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

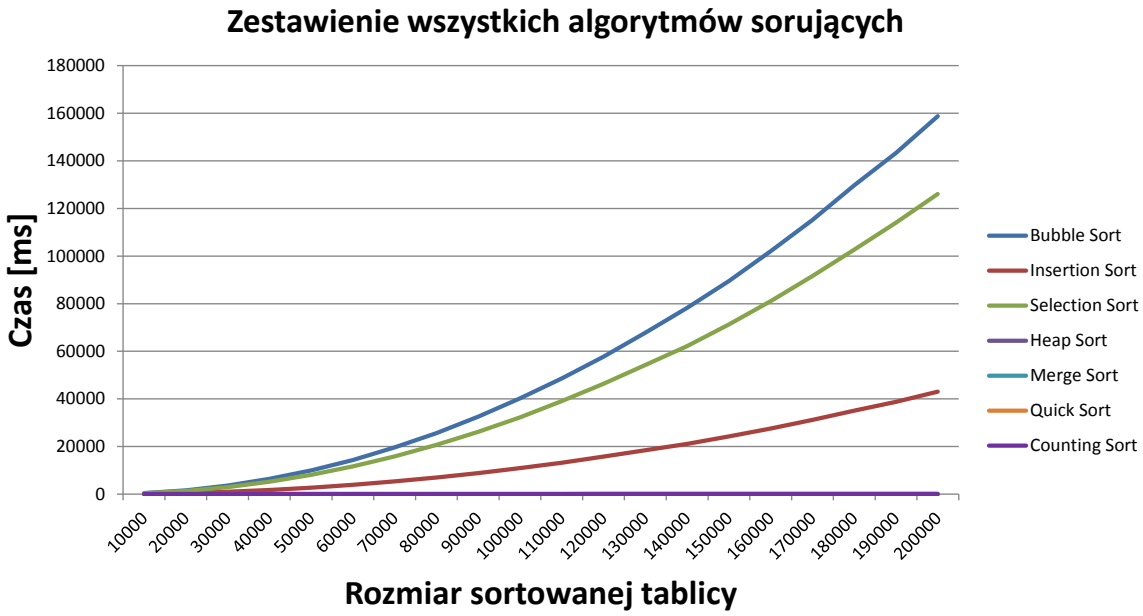
1 Implementacja algorytmów sortujących

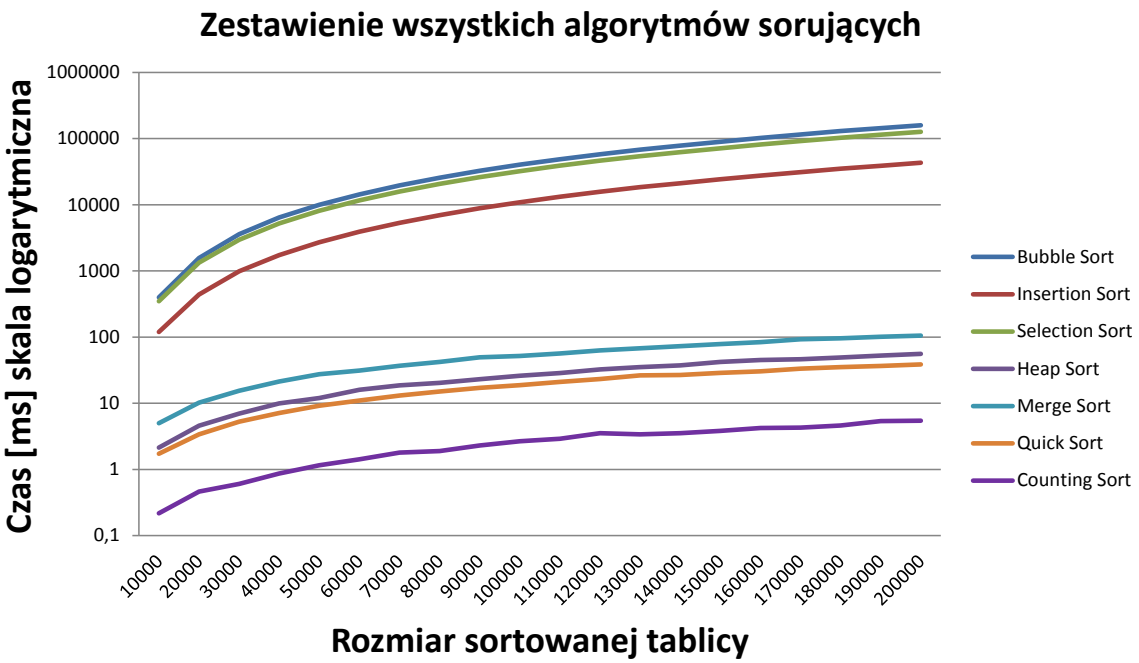
Do implementacji metod sortowania posłużyliśmy się językiem $C++$, każda metoda została napisana w odrębnej funkcji, która za parametry przyjmuje kolejno: wskaźnik na tablicę, rozmiar sortowanej tablicy oraz jako ostatni wartość opcjonalną “reverse” typu bool, która odpowiada za to czy tablica będzie 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 przedstawiajacym 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	$O(n^2)$	$O(n^2)$	$O(n^2)$

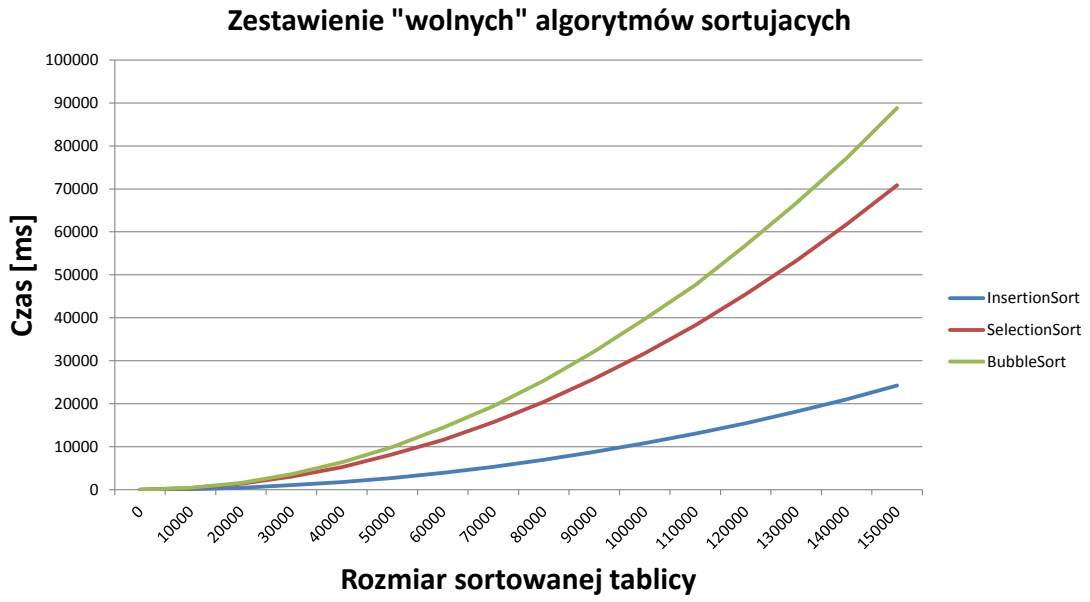
Tablica 3: Tablica złożoności obliczeniowej dla metody Bubble Sort

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 iloci 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

Wady:

- niestabilny
- 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
- stabilny

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:

- niestabilny
- 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
- stabilny

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	$O(n)$	$O(n)$	$O(n)$

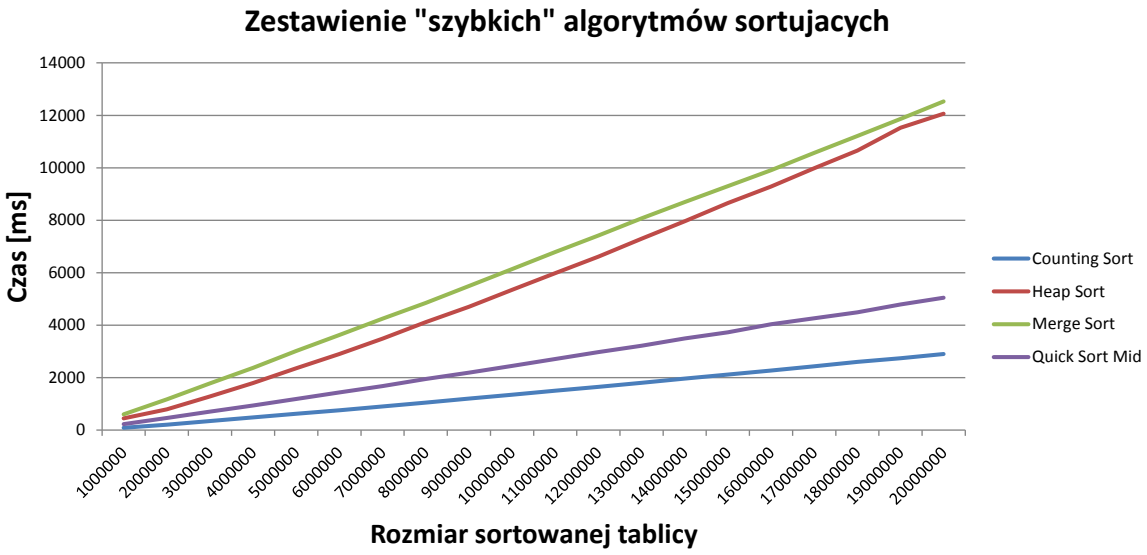
Tablica 8: Tablica złożoności obliczeniowej dla metody Counting Sort

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 iloci elementów dla metod "szybkich"

Wykres ilustrujący zależności 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 środowego 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 Qick Sort Right.

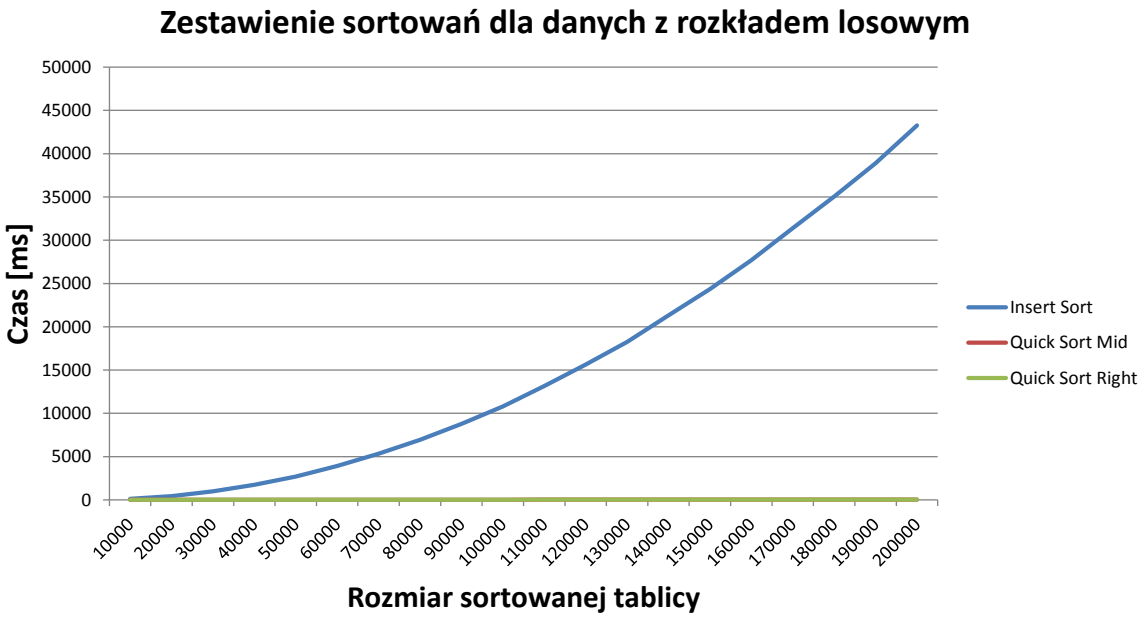
Tabela ilustrująca zależności czasu sortowania od ilości elementów dla metod Quick Sort Mid, Quick Sort Right, Insertion Sort, dla rozkładów losowego i rosnącego.

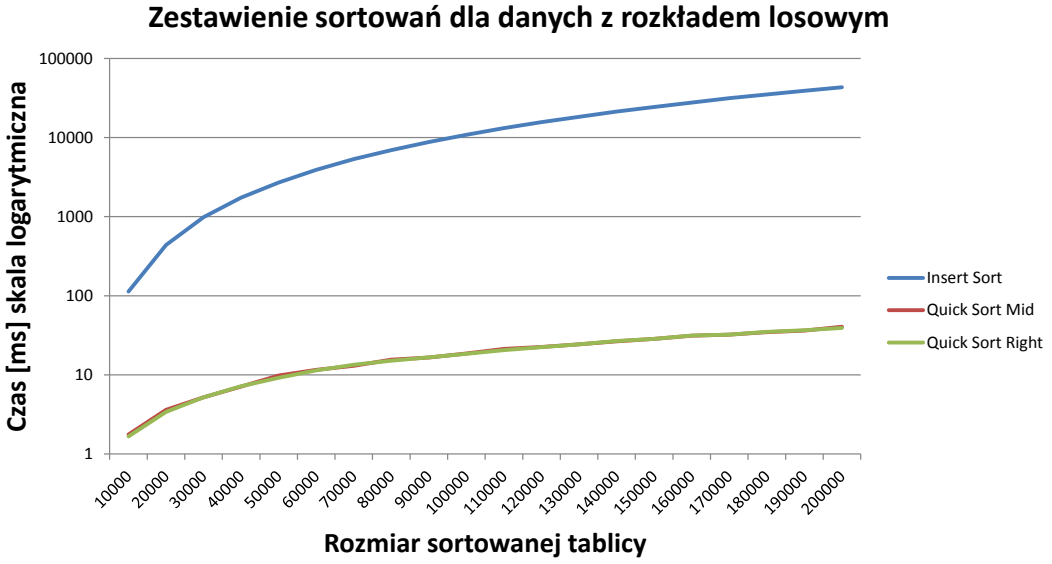
L. ele.	Insert Sort roz. losowy	Quick Sort Right roz. losowy	Quick Sort Mid roz. losowy	Insert Sort roz.rosnący	Quick Sort Right roz. rosnący	Quick Sort Mid roz. rosnący

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.

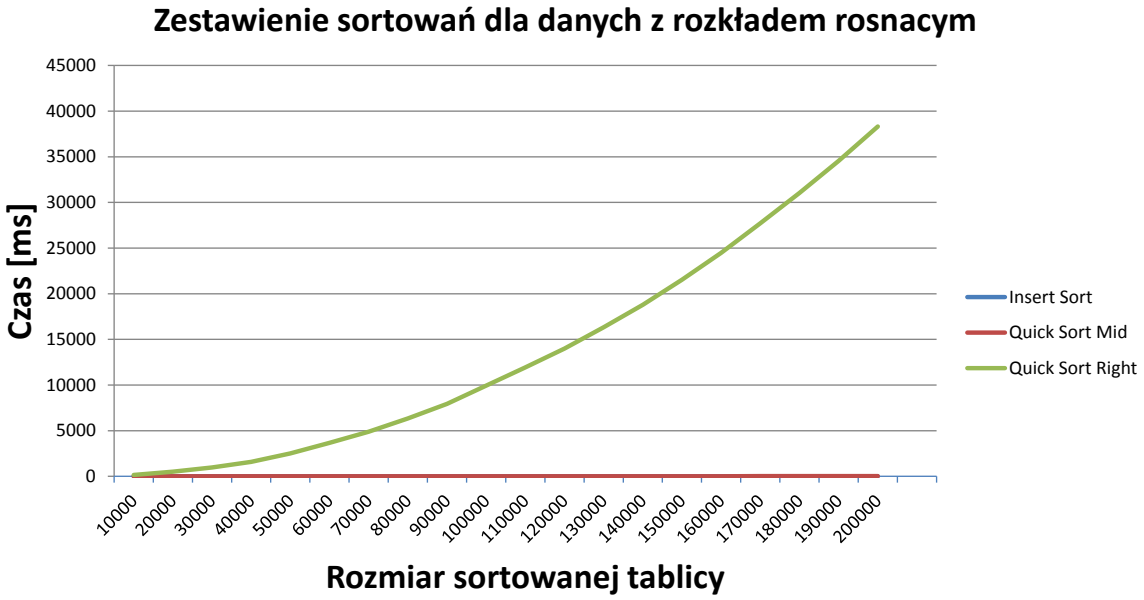


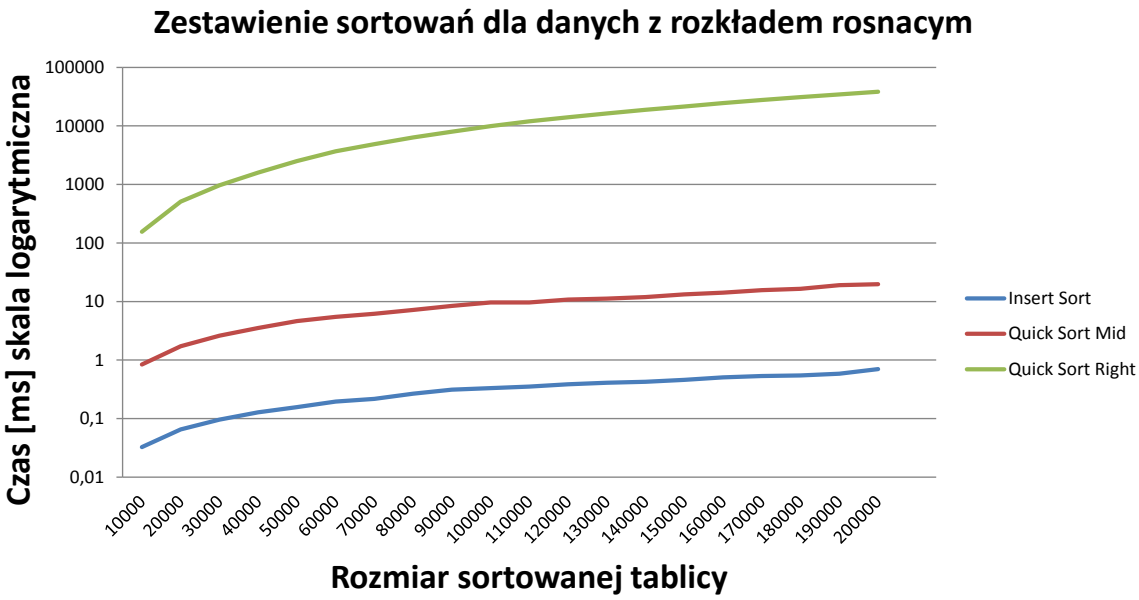


3.1.1 Wnioski do rozkładu losowego

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.





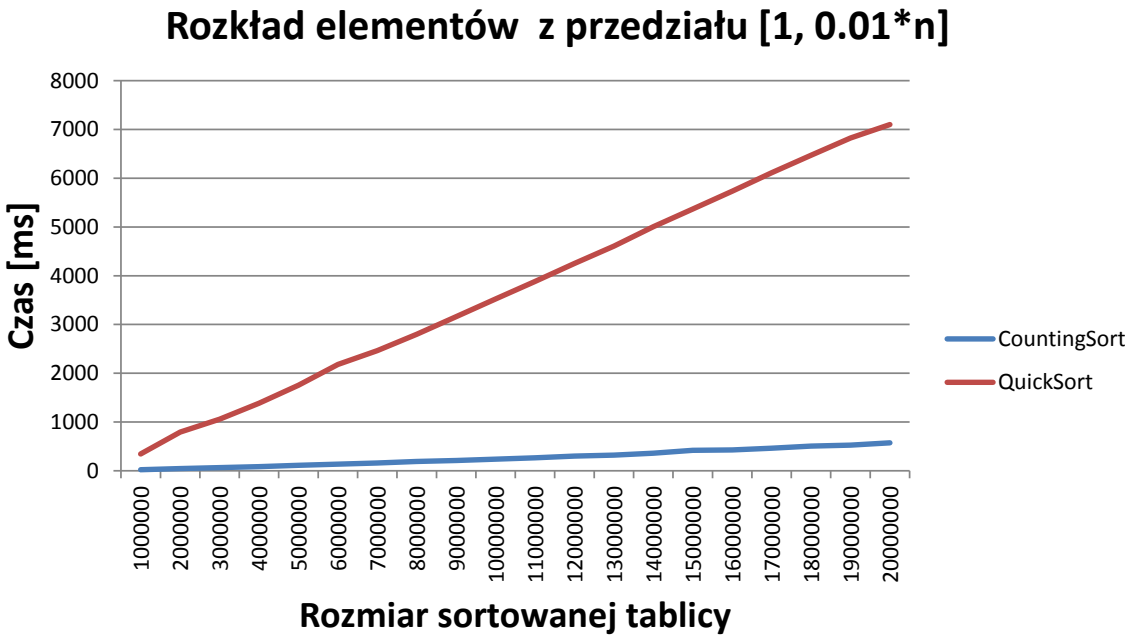
3.2.1 Wnioski do rozkładu rosnącego

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 środowego elementu, oraz dla metody Insert Sort

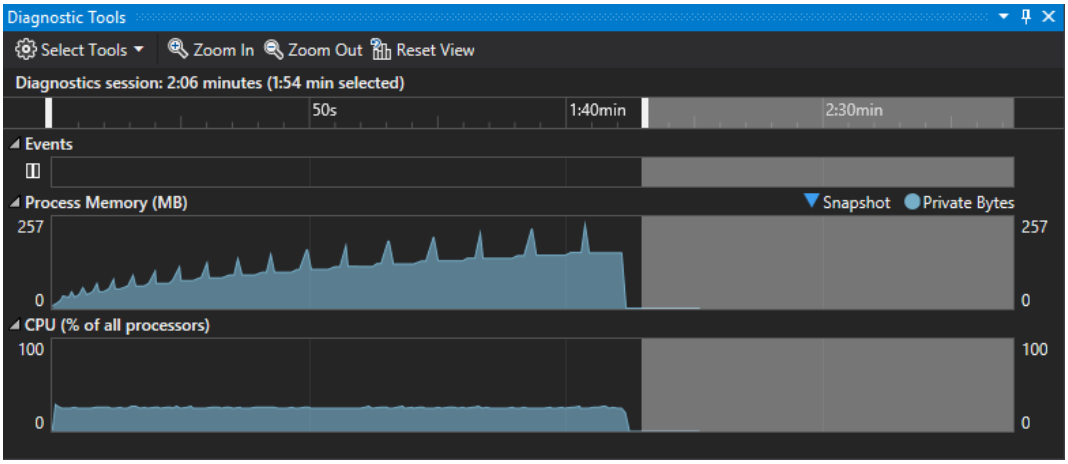
4 Badanie zależności czasu obliczeń t od liczby sortowanych elementów n dla metod Counting Sort, Quick Sort, przy rozkaładzie losowym, gdy wartości elementów mieszczą się w przedziałach $[1;0,01n]$,$[1;100n]$.

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 $[1;0,01n]$.



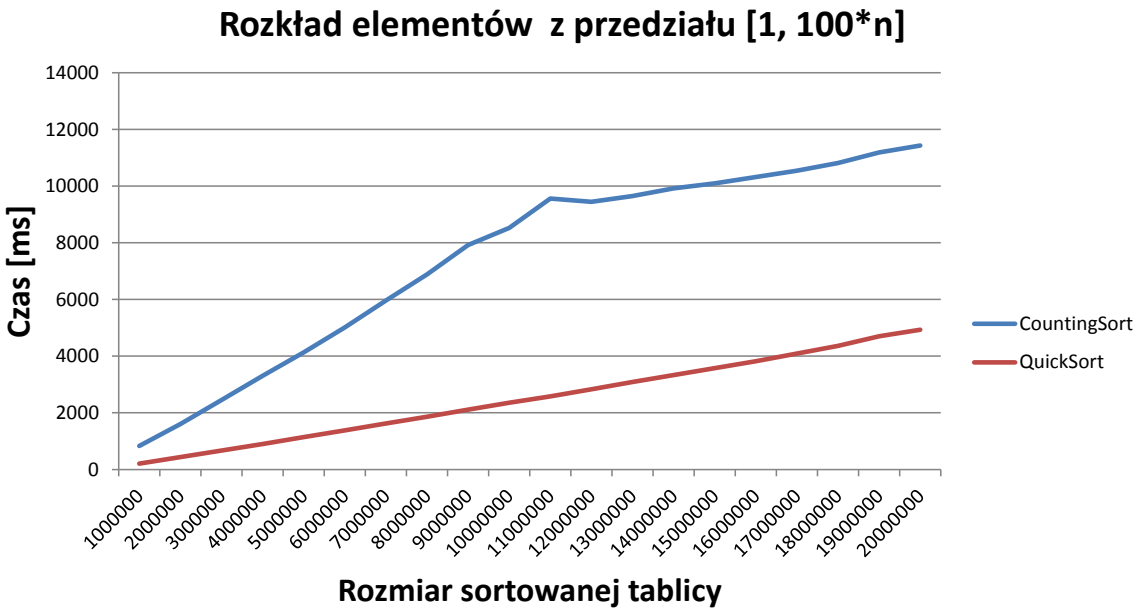
Zrzut ekranu sekcji Diagnostic Tools programu Visual Studio 2015 prezentujący, zużycie pamięci i procesora w trakcie sortowania algorytmami Counting Sort i Quick Sort, dla rozkładu losowego w przedziale $[1;0,01n]$.



4.1.1 Wnioski do przedziału $[1;0.01n]$

4.2 Przedział $[1;100n]$

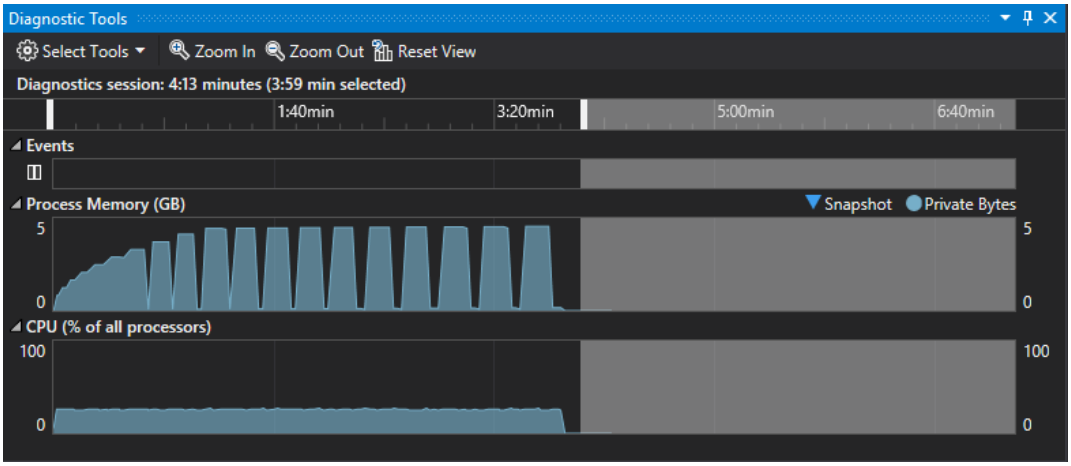
Wykres ilustrujący zależność czasu obliczeń od ilości elementów dla metod Counting Sort i Quick Sort, dla rozkładu losowego w przedziale $[1;100n]$.



Badanie odbywało się w sposób następujący: wylosowanie tablicy do posortowania o rozmiarze n z wartościami z przedziału $[1,100n]$, skopiowanie wylosowanej tablicy do tablicy pomocniczej, sortowanie metodą Quick Sort z pomiarem czasu, przywrócenie posortowanej tablicy do stanu przed sortowaniem przy użyciu tablicy pomocniczej, sortowanie metodą Counting Sort z pomiarem czasu, zwiększenie rozmiaru n i ponowne przeprowadzenie całego procesu, do momentu aż n osiągnie pożądaną wartość. Dzięki takiemu rozwiązaniu w sekcji Diagnostic

Tools programu Visual Studio 2015 można było obserwować nagły wzrost zużywanej pamięci podczas metodą Counting Sort, w porównaniu do ilości pamięci używanej w trakcie sortowania metodą Quick Sort.

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 $[1;100n]$

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