Projektowanie Algorytmów i Metody Sztucznej Inteligencji
Marta Emirsajłow piątek 13:15-15:00
Marcin Gruchała 248982 Projekt 1

1 Wstęp.

Celem projektu jest zaimplementownie trzech różnych algorytmów sortowania oraz przeprowadzenie analizy ich efektywności. Zaimplementowane zostały algorytmy: przez scalanie(merge sort), szybkie sortowanie(quicksort), introspektywne(introsort). Test efektywności polega na zmierzeniu czasu sortowania 100 tablic o rozmiarach: 10 000,50 000,100 000, 500 000, 1 000 000. Testy efektywnościu należy sprawdzić dla różnych stanów początkowych: wszystkie elementy tablicy losowe, pewien procent początkowych elementów tablicy jest już posortowany(25%, 50%, 75%, 95%, 99%), 99, 7%, wszystkie elementy tablicy posortowane w odwrotnej kolejności.

2 Opis algorytmów.

2.1 Przez scalanie(mergesort)

Jest to rekurencujny stabilny algorytm sortowania. Algorytm dzieli tablice na dwie części następinie dla każdej z częsci stosuje algorytm sortowania przez scalanie do momentu dojścia do jednego elementu tablicy. Następnie program scala pojedyńcze podtablice w coraz większe już posortowane części orginalnej tablciy aż otrzyma całą tablice posortowaną. Złożoność obliczniecowa tego algorytmu to dla każdego przypadku $nlog(n)$. Implementacja tego algorytmu w projekcie znajduje się w metodzie Array::mergesort. Metoda ta kożysta z prywatej metody Array::sortAndMerge która wykonuje sortowanie oraz scalanie elementów tablicy.

2.2 Szybkie sortowanie(quicksort)

Popularny algorytm sortowania ze względu na swoją szybkośc i prostote implementacji. Działanie algorytmu polega na wybraniu elementu tablicy nazywanego pivot a następnie sortowanie tablicy w taki sposób że elementy mniejsze lub równe znajdują się za tym elementem(na lewo od niego) a elementy więszke przed wybranym elementem(na prawo od niego). Następnie twożone są podtablice z obu stron elementu pivot które są sortowane rekurencyjnie według opisanego schematu. Algorytm można zaimplementować na różne sposoby w zależności od sposbu wybierania pivotu. Złożoność obliczeniowa algorytmu w normalnej sytuacji to $O(nlog(n))$ a w najgorszej to $O(n^2)$. Implementacja tego algorytmu w projekcie znajduje się w metodzie Array::quicksort. Metoda ta kożysta z prywatnej metody Array::divide która jako pivot przyjmuje ostatni element tablicy a następnie sortuje tablice względem niego.

2.3 Introspektywne(introsort)

Hybrydowy algorytm sortowania bazujący na algorytmnie sortowania szybkiego z wykorzystaniem innych algorytmów w celu wyeliminowania najgorszego przypadku sortowania szybkiego. Algorytm dzileli tablice na podtablice tak jak robił to algorytm szybkiego sortowania do pewnej określonej przy pierwszym wywołaniu algorytmu głębokości okreslonej jako $2log$(początkowy rozmiar tablicy). Z każdym wywołaniem głębokość ta się zmniejsza aż dojedzie do zera. Następnie algorytm obsługuje dwa przypadkki. Pierwsz gdy algorytm dojdzie do maksymalnej głębokości wykonuje sortowanie podtablic algorytmem sortowania przez kopcowanie(heap sort). Drugi to jeżeli podtablica ma mniej niż 16 elementów to jest sortowana za pomocą algorytmu sortowania przez wstawianie (insertion sort). Złożoność obliczeniowa algorytmu w normalnej sytuacji to $O(nlog(n))$ a w najgorszej to $O(n^2)$. Implementacja tego algorytmu w projekcie znajduje się w metodzie

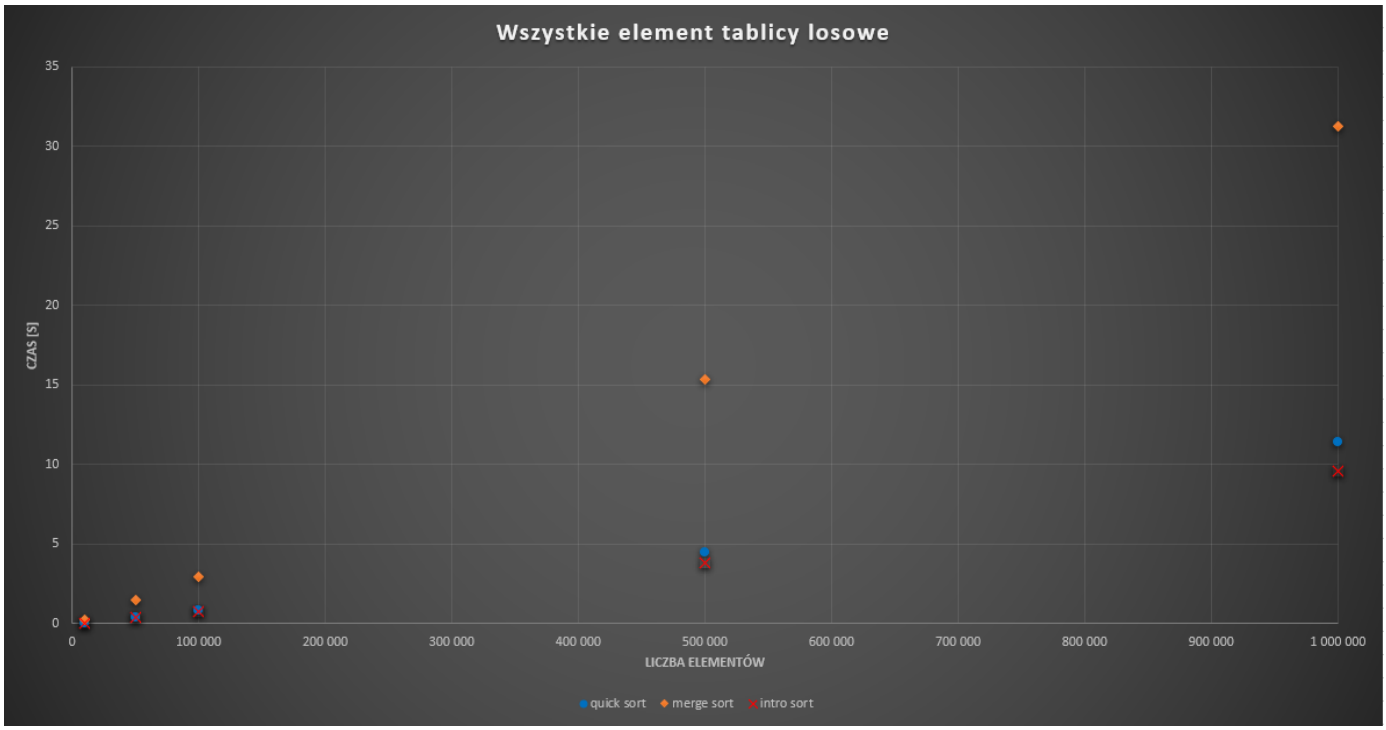
Array::introsort. Metoda działa podobnie do metody Array::quicksort z uwzględnieiem hybrydowej natury algorytmu introspektywnego.

3 Implementacja oraz przebeig eksperymentów.

Implementacja projektu opiera się o klasę Array która alokuje pamieć dla różnych rozmiarów tabel dla różnych typów liczbowych. Implementacja klasy znajduje się w pliku Array.h. Usuwanie pamięci odbywa się w destruktorach. Elementy tablicy są generowane przy pomocy metody Array::random. Testowanie poprawności działania metod sortujących pierw przebiegało wizualnie na małych tablicach a następnie na dużych za pomocą metody Array::check. Testowanie efektywności programu polegało na zmierzeniu czasu relaizacji metody relizującej sortowanie. Program przy jendym wywyłaniu sortuje 100 tablic o określonym w kodzie rozmiarze(drugi parametr szablonu klasy Array), dla wszystkich stanów początkowego posortowania. Implementacja testów efektywnośći znajduje się w pliku main.cpp. Czas jest mierzony przy użyciu własnośći biblioteki < chrono >. Implementacja stopera znajduje się w klasie timer w plikach tmier.h oraz timer.cpp. Niestety ze wzglęu na problemy z algorytmem sprtowania szybkiego podczas sortowania tablic posortowanych w odwrotnej kolejności projekt należ uruchomić z wykomentowaną częścią odpowiedzialną za sortowanie quciksort tablicy odwrotnie posortowanej(w załącznuiku na końcu jest zdjęcie). Wyniki eksperymentów dla poszczególnych stanów posortowania prezętują się następująco. (W załączniku na końcu znajdują się zdjęcia testów użytch do zrobienia pomiarów.)

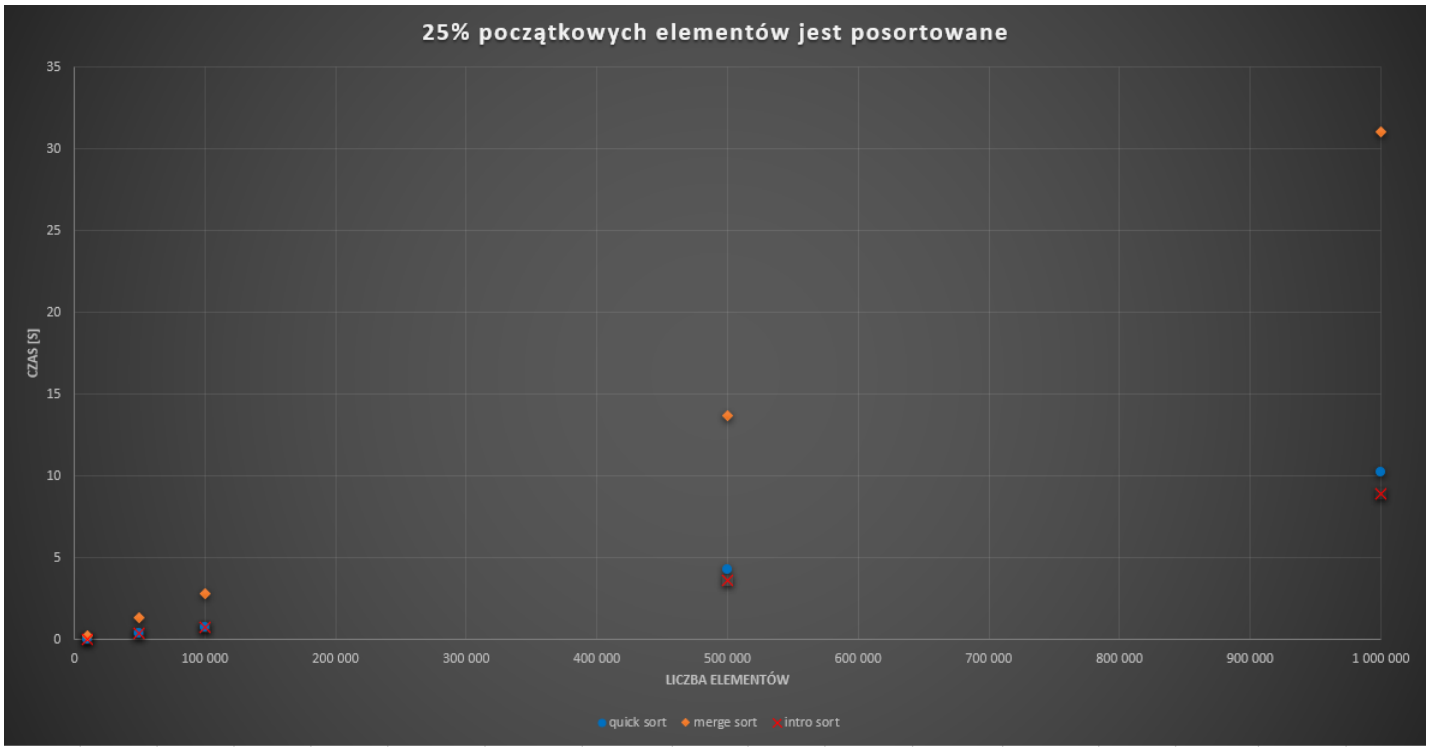
3.1 Wszystkie elementy tablicy losowe.

Wszystkie element tablicy losowe			
Rozmiar	quisk sort [s]	merge sort [s]	intro sort [s]
10 000	0,0649439	0,2666180	0,0569878
50 000	0,4046310	1,5094800	0,3735260
100 000	0,8369900	2,9766900	0,7520100
500 000	4,4593200	15,3855000	3,8312100
1 000 000	11,4063000	31,2544000	9,5597000



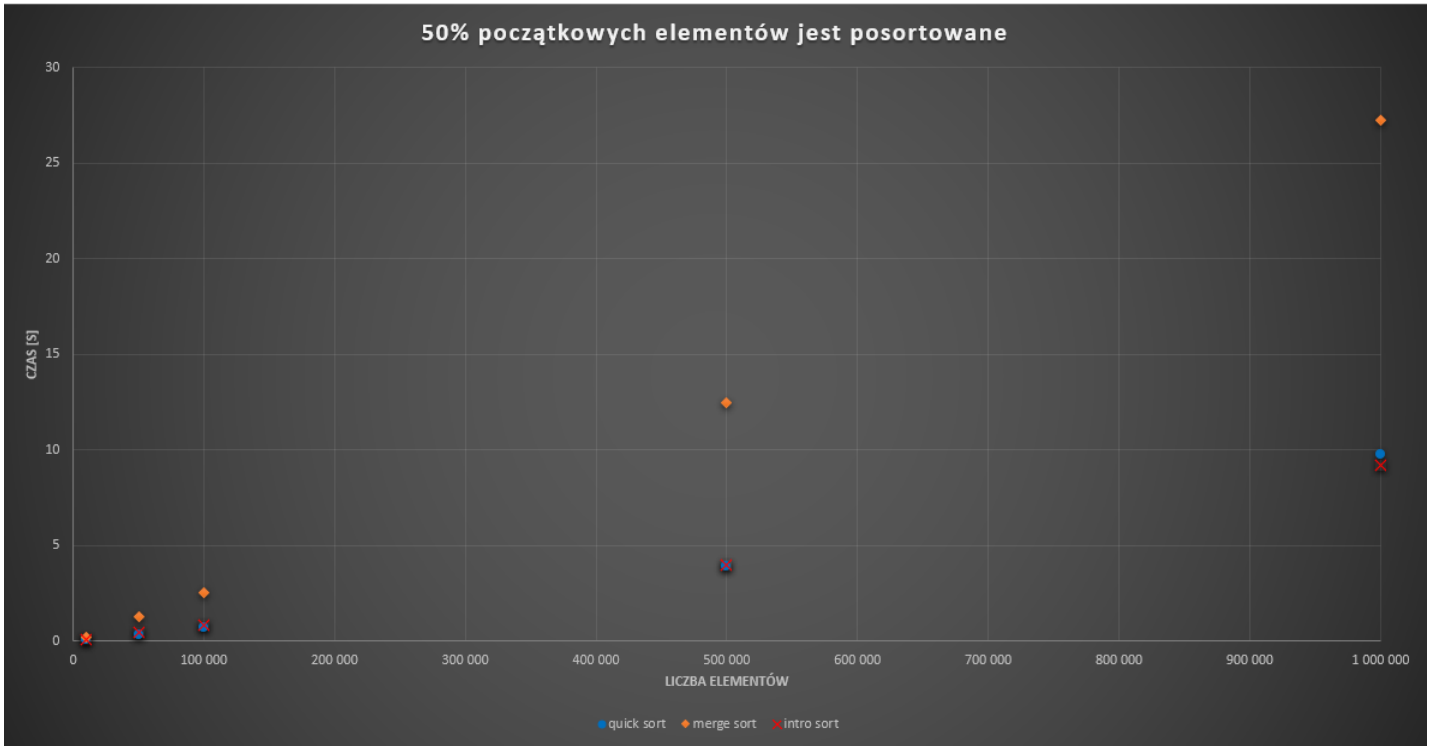
3.2 25 % początkowych elementów jest posortowanych.

25% początkowych elementów jest posortowane			
Rozmiar	quisk sort [s]	merge sort [s]	intro sort [s]
10 000	0,0606671	0,24829	0,054795
50 000	0,387924	1,38123	0,36459
100 000	0,773384	2,79115	0,737603
500 000	4,27816	13,6669	3,61972
1 000 000	10,266	31,0347	8,94685



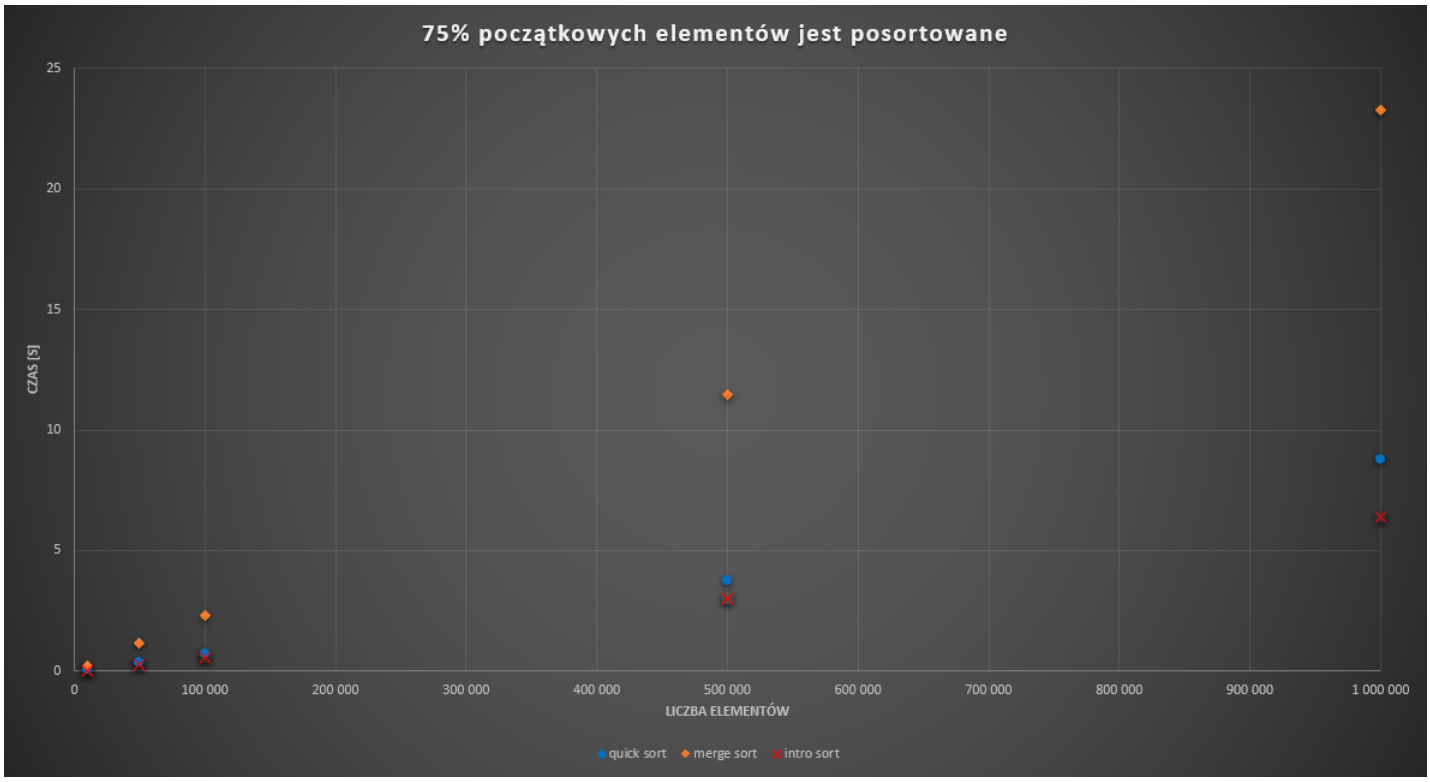
3.3 50 % początkowych elementów jest posortowanych.

50% początkowych elementów jest posortowane			
Rozmiar	quisk sort [s]	merge sort [s]	intro sort [s]
10 000	0,0617681	0,230215	0,0779355
50 000	0,353101	1,26012	0,4696928
100 000	0,736136	2,56194	0,817823
500 000	3,90768	12,4528	3,99028
1 000 000	9,76145	27,2372	9,1996



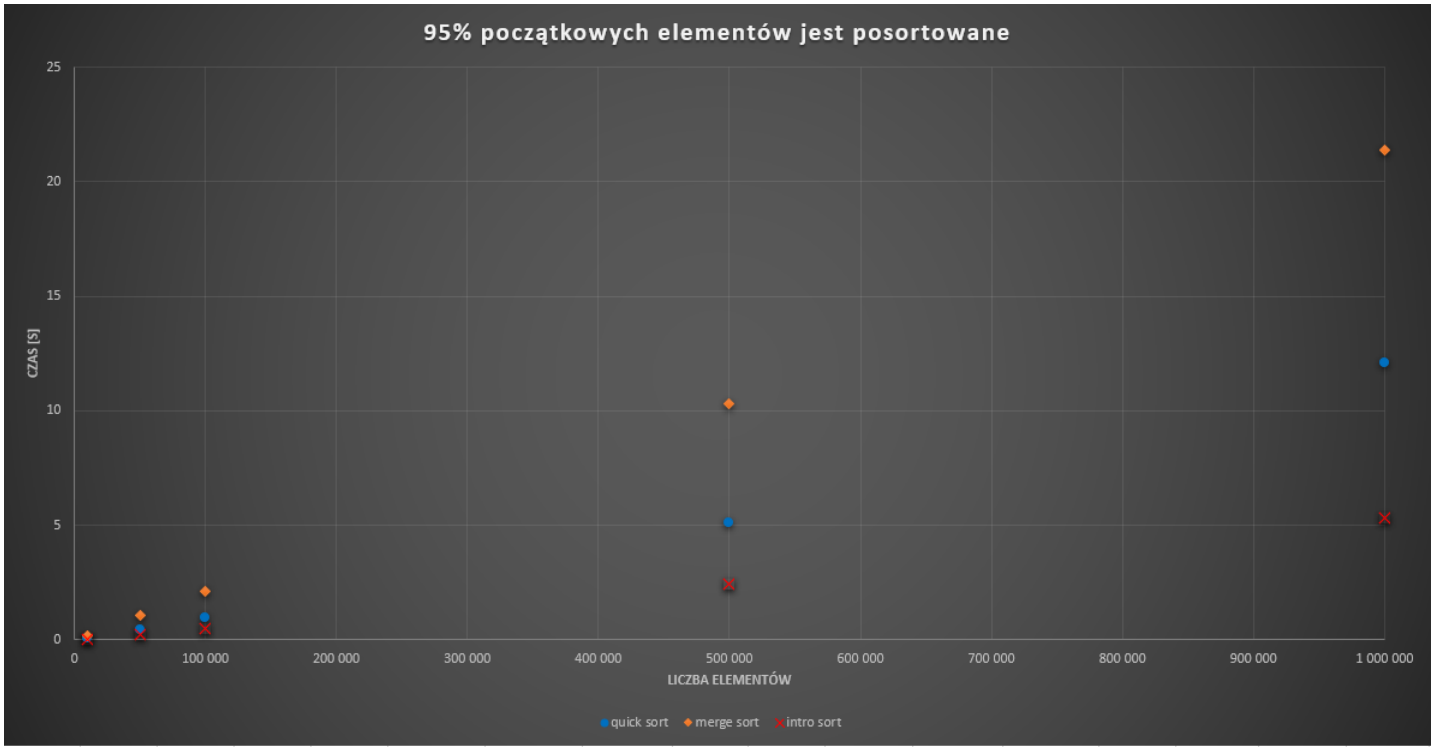
3.4 75 % początkowych elementów jest posortowanych.

75% początkowych elementów jest posortowane			
Rozmiar	quisk sort [s]	merge sort [s]	intro sort [s]
10 000	0,0519703	0,210293	0,0397094
50 000	0,348352	1,15963	0,262566
100 000	0,695233	2,31497	0,567716
500 000	3,77816	11,448	3,03266
1 000 000	8,76549	23,2855	6,40946



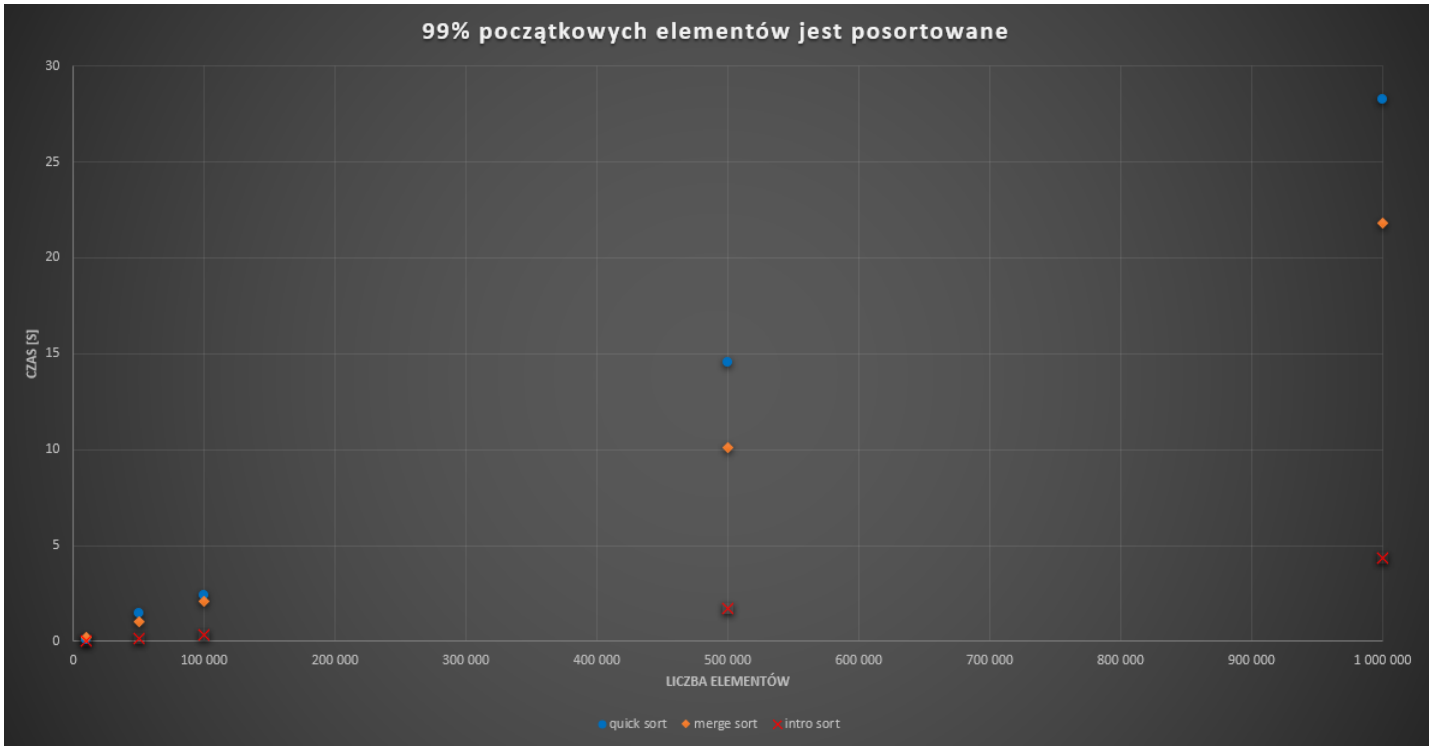
3.5 95 % początkowych elementów jest posortowanych.

95% początkowych elementów jest posortowane			
Rozmiar	quisk sort [s]	merge sort [s]	intro sort [s]
10 000	0,0753162	0,19456	0,0323503
50 000	0,42587	1,05464	0,221816
100 000	0,947471	2,13788	0,483599
500 000	5,13044	10,3085	2,41872
1 000 000	12,0891	21,3954	5,32542



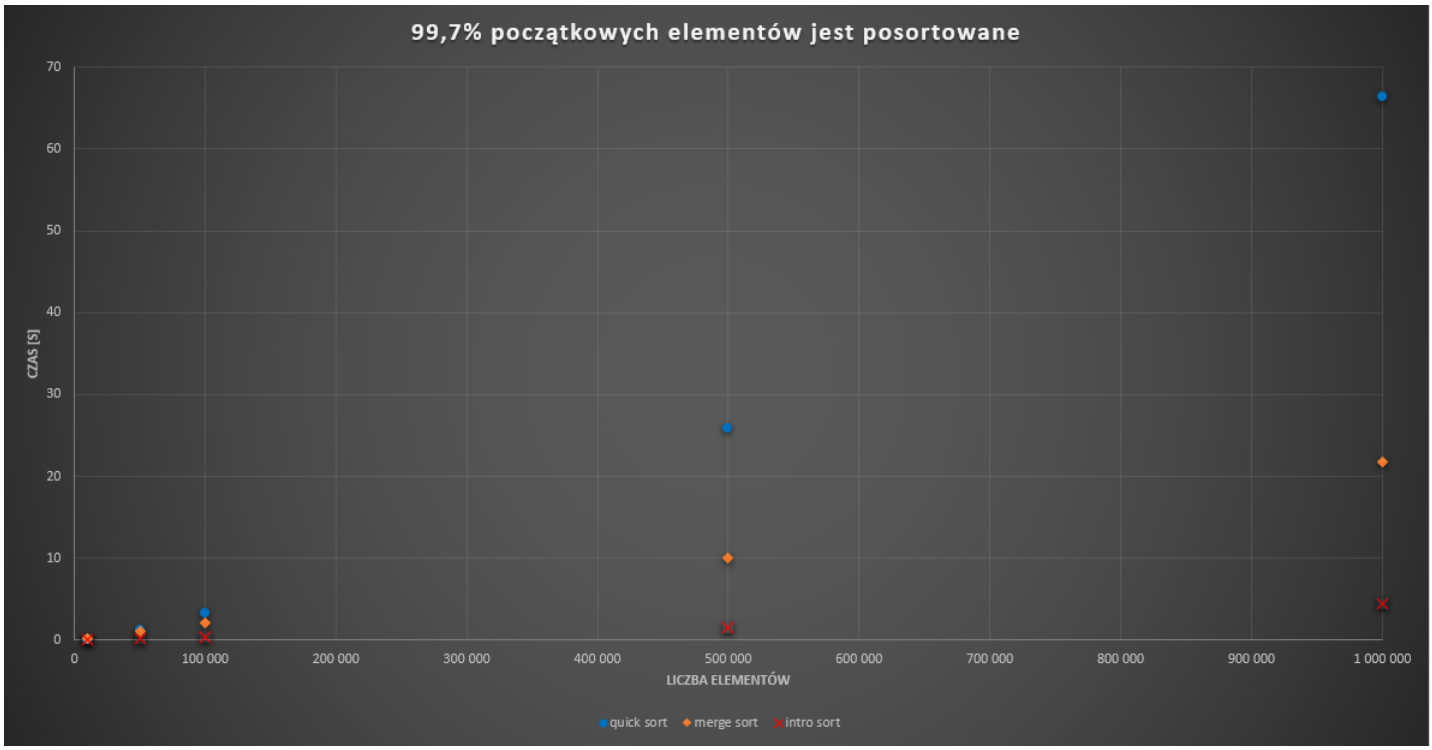
3.6 99 % początkowych elementów jest posortowanych.

99% początkowych elementów jest posortowane			
Rozmiar	quisk sort [s]	merge sort [s]	intro sort [s]
10 000	0,0882767	0,193702	0,0247137
50 000	1,4627	1,04566	0,16199
100 000	2,38553	2,08307	0,330996
500 000	14,5737	10,1251	1,69844
1 000 000	28,2292	21,7978	4,33288



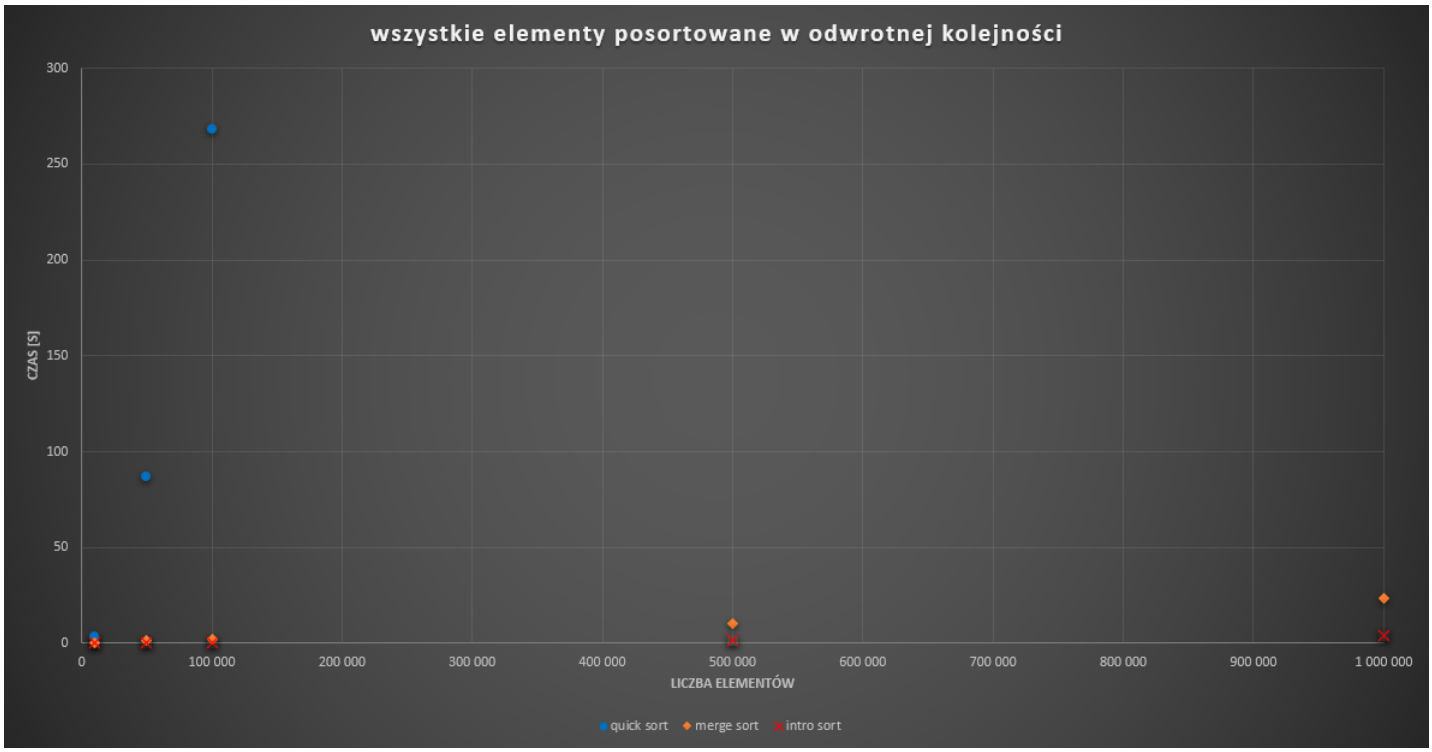
3.7 99,7 % początkowych elementów jest posortowanych.

99,7% początkowych elementów jest posortowane			
Rozmiar	quisk sort [s]	merge sort [s]	intro sort [s]
10 000	0,115086	0,195796	0,022998
50 000	1,18328	1,02712	0,145696
100 000	3,32677	2,0729	0,302142
500 000	25,9145	10,0873	1,54857
1 000 000	66,3466	21,8296	4,49073



3.8 Wszystkie elementy posortowane w odwrotnej kolejności.

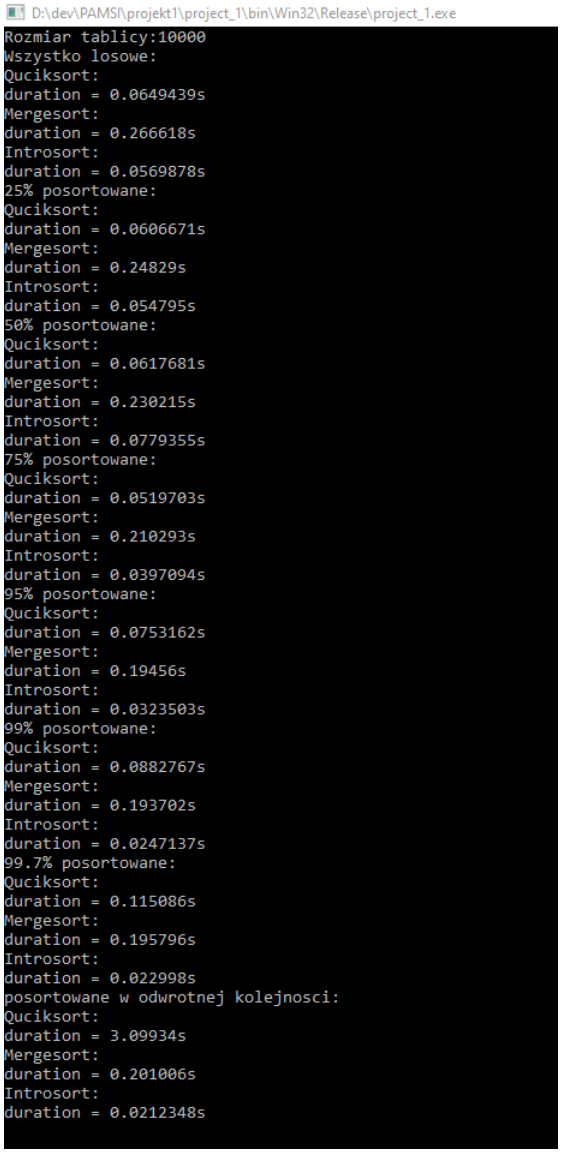
wszystkie elementy posortowane w odwrotnej kolejności			
Rozmiar	quisk sort [s]	merge sort [s]	intro sort [s]
10 000	3,09934	0,201006	0,0212348
50 000	86,6355	1,31798	0,136202
100 000	268,052	2,23399	0,271073
500 000		10,5111	1,25779
1 000 000		23,7214	4,21023

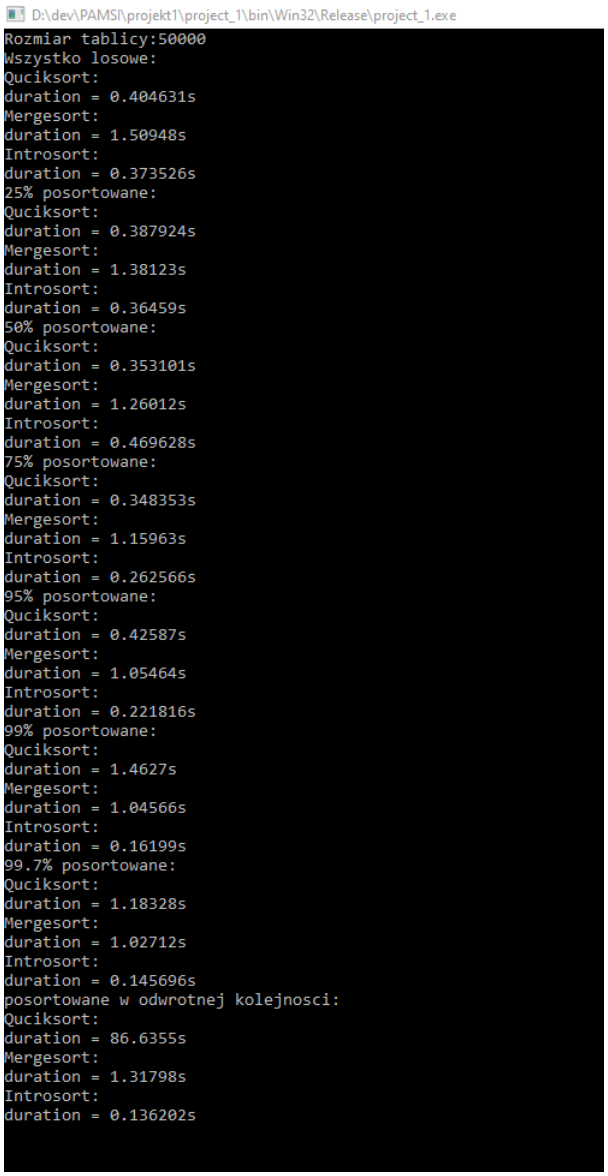


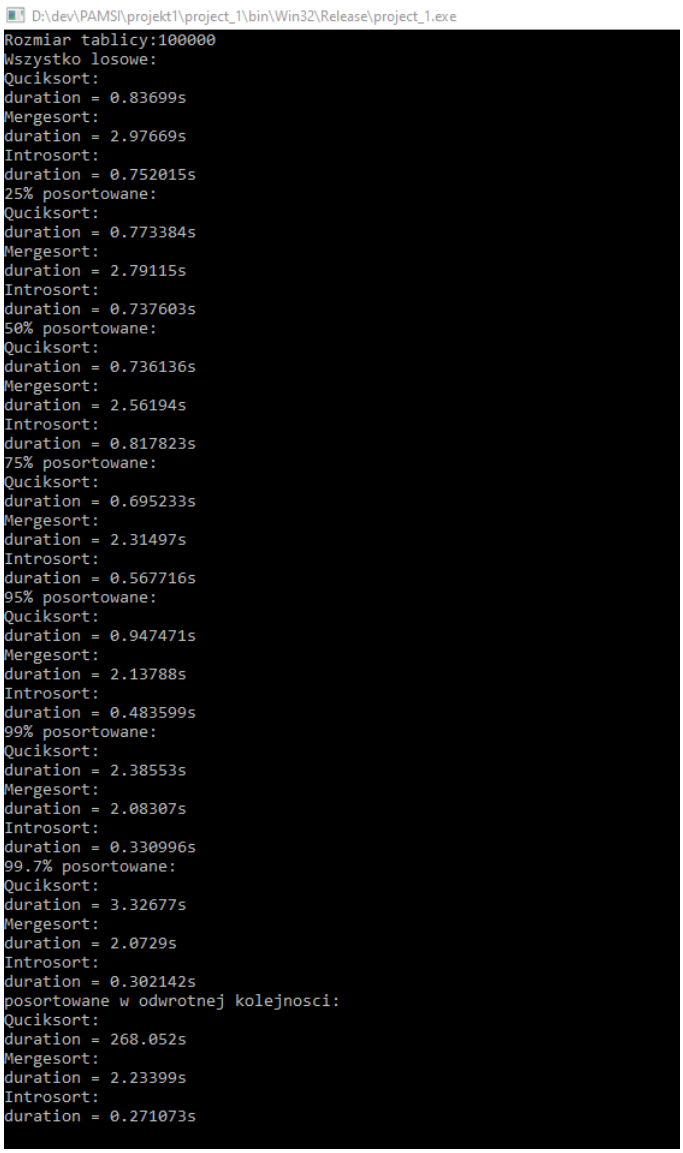
4 Podsumowanie i wnioski.

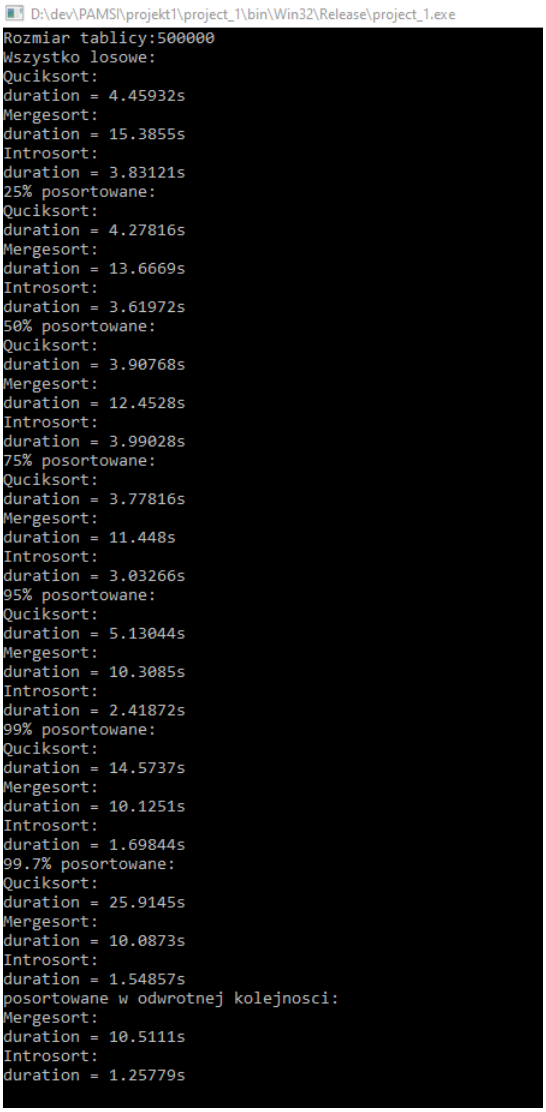
Najskuteczniejszym algorytmem zgodnie z oczekiwaniami okazał się algorytm introspektywny. Zachował on krótki czas sortowania dla każdego przypadku. Algorytm sortowania przez scalanie dla każdego przypadku utrzymywał podobny czas około 20[s]-30[s]. Algorytm sortowania szybkiego dla wszystkich elementów losowych i dla sytuacji z małym procentem już posortowanych elementów ma czasy zbliżone do algorytmu introspektywnego, niestety wraz z wsrostem już posortowanych elementów znacząco wydłuża się czas sortowania a dla sytuacji w której elementy są posortowane w odwrotenj kolejności dla dwuch największych rozmiarów tablicy algorytm się zawieszał. Ciężko określić powód problemów z działaniem algorytmu szybkiego sortowania ponieważ ma on badrdzo podobną implementacje do algorytmu sortowania introspektywnego który działa najelpiej z trzech przetestowanch algorytmów.

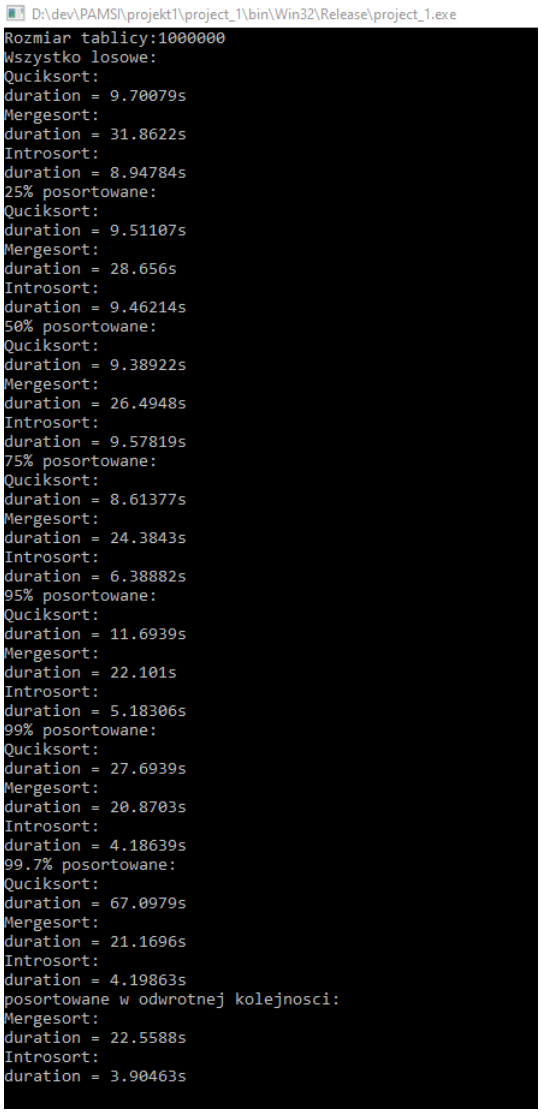
5 Załącznik.


D:\dev\PAMSI\projekt1\project_1\bin\Win32\Release\project_1.exe
Rozmiar tablicy:10000
Wszystko losowe:
Quciksort:
duration = 0.0649439s
Mergesort:
duration = 0.266618s
Introsort:
duration = 0.0569878s
25% posortowane:
Quciksort:
duration = 0.0606671s
Mergesort:
duration = 0.24829s
Introsort:
duration = 0.054795s
50% posortowane:
Quciksort:
duration = 0.0617681s
Mergesort:
duration = 0.230215s
Introsort:
duration = 0.0779355s
75% posortowane:
Quciksort:
duration = 0.0519703s
Mergesort:
duration = 0.210293s
Introsort:
duration = 0.0397094s
95% posortowane:
Quciksort:
duration = 0.0753162s
Mergesort:
duration = 0.19456s
Introsort:
duration = 0.0323503s
99% posortowane:
Quciksort:
duration = 0.0882767s
Mergesort:
duration = 0.193702s
Introsort:
duration = 0.0247137s
99.7% posortowane:
Quciksort:
duration = 0.115086s
Mergesort:
duration = 0.195796s
Introsort:
duration = 0.022998s
posortowane w odwrotnej kolejnosci:
Quciksort:
duration = 3.09934s
Mergesort:
duration = 0.201006s
Introsort:
duration = 0.0212348s


D:\dev\PAMSI\projekt1\project_1\bin\Win32\Release\project_1.exe
Rozmiar tablicy:50000
Wszystko losowe:
Quciksort:
duration = 0.404631s
Mergesort:
duration = 1.50948s
Introsort:
duration = 0.373526s
25% posortowane:
Quciksort:
duration = 0.387924s
Mergesort:
duration = 1.38123s
Introsort:
duration = 0.36459s
50% posortowane:
Quciksort:
duration = 0.353101s
Mergesort:
duration = 1.26012s
Introsort:
duration = 0.469628s
75% posortowane:
Quciksort:
duration = 0.348353s
Mergesort:
duration = 1.15963s
Introsort:
duration = 0.262566s
95% posortowane:
Quciksort:
duration = 0.42587s
Mergesort:
duration = 1.05464s
Introsort:
duration = 0.221816s
99% posortowane:
Quciksort:
duration = 1.4627s
Mergesort:
duration = 1.04566s
Introsort:
duration = 0.16199s
99.7% posortowane:
Quciksort:
duration = 1.18328s
Mergesort:
duration = 1.02712s
Introsort:
duration = 0.145696s
posortowane w odwrotnej kolejnosci:
Quciksort:
duration = 86.6355s
Mergesort:
duration = 1.31798s
Introsort:
duration = 0.136202s


D:\dev\PAMSI\projekt1\project_1\bin\Win32\Release\project_1.exe
Rozmiar tablicy:100000
Wszystko losowe:
Quciksort:
duration = 0.83699s
Mergesort:
duration = 2.97669s
Introsort:
duration = 0.752015s
25% posortowane:
Quciksort:
duration = 0.773384s
Mergesort:
duration = 2.79115s
Introsort:
duration = 0.737603s
50% posortowane:
Quciksort:
duration = 0.736136s
Mergesort:
duration = 2.56194s
Introsort:
duration = 0.817823s
75% posortowane:
Quciksort:
duration = 0.695233s
Mergesort:
duration = 2.31497s
Introsort:
duration = 0.567716s
95% posortowane:
Quciksort:
duration = 0.947471s
Mergesort:
duration = 2.13788s
Introsort:
duration = 0.483599s
99% posortowane:
Quciksort:
duration = 2.38553s
Mergesort:
duration = 2.08307s
Introsort:
duration = 0.330996s
99.7% posortowane:
Quciksort:
duration = 3.32677s
Mergesort:
duration = 2.0729s
Introsort:
duration = 0.302142s
posortowane w odwrotnej kolejnosci:
Quciksort:
duration = 268.052s
Mergesort:
duration = 2.23399s
Introsort:
duration = 0.271073s


D:\dev\PAMSI\projekt1\project_1\bin\Win32\Release\project_1.exe
Rozmiar tablicy:500000
Wszystko losowe:
Quciksort:
duration = 4.45932s
Mergesort:
duration = 15.3855s
Introsort:
duration = 3.83121s
25% posortowane:
Quciksort:
duration = 4.27816s
Mergesort:
duration = 13.6669s
Introsort:
duration = 3.61972s
50% posortowane:
Quciksort:
duration = 3.90768s
Mergesort:
duration = 12.4528s
Introsort:
duration = 3.99028s
75% posortowane:
Quciksort:
duration = 3.77816s
Mergesort:
duration = 11.448s
Introsort:
duration = 3.03266s
95% posortowane:
Quciksort:
duration = 5.13044s
Mergesort:
duration = 10.3085s
Introsort:
duration = 2.41872s
99% posortowane:
Quciksort:
duration = 14.5737s
Mergesort:
duration = 10.1251s
Introsort:
duration = 1.69844s
99.7% posortowane:
Quciksort:
duration = 25.9145s
Mergesort:
duration = 10.0873s
Introsort:
duration = 1.54857s
posortowane w odwrotnej kolejnosci:
Mergesort:
duration = 10.5111s
Introsort:
duration = 1.25779s




```
222
223     ///posortowane w odwrotnej kolejności
224     std::cout << "posortowane w odwrotnej kolejnosci:\n";
225     //quicksort
226     /*for (int i = 0; i < arr_count; i++)
227         arrs[i].random(-1);
228     timer.start();
229     for (int i = 0; i < arr_count; i++)
230         arrs[i].quicksort();
231     timer.stop();
232     std::cout << "Quciksort:\n";
233     timer.print_duration();*/
```