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

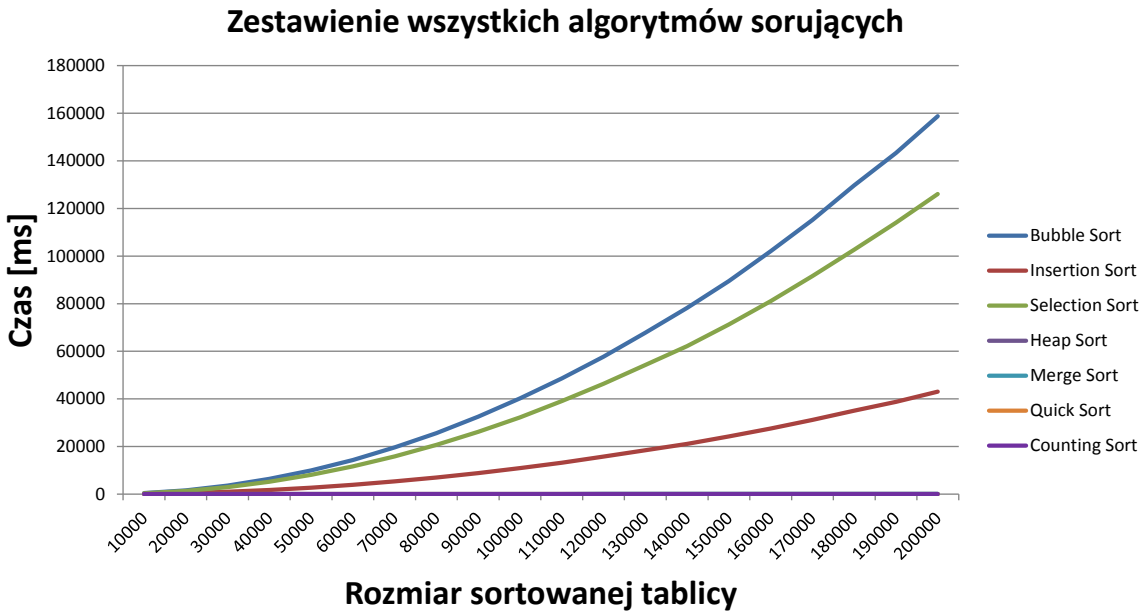
1 Implementacja algorytmów sortujących

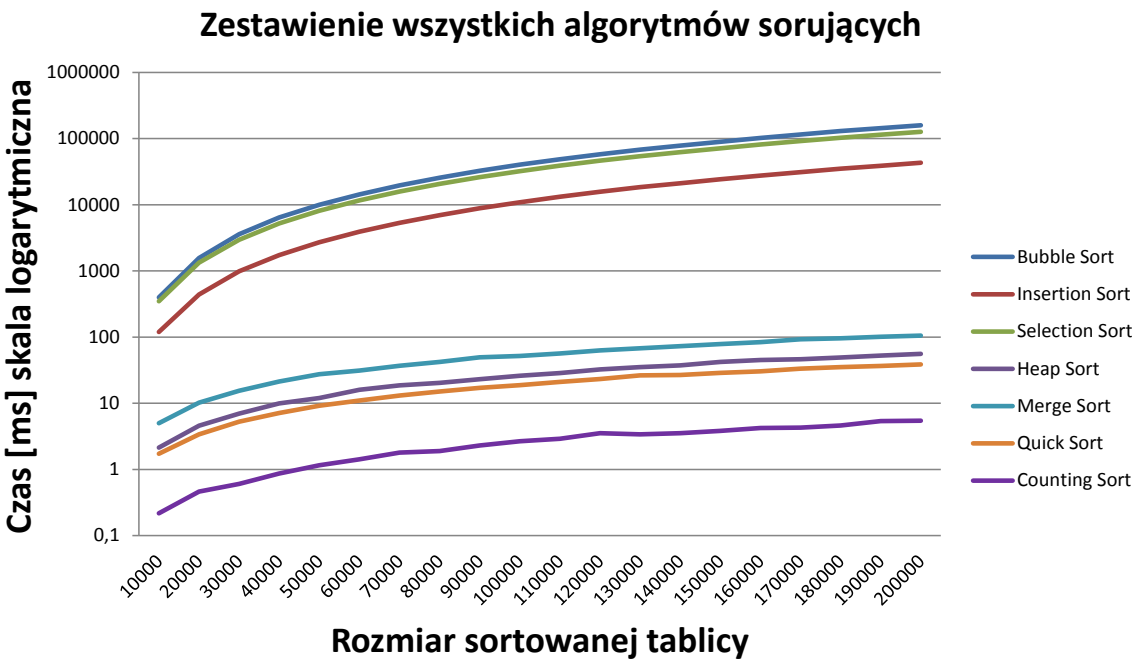
Do implementacji metod sortowania posłużyliśmy się językiem $C++$, każda metoda została napisana w odrębnej funkcji, która za parametry przyjmuje kolejno: wskaźnik na tablicę, rozmiar sortowanej tablicy oraz jako ostatni wartość opcjonalną “reverse” typu bool, która odpowiada za to czy tablica będzie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

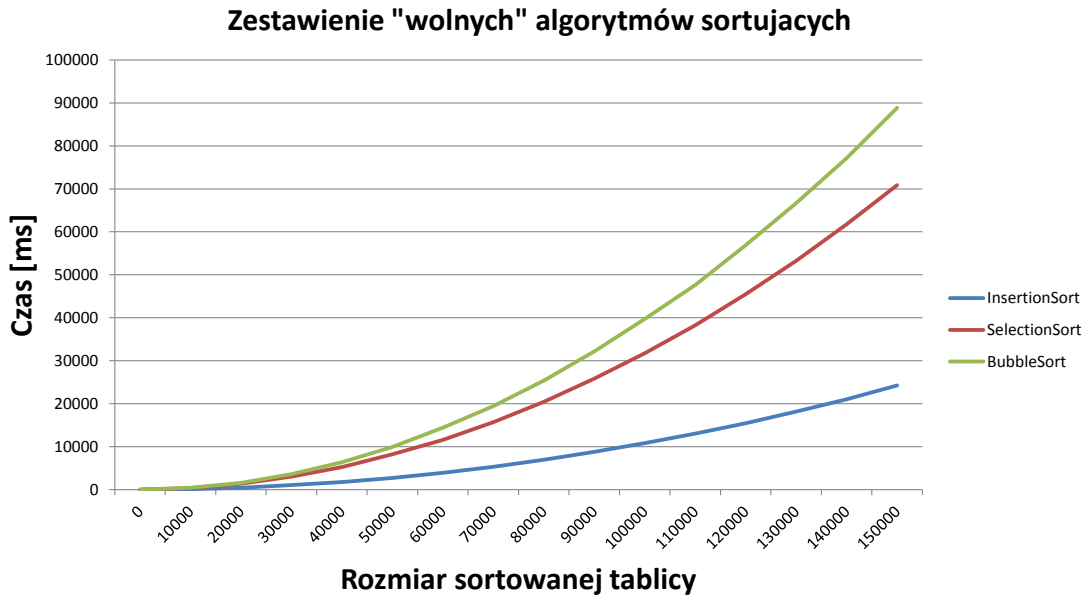
Tablica 3: Tablica złożoności obliczeniowej dla metody Bubble Sort

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

Liczba elem.	Insertion Sort	Selection Sort	Bubble Sort
10000	116,367	343,188	399,54
20000	443,897	1353,49	1553,66
30000	1031,43	3012	3594,1
40000	1729,67	5245,04	6400,49
50000	2736,71	8192,25	9928,66
60000	3893,3	11588,5	14420,7
70000	5305	15701,2	19430,2
80000	6920,76	20416,9	25364,5
90000	8782,01	25847,5	32131,7
100000	10820,4	31732,1	39683,9
110000	13004,1	38229,9	47605
120000	15467	45476,8	56842,2
130000	18181,8	53262,1	66675,1
140000	21019,3	61749	77105,2
150000	24263,4	70873,2	88830,4

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

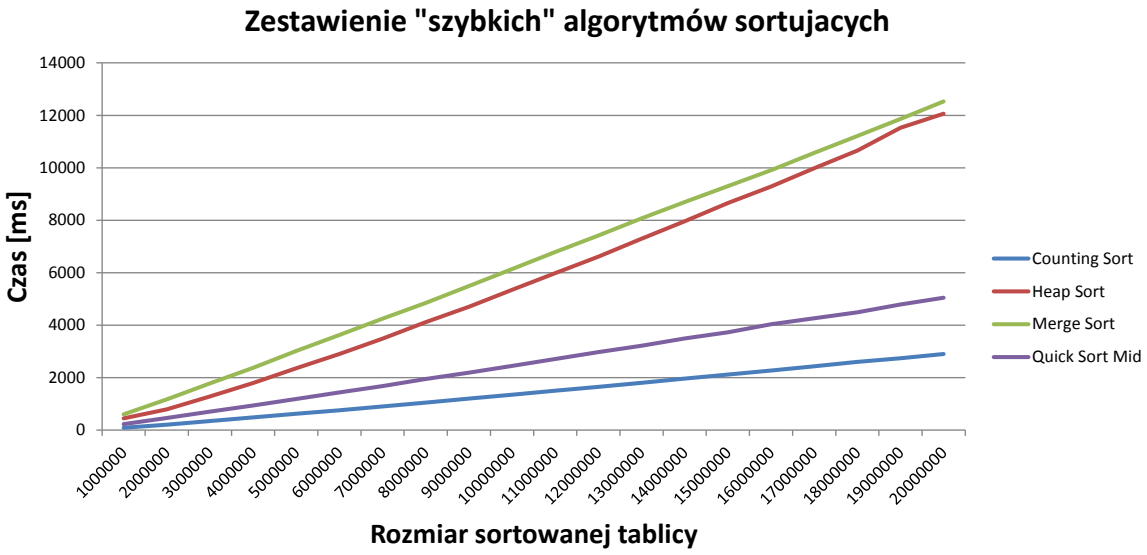
Tablica 8: Tablica złożoności obliczeniowej dla metody Counting Sort

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

Liczba elem.	Counting Sort	Heap Sort	Merge Sort	Quick Sort
1000000	85,9862	441,528	599,972	223,697
2000000	204,386	790,249	1168,86	459,433
3000000	344,066	1281,85	1781,07	698,22
4000000	480,98	1793,7	2373,11	938,052
5000000	617,621	2358,77	3014,62	1185,58
6000000	756,102	2902,17	3626,7	1433,96
7000000	901,962	3486,93	4246,06	1675,53
8000000	1043,01	4115,85	4843,67	1947,04
9000000	1200,79	4700,67	5490,97	2185,94
10000000	1347,63	5347,84	6139,22	2446,77
11000000	1503,04	5980,28	6791,13	2704,96
12000000	1649,68	6614,99	7416,89	2969,83
13000000	1803,17	7291,14	8067,89	3210,14
14000000	1961,38	7964,01	8694,71	3489,29
15000000	2114,89	8658,49	9302,72	3724,19
16000000	2269,11	9285,66	9908	4032,19
17000000	2432,23	9984,3	10570,9	4258,68
18000000	2597,7	10655,1	11213,5	4489,62
19000000	2740,3	11530,3	11864,9	4786,57
20000000	2896,79	12065,7	12529,9	5046,56

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.

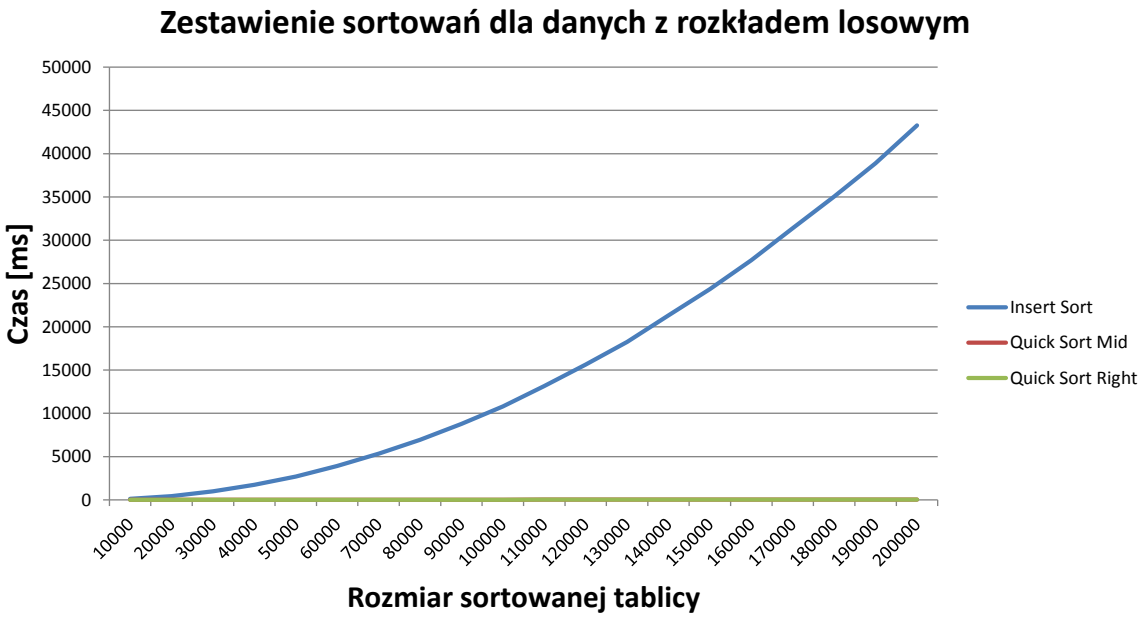
Tabela ilustrująca zależności czasu sortowania od ilości elementów dla metod Quick Sort Mid, Quick Sort Right, Insertion Sort, dla rozkładów losowego i rosnącego.

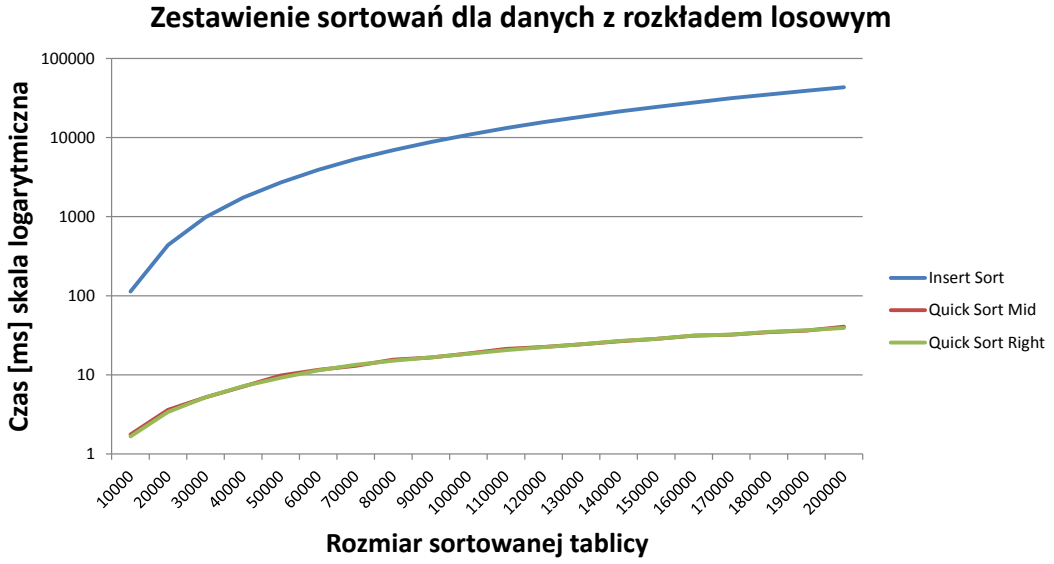
Liczba. elem.	Insert Sort roz. losowy	Quick Sort R roz. losowy	Quick Sort M roz. losowy	Insert Sort roz.rosnący	Quick Sort R roz. rosnący	Quick Sort M roz. rosnący
10000	113,203	1,6697	1,76432	0,032714	155,83	0,838059
20000	437,488	3,39137	3,60882	0,065428	509,5	1,72166
30000	984,901	5,20155	5,21053	0,096218	969,508	2,61136
40000	1735,75	7,16728	7,09833	0,127649	1590,15	3,52479
50000	2702,44	9,20614	9,80398	0,156515	2498,85	4,63739
60000	3912,19	11,4185	11,567	0,195002	3675,9	5,46967
70000	5331,38	13,4157	12,9942	0,217132	4884,01	6,15699
80000	6932,69	15,1264	15,5626	0,265882	6331,66	7,15702
90000	8767,07	16,623	16,5934	0,31335	7946	8,44217
100000	10802,7	18,4383	18,6365	0,331311	9934,8	9,62822
110000	13141,5	20,523	21,3091	0,351517	11940	9,6619
120000	15639,8	22,4413	22,4955	0,386476	13993,6	10,8232
130000	18263,4	24,3281	24,3358	0,409889	16322,7	11,2511
140000	21317,5	26,8089	26,5324	0,425926	18797	11,8903
150000	24363,9	28,5161	28,5139	0,459281	21520,3	13,2566
160000	27679	31,3674	31,224	0,505466	24480,6	14,1598
170000	31397,9	32,184	32,2783	0,535934	27719,3	15,5934
180000	35062,8	35,0455	34,6744	0,54748	31037,2	16,4565
190000	38904,7	36,609	36,2957	0,586288	34542,1	19,0432
200000	43261,1	39,1665	40,5784	0,701429	38319,8	19,7654

Tablica 10: Wyniki badań zależności czasu od ilości elementów dla metod Quick Sort M (z podziałem według środkowego elementu), Quick Sort R (z podziałem według skrajnego elementu), 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.

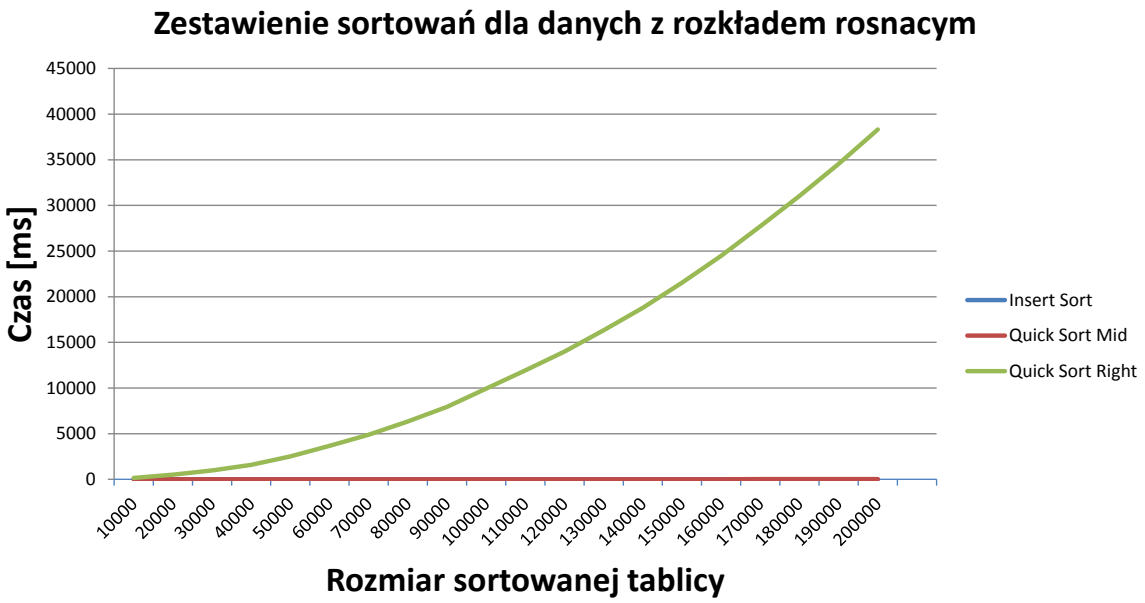


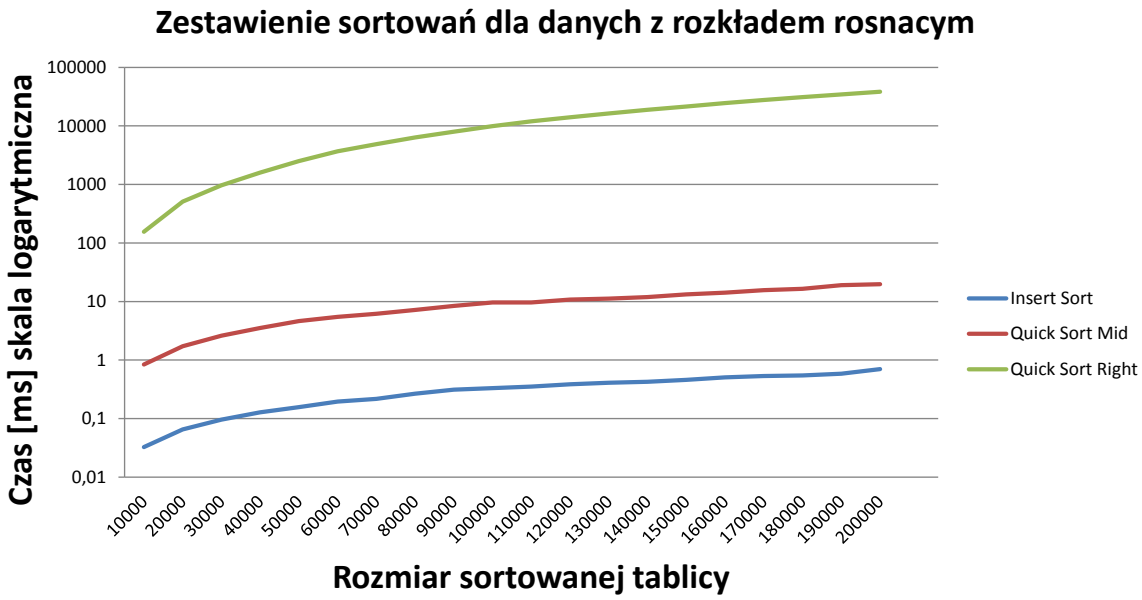


3.1.1 Wnioski do rozkładu losowego

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.





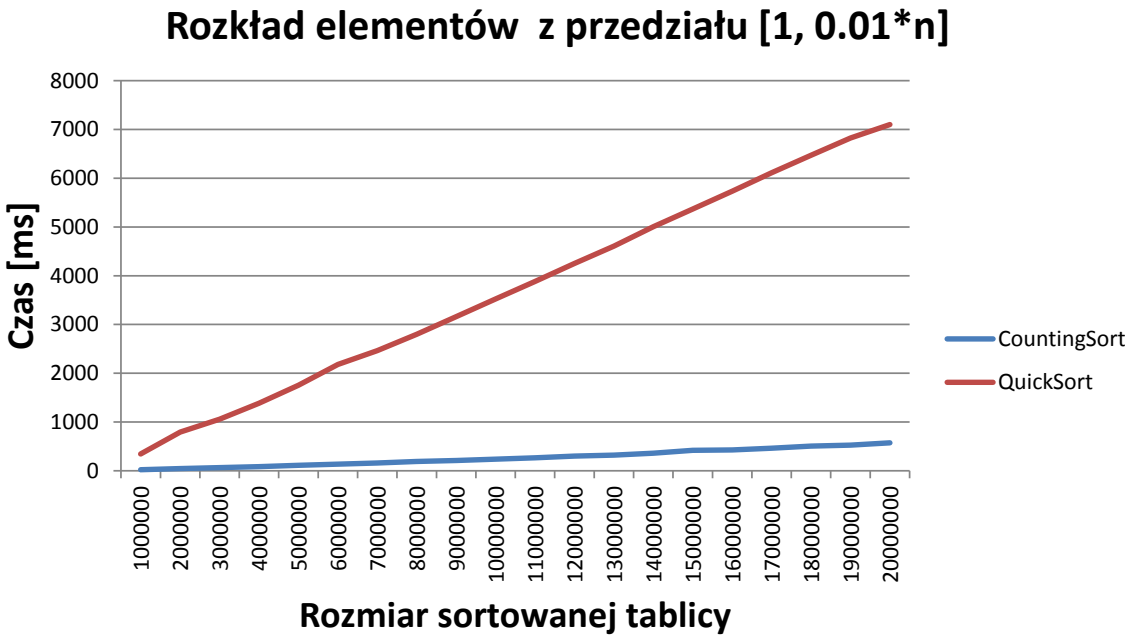
3.2.1 Wnioski do rozkładu rosnącego

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 $[1;0,01n]$,$[1;100n]$.

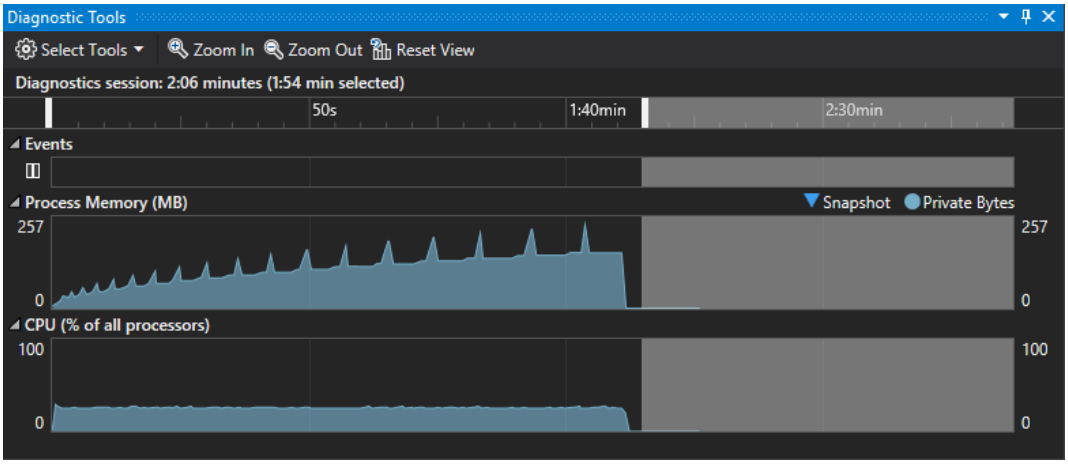
4.1 Przedział $[1;0,01n]$

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



Badanie odbywało się w sposób następujący: wylosowanie tablicy do posortowania o rozmiarze n z wartościami z przedziału $[1;0,01n]$, 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 zauważalny 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 $[1;0,01n]$.

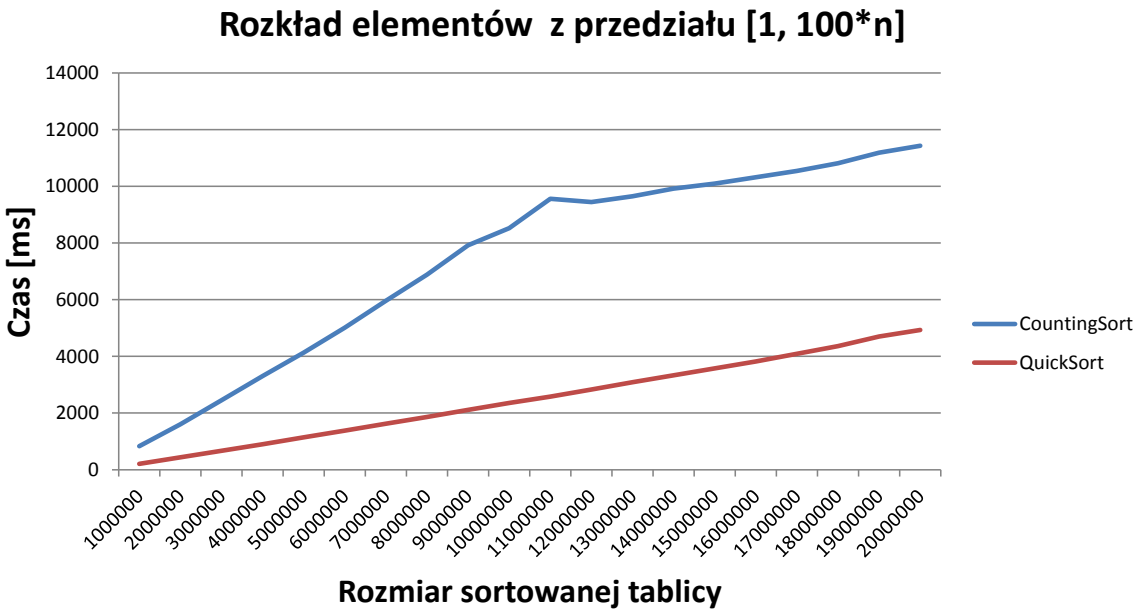


4.1.1 Wnioski do przedziału $[1;0.01n]$

Po przeanalizowaniu wyników badania można zauważyć że metoda Counting Sort działa znacznie szybciej niż Quick Sort w przedziale $[1;0,01n]$, przy trochę większym wykorzystaniu pamięci. Metoda Counting Sort sprawdza się lepiej przy sortowaniu tablic z małym zakresem liczb od Quick Sort, co czyni ją efektywniejszą, ale tylko w przypadku kiedy mamy wystarczającą ilość pamięci przeznaczonej dla procesu.

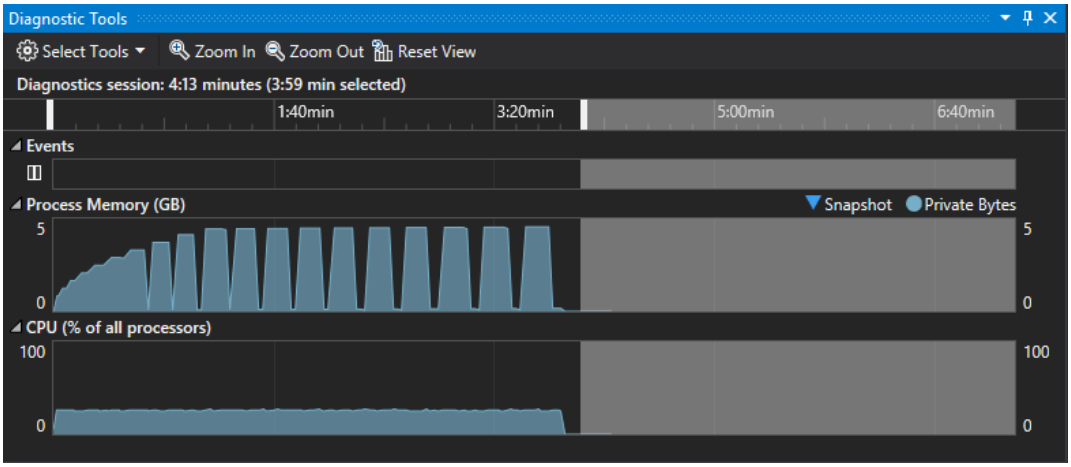
4.2 Przedział $[1;100n]$

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 $[1;100n]$

Po przeanalizowaniu wyników badania można zauważyć że metoda Quick Sort działa znacznie szybciej niż Counting Sort w przedziale $[1;100n]$, przy szalenie większym (w niektórych przypadkach nawet dwudziestokrotnie) wykorzystaniu pamięci. Metoda Quick Sort sprawdza się lepiej przy sortowaniu tablic z dużym zakresem liczb metody od Counting Sort, co czyni ją efektywniejszą.

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 $[1;0,01n]$,$[1;100n]$.

Po przeprowadzeniu badana jasno widać jak ważne jest dobranie odpowiedniej metody sortowania dla swoich danych. Na efektywność użytej metody ma wpływ między innymi przedział elementów. W przypadku kiedy pracujemy na mniejszym przedziale warto wybrać metodę Counting Sort, jeżeli posiadamy wystarczającą ilość pamięci. Przy większym przedziale metoda Quick Sort staje się efektywniejsza, Counting Sort nie tylko działa wolniej, ale też zużywa wielokrotnie więcej pamięci.

Spis treści