	złożoność obliczeniowa dla przypadku pesymistycznego
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$

Tablica 2: Tablica złożoności obliczeniowej dla metody Selection Sort

### **Bubble Sort**

Zalety:

- działa w miejscu
- stabilny

Wady:

- wolniejszy od metod "szybkich"
- mało wydajne dla dużej ilości elementów do posortowania

Inne cechy:

- zachowanie naturalne

Tabela przedstawiająca złożoność obliczeniową dla przypadków optymistycznego, średniego i pesymistycznego

	złożoność obliczeniowa dla przypadku optymistycznego	złożoność obliczeniowa dla przypadku średniego	złożoność obliczeniowa dla przypadku pesymistycznego
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$

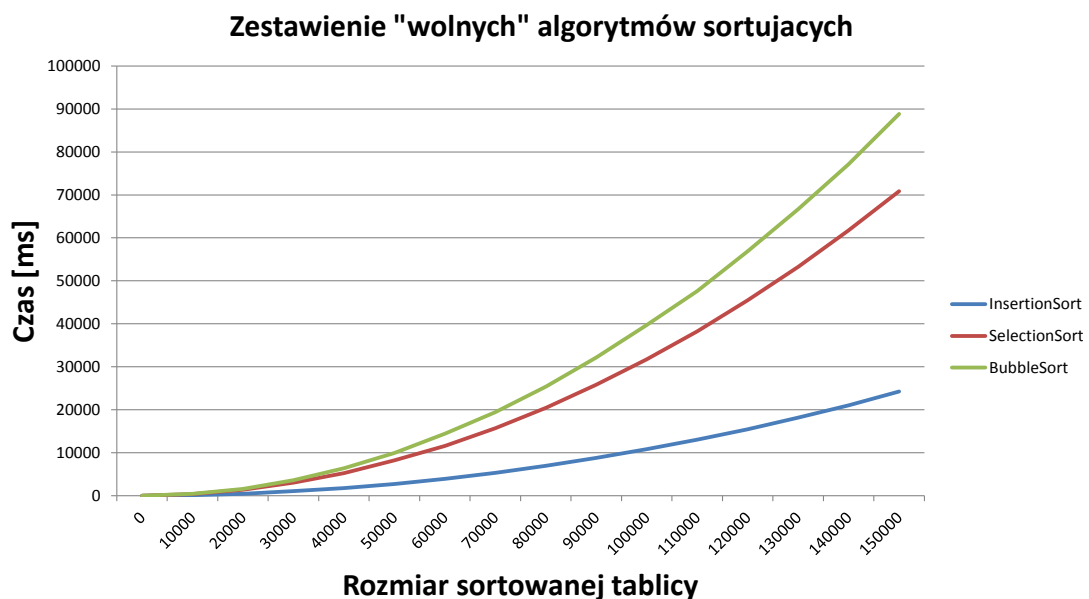
Tablica 3: Tablica złożoności obliczeniowej dla metody Bubble Sort

Tabela ilustrująca zależności czasu sortowania od ilości elementów dla metod "wolnych", zakres liczb  $[1, n]$ .

Liczba elementów	Insertion Sort	Selection Sort	Bubble Sort
------------------	----------------	----------------	-------------

Tablica 4: Wyniki badań zależności czasu od ilości elementów dla metod "wolnych"

Wykres ilustrujący zależności czasu sortowania od ilości elementów dla metod "wolnych", zakres liczb  $[1, n]$ .



### 2.2.2 Wnioski do metod "szybkich"

## 2.3 Metody "szybkie"

### 2.3.1 Opis algorytmów "szybkich"

#### Quick Sort

Zalety:

- działa w miejscu
- stabilny

Wady:

- wrażliwy na dane wejściowe

Inne cechy:

- zachowanie nienaturalne
- korzysta z metody "dziel i rządź"
- algorytm rekurencyjny

**Tabela przedstawiająca złożoność obliczeniową dla przypadków optymistycznego, średniego i pesymistycznego**

	złożoność obliczeniowa dla przypadku optymistycznego	złożoność obliczeniowa dla przypadku średniego	złożoność obliczeniowa dla przypadku pesymistycznego
Quick Sort	$O(n \log_2(n))$	$O(n \log_2(n))$	$O(n^2)$

Tablica 5: Tablica złożoności obliczeniowej dla metody Quick Sort

#### Merge Sort

Zalety:

- algorytm asymptotycznie optymalny

Wady:

- nie działa w miejscu
- wrażliwy na dane wejściowe

Inne cechy:

- zachowanie nienaturalne
- korzysta z metody "dziel i rządź"
- algorytm rekurencyjny

**Tabela przedstawiająca złożoność obliczeniową dla przypadków optymistycznego, średniego i pesymistycznego**

	złożoność obliczeniowa dla przypadku optymistycznego	złożoność obliczeniowa dla przypadku średniego	złożoność obliczeniowa dla przypadku pesymistycznego
Merge Sort	$O(n \log_2(n))$	$O(n \log_2(n))$	$O(n^2)$

Tablica 6: Tablica złożoności obliczeniowej dla metody Merge Sort

## Heap Sort

Zalety:

- działa w miejscu

Wady:

- wrażliwy na dane wejściowe

Inne cechy:

- zachowanie nienaturalne
- korzysta ze stogów

**Tabela przedstawiająca złożoność obliczeniową dla przypadków optymistycznego, średniego i pesymistycznego**

	złożoność obliczeniowa dla przypadku optymistycznego	złożoność obliczeniowa dla przypadku średniego	złożoność obliczeniowa dla przypadku pesymistycznego
Heap Sort	$O(n)$	$kek$	$O(n \log_2(n))$

Tablica 7: Tablica złożoności obliczeniowej dla metody Heap Sort

## Counting Sort

Zalety:

- bardzo szybki dla danych z małego zakresu

Wady:

- nie działa w miejscu
- ograniczony ze względu na zakres sortowanych liczb
- mało wydajny dla danych z dużego przedziału

Inne cechy:

- 

**Tabela przedstawiająca złożoność obliczeniową dla przypadków optymistycznego, średniego i pesymistycznego**

	złożoność obliczeniowa dla przypadku optymistycznego	złożoność obliczeniowa dla przypadku średniego	złożoność obliczeniowa dla przypadku pesymistycznego
Counting Sort	$O(n)$	$O(n)$	$O(n)$

Tablica 8: Tablica złożoności obliczeniowej dla metody Counting Sort

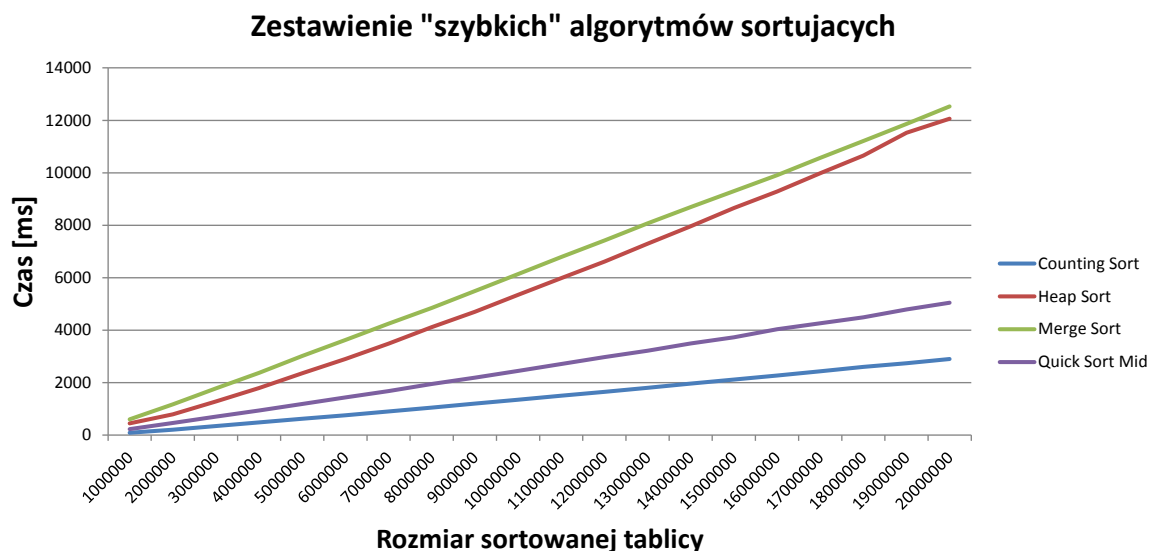
**Tabela ilustrująca zależności czasu sortowania od ilości elementów dla metod "szybkich", zakres liczb  $[1, n]$ .**

Liczba elementów	Quick Sort	Merge Sort	Heap Sort	Counting Sort
------------------	------------	------------	-----------	---------------

Tablica 9: Wyniki badań zależności czasu od ilości elementów dla metod "szybkich"



Wykres ilustrujący zależność czasu sortowania od ilości elementów dla metod "szybkich", zakres liczb  $[1, n]$ .



### 2.3.2 Wnioski do metod "szybkich"

## 2.4 Wnioski do podziału metod sortowania

# 3 Badanie zależności czasu $t$ od liczby sortowanych elementów $n$ , przy rozkładach losowych i rosnących, dla metod Quick Sort z podziałem wg: skrajnego i środkowego elementu, oraz dla metody Insert Sort

W dalszej części sprawozdania algorytm Quick Sort z podziałem według środkowego elementu będziemy nazywać Quick Sort Mid, a z podziałem według skrajnego elementu Quick Sort Right.

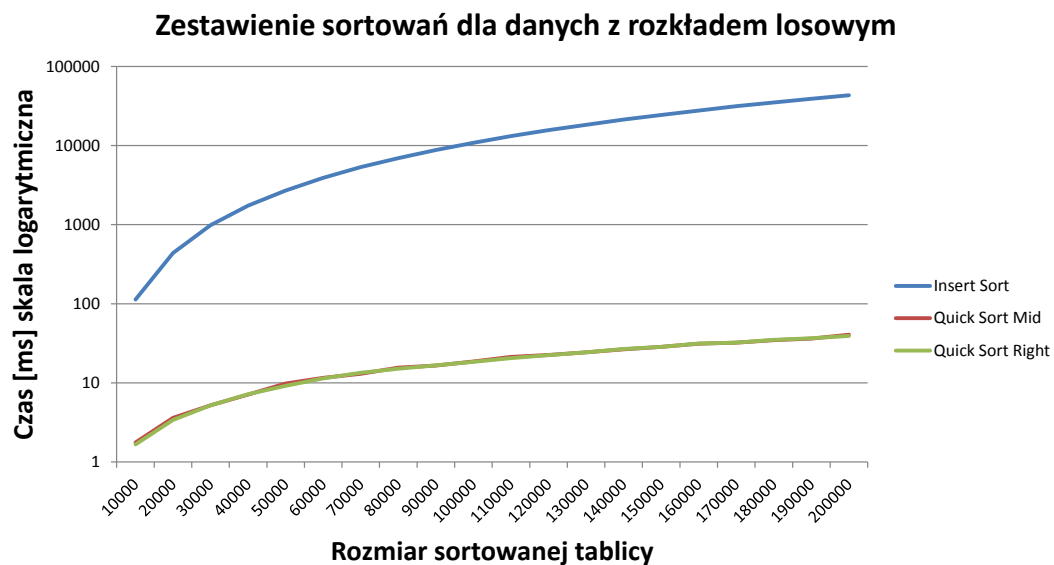
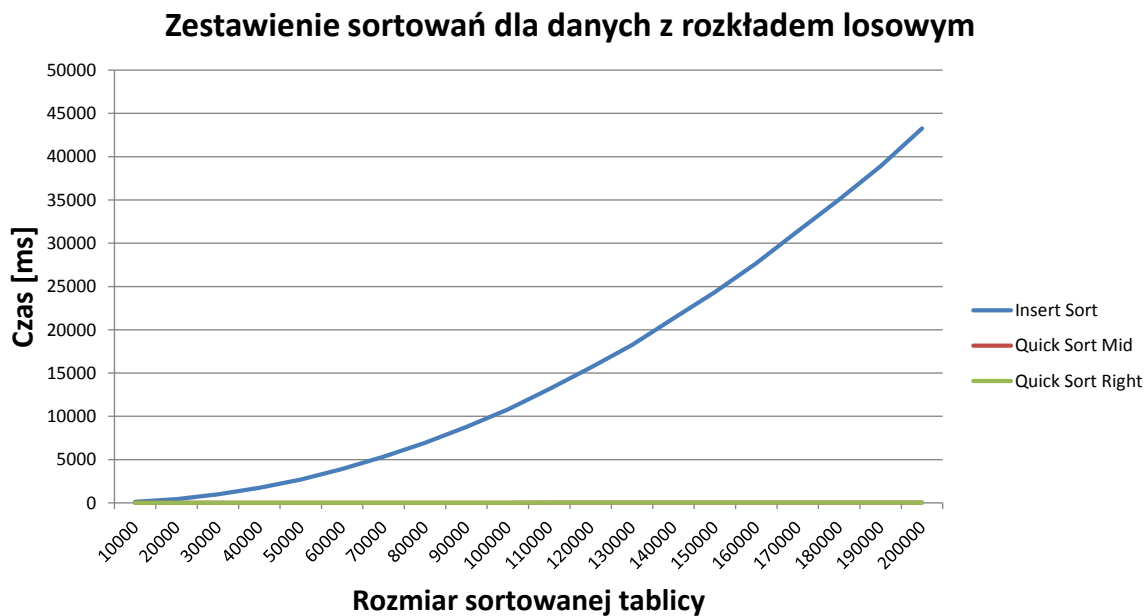
Tabela ilustrująca zależność czasu sortowania od ilości elementów dla metod Quick Sort Mid, Quick Sort Right, Insertion Sort, dla rozkładów losowego i rosnącego.

	Insert Sort	Quick Sort Right	Quick Sort Mid	Insert Sort	Quick Sort Right	Quick Sort Mid
L. ele.	roz. losowy	roz. losowy	roz. losowy	roz. rosnący	roz. rosnący	roz. rosnący

Tablica 10: Wyniki badań zależności czasu od ilości elementów dla metod Quick Sort Mid, Quick Sort Right, Insertion Sort, dla rozkładów losowego i rosnącego.

### 3.1 Rozkład losowy

Wykresy ilustrujące zależności czasu sortowania od ilości elementów dla metod Quick Sort Mid, Quick Sort Right, Insertion Sort, dla rozkładu losowego.

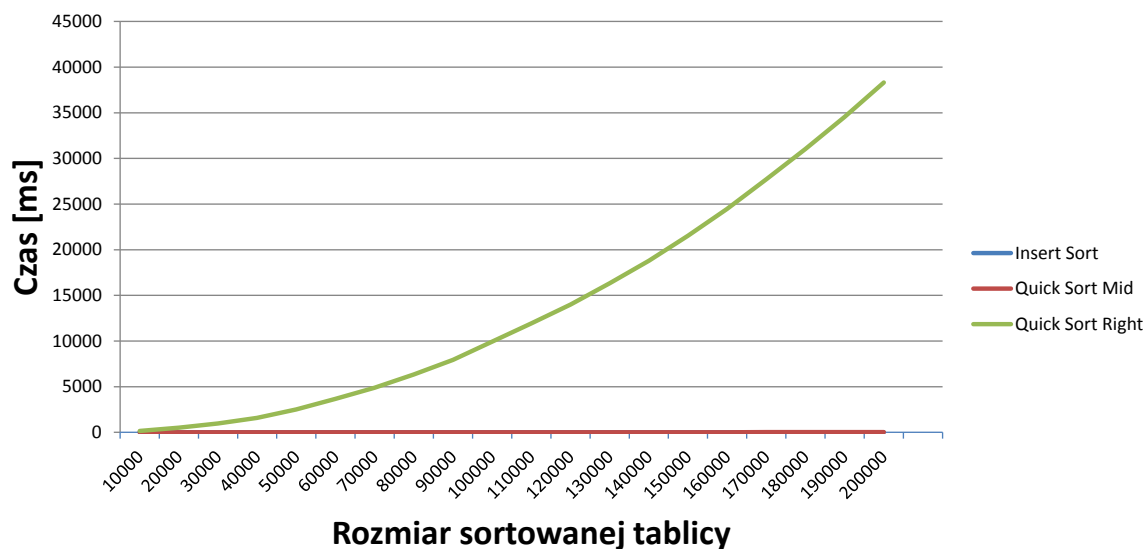


### 3.1.1 Wnioski do rozkładu losowego

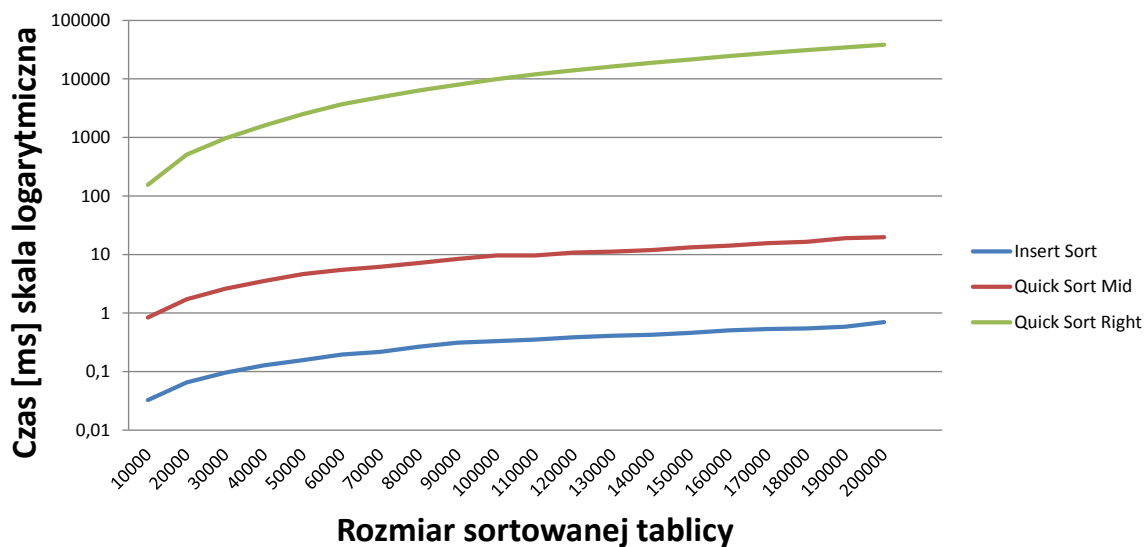
## 3.2 Rozkład rosnący

Wykresy ilustrujące zależności czasu sortowania od ilości elementów dla metod Quick Sort Mid, Quick Sort Right, Insertion Sort, dla rozkładu rosnącego.

**Zestawienie sortowań dla danych z rozkładem rosnącym**



**Zestawienie sortowań dla danych z rozkładem rosnącym**



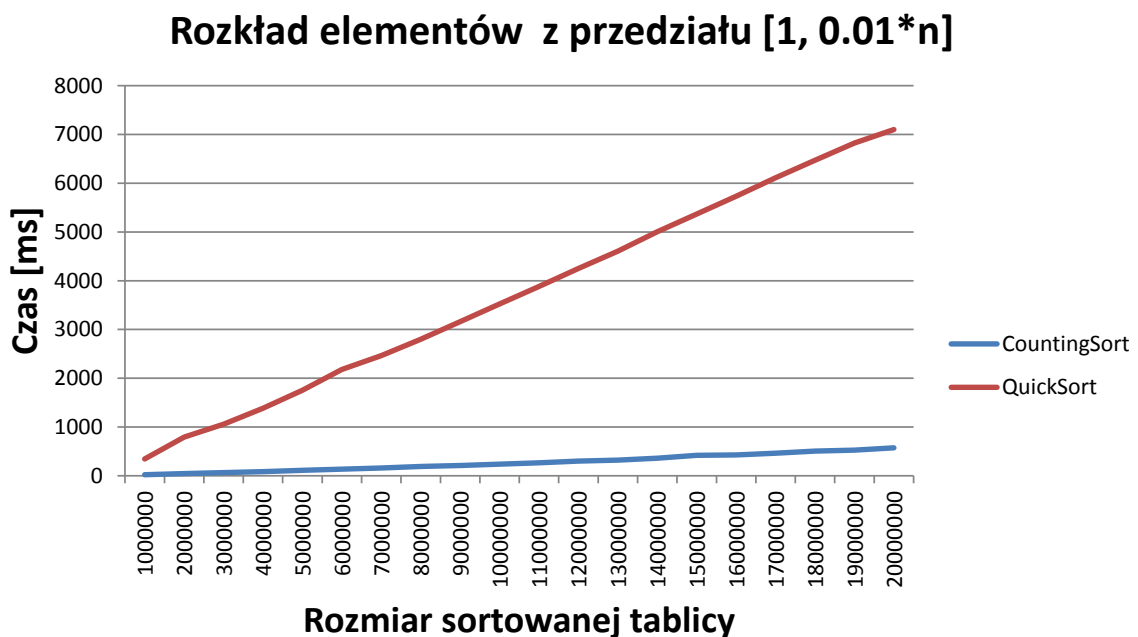
### 3.2.1 Wnioski do rozkładu rosnącego

### 3.3 Wnioski do zależności czasu $t$ od liczby sortowanych elementów $n$ , przy rozkładach losowych i rosnących, dla metod Quick Sort z podziałem wg: skrajnego i środkowego elementu, oraz dla metody Insert Sort

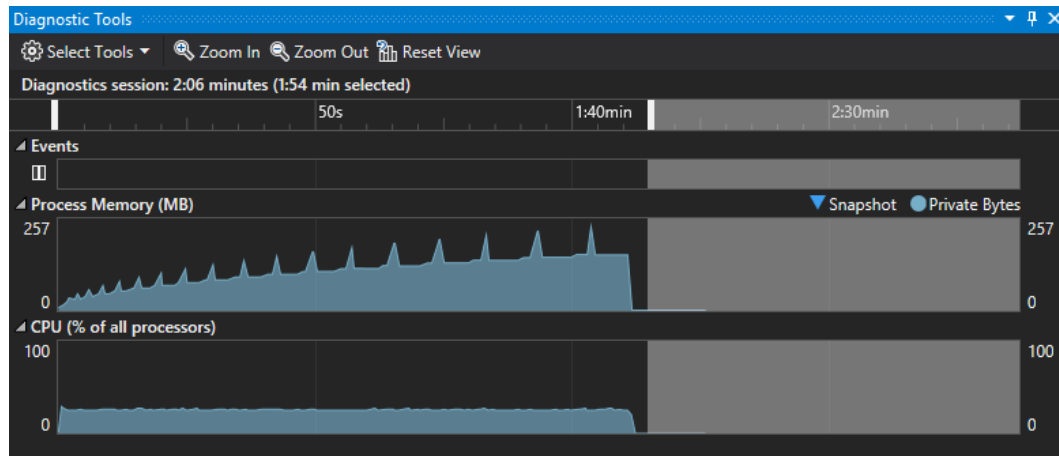
## 4 Badanie zależności czasu obliczeń $t$ od liczby sortowanych elementów $n$ dla metod Counting Sort, Quick Sort, przy rozkładzie losowym, gdy wartości elementów mieszczą się w przedziałach $[1; 0,01n]$ , $[1; 100n]$ .

### 4.1 Przedział $[1; 0,01n]$

Wykres ilustrujący zależność czasu obliczeń od ilości elementów dla metod Counting Sort i Quick Sort, dla rozkładu losowego w przedziale  $[1; 0,01n]$ .



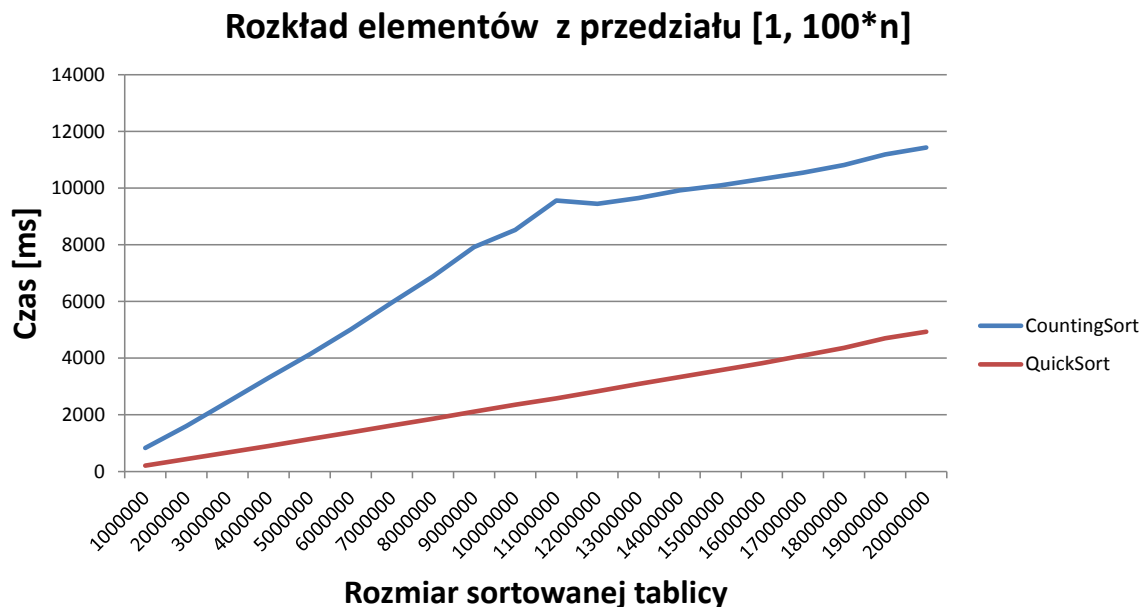
Zrzut ekranu sekcji Diagnostic Tools programu Visual Studio 2015 prezentujący, zużycie pamięci i procesora w trakcie sortowania algorytmami Counting Sort i Quick Sort, dla rozkładu losowego w przedziale  $[1; 0,01n]$ .



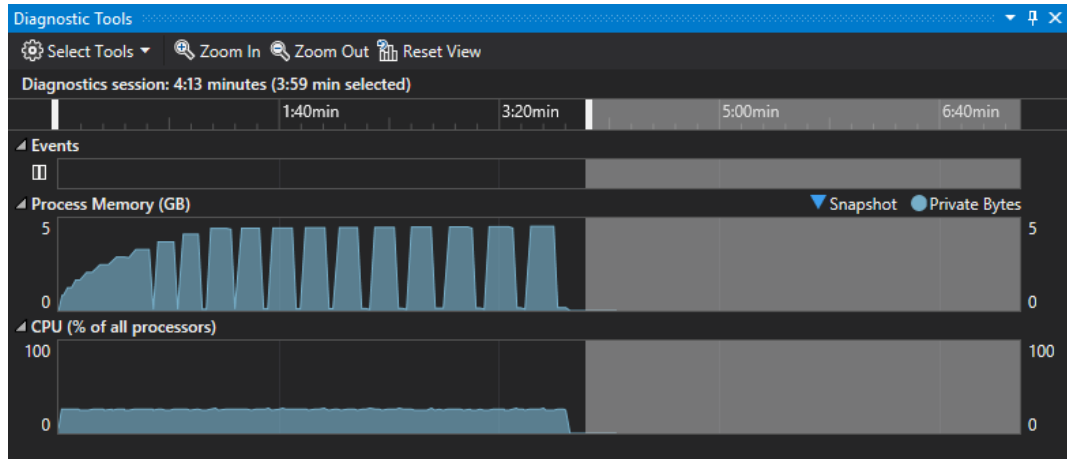
#### 4.1.1 Wnioski do przedziału $[1; 100n]$

### 4.2 Przedział $[1; 100n]$

Wykres ilustrujący zależność czasu obliczeń od ilości elementów dla metod Counting Sort i Quick Sort, dla rozkładu losowego w przedziale  $[1; 100n]$ .



Zrzut ekranu sekcji Diagnostic Tools programu Visual Studio 2015 prezentujący, zużycie pamięci i procesora w trakcie sortowania algorytmami Counting Sort i Quick Sort, dla rozkładu losowego w przedziale  $[1; 100n]$ .



#### 4.2.1 Wnioski do przedziału $[1; 0, 01n]$

#### 4.3 Wnioski do badania zależności czasu obliczeń $t$ od liczby sortowanych elementów $n$ dla metod Counting Sort, Quick Sort, przy rozkładzie losowym, gdy wartości elementów mieszczą się w przedziałach $[1; 0, 01n], [1; 100n]$ .

# Spis treści

<b>1</b>	<b>Implementacja algorytmów sortujących</b>	<b>1</b>
<b>2</b>	<b>Badana zależność czasu obliczeń <math>t[s]</math> od liczby sortowanych elementów <math>n</math>.</b>	<b>1</b>
2.1	Podział metod sortowania	1
2.2	Metody "wolne"	2
2.2.1	Opis algorytmów "wolnych"	2
2.2.2	Wnioski do metod "szybkich"	5
2.3	Metody "szybkie"	5
2.3.1	Opis algorytmów "szybkich"	5
2.3.2	Wnioski do metod "szybkich"	8
2.4	Wnioski do podziału metod sortowania	8
<b>3</b>	<b>Badanie zależności czasu <math>t</math> od liczby sortowanych elementów <math>n</math>, przy rozkładach losowych i rosnących, dla metod Quick Sort z podziałem wg: skrajnego i środkowego elementu, oraz dla metody Insert Sort</b>	<b>8</b>
3.1	Rozkład losowy	9
3.1.1	Wnioski do rozkładu losowego	10
3.2	Rozkład rosnący	10
3.2.1	Wnioski do rozkładu rosnącego	11
3.3	Wnioski do zależności czasu $t$ od liczby sortowanych elementów $n$ , przy rozkładach losowych i rosnących, dla metod Quick Sort z podziałem wg: skrajnego i środkowego elementu, oraz dla metody Insert Sort	11
<b>4</b>	<b>Badanie zależności czasu obliczeń <math>t</math> od liczby sortowanych elementów <math>n</math> dla metod Counting Sort, Quick Sort, przy rozkładzie losowym, gdy wartości elementów mieszczą się w przedziałach <math>[1; 0,01n], [1; 100n]</math>.</b>	<b>11</b>
4.1	Przedział $[1; 0,01n]$	11
4.1.1	Wnioski do przedziału $[1; 100n]$	12
4.2	Przedział $[1; 100n]$	12
4.2.1	Wnioski do przedziału $[1; 0,01n]$	13
4.3	Wnioski do badania zależności czasu obliczeń $t$ od liczby sortowanych elementów $n$ dla metod Counting Sort, Quick Sort, przy rozkładzie losowym, gdy wartości elementów mieszczą się w przedziałach $[1; 0,01n], [1; 100n]$ .	13