Projektowanie algorytmów i metody sztucznej inteligencji
Prowadzący: Mgr inż. Marcin Ochman
Wt. 15:15-16:55

Projekt 2

Autor:

Sieracka Weronika

14 kwietnia 2020

1 Wstęp:

Celem projektu było zbadanie efektywności trzech algorytmów sortujących, brana była pod uwagę szybkość wykonywania się sortowania w zależności od ilości danych do posortowania. Wpływ na nią ma czasowa złożoność obliczeniowa. Przy jej określaniu bierzemy pod uwagę operacje, które mają bezpośredni wpływ na czas wykonywania się algorytmu, czli np. operacje arytmetyczne, przypisania itp. Sortowanie było wykonywane dla 100 tablic (elementy typu całkowitoliczbowego) o następujących rozmiarach: 10 000, 50 000, 100 000, 500 000 i 1 000 000. Były one sortowane za pomocą sortowania szybkiego, sortowania przez scalanie oraz sortowanie introspektywnego. Eksperymenty były wykonane w następujących przypadkach:
• wszystkie elementy tablicy losowe,
• 25%, 50%, 75%, 95%, 99%, 99,7% początkowych elementów tablicy jest już posortowanych,
• wszystkie elementy tablicy już posortowane ale w odwrotnej kolejności.

2 Algorytmy:

2.1 Sortowanie szybkie

Działa na zasadzie „dziel i zwyciężaj". W strategii tej tablice dzielimy na dwie mniejsze tablice pod względem wybranego środkowego elementu, który nazywamy pivot. Jednkże jako pivot nie musimy wybierać element środkowego, jak u w programie, można również wybrać element pierwszy, ostatni, medianę lub losowy. Pierwsza z nich zawiera elementy mniejsze pivota,a druga elementy większe lub równe. Miejsce, w którym znajdą się elementy równe, nie ma wpływu na proces sortowania, dlatego mogą one występować w obu partycjach. Następnie oddzielnie sortujemy obie tablice sortujemy oddzielnie aż do uzyskania tablic zawierających dokładnie jeden element, który nie musimy już sortować.
Zalety:
• nie potrzebujemy tworzyć dodatkowych tablic,
• łatwy w implementacji,
• złożoność czasowa jest rzędu O(n logn)
• dobrze współpracuje z różnymi typami danych
Wady:
• jest niestabilny
• jest wrażliwy na błędy w implementacji
• w sytuacji pesymistycznej złożoność może wynosić $O(n^2)$
Złożoność obliczeniowa:
Najlepszy przypadek: O(n logn)
Standardowy przypadek: O(n logn)
Najgorszy przypadek: $O(n^2)$
Jeśli chcemy mieć pewność wykonania sortowania w czasie nie dłuższym niż $O(n \log_2 n)$, należy posiadać "dobrą" medianę jako elementu dzielącego posortowaną tablicę. Wtedy pesymistyczne oszacowanie złożoności zrówna się z optymistycznym.

2.2 Sortowanie przez scalanie:

Działa na zasadzie „dziel i zwyciężaj",tak jak opisane wcześniej sortowanie szybkie.
Wyróżnić możemy trzy kroki:
• Podział tablicy na dwie równe części,
• Podział tablic az do uzyskania tablic jedno elementowych,
• Połączenie posortowanych podciągów w jeden posortowany ciąg.

Zastosowanie:
Najczęściej stosowane przy danych dostępnych sekwencyjnie, na przykład w postaci listy jednokierunkowej albo pliku sekwencyjnego.
Złożoność obliczeniowa:
Najlepszy przypadek: O(n logn)
Standardowy przypadek: O(n logn)
Najgorszy przypadek: O(n logn)
Złożoność pamięciowa: O(n)
Zalety:
• stabilność,
• wydajność,
• prostota implementacji,
• algorytm sortuje zbiór n-elementowy w czasie proporcjonalnym do liczby nlogn bez względu na rodzaj danych wejściowych.
Wady:
• podczas scalania potrzebny jest dodatkowy obszar pamięci przechowujący kopie podtablic do scalenia

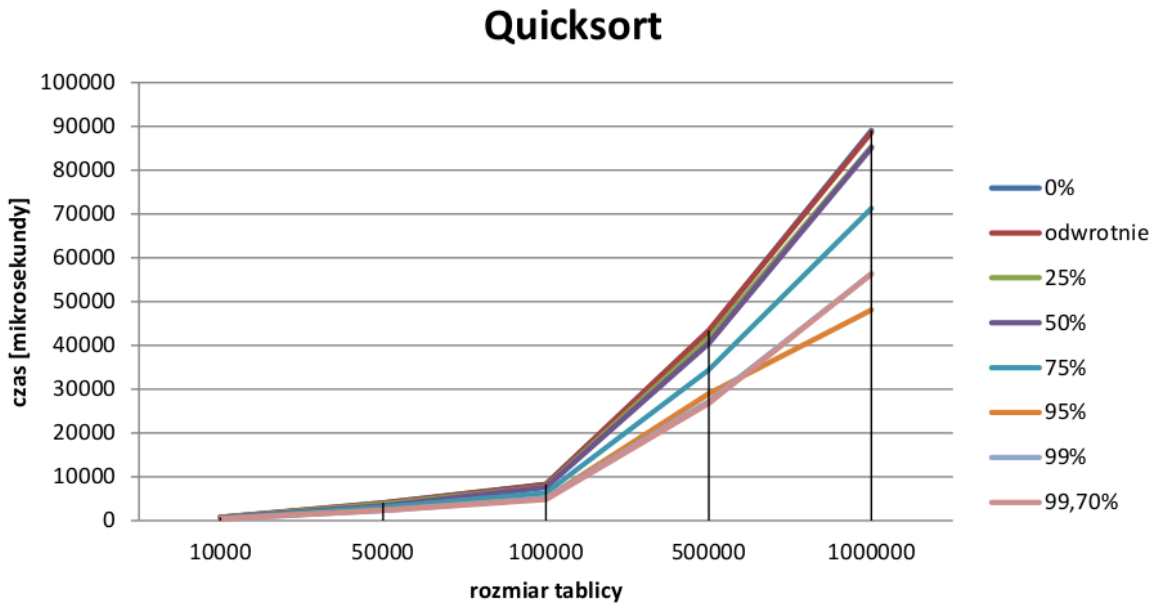
2.3 Sortowanie introspektywne:

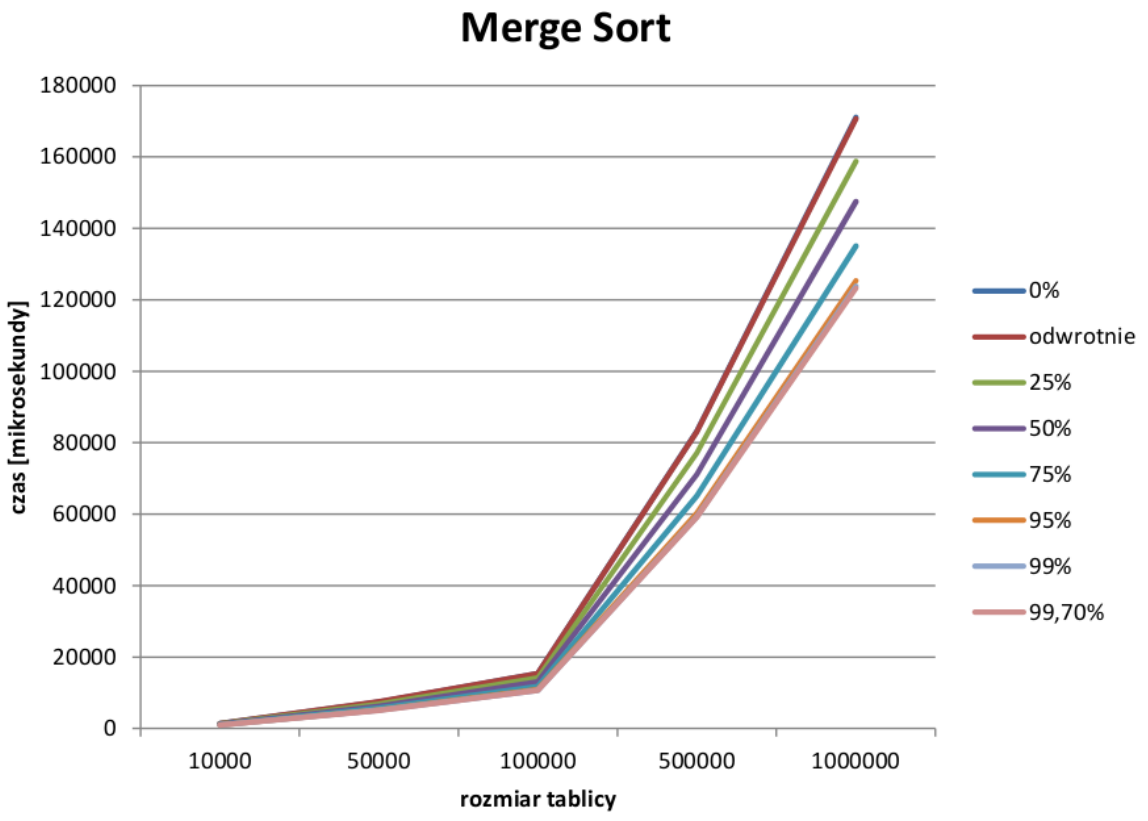
Metoda hybrydowa, ponieważ składa się z sortowania szybkiego i sortowania przez kopcowanie. Sortowanie introspektywne pozwala uniknąć najgorszego przypadku dla sortowania szybkiego.
Sortowanie przez kopcowanie:
Z sortowanej tablicy tworzony jest kopiec binarny. Jeśli przyjmiemy sortowanie w porządku niemalejącym to w korzeniu kopca znajdzie się element największy. Jeśli odwrotnie to element najmniejszy. Element ten zamieniany jest z ostatnim elementem kopca. Elementy, które są przenoszone na koniec tworzą część posortowaną i nie wchodzą już w skład kopca. Następnie przywracane są własności kopca. Dopóki wszystkie elementy nie zostaną posortowane powyższe kroki są powtarzane.
Sortowanie przez wstawianie (użyte u mnie w kodzie):
Tworzymy zbiór elementów posortowanych i przenosimy do niego dowolny element ze zbioru nieposortowanego.
Bierzemy element ze zbioru nieposortowanego.
Wyciągnięty element porównujemy z kolejnymi elementami zbioru posortowanego, dopóki nie napotkamy elementu równego lub elementu większego, jeśli chcemy otrzymać ciąg niemalejący lub nie znajdziemy się na początku/końcu zbioru uporządkowanego.
Wyciągnięty element wstawiamy w miejsce, gdzie skończyliśmy porównywać.
Procedura jest powtarzana dopóki pozostają jeszcze jakieś elementy w drugiej, nieuporządkowanej części tablicy.
Złożoność obliczeniowa:
Najlepszy przypadek: O(n logn)
Standardowy przypadek: O(n logn)
Najgorszy przypadek: O(n logn)
Złożoność pamięciowa: O(n)
Algorytm Sortowania Introspektywnego potrzebuje O(log n) pamięci na stos w każdym przypadku i jest algorytmem sortującym w miejscu.
Zalety:
• nie potrzebuje dodatkowej struktury w celu posortowania,
• pozwala uniknąć najgorszego przypadku dla sortowania szybkiego.
Wady:
• jest niestabilny,

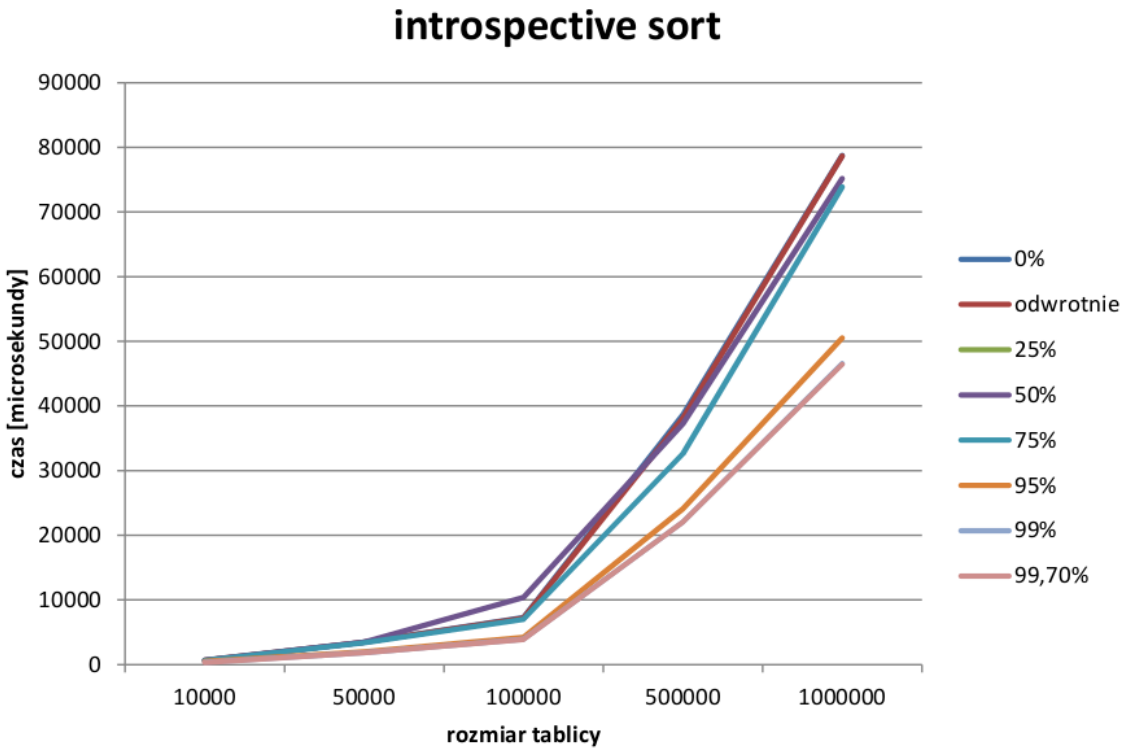
• jest trudny w implementacji.

3 Wykresy:

W legendzie zawarta jest ilość tablicy już posortowanej. Dane dotyczą obliczonego średniego czasu wykonywania jednej tablicy o danym rozmiarze.

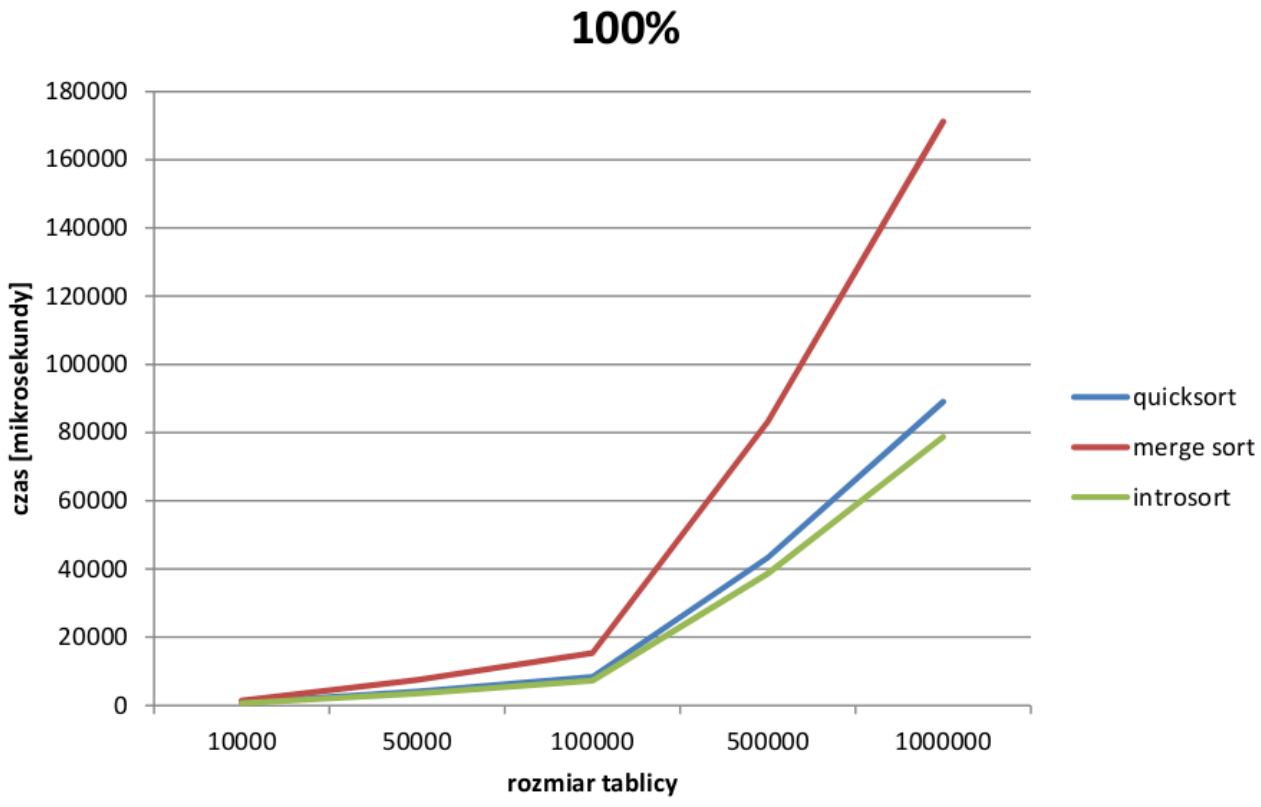




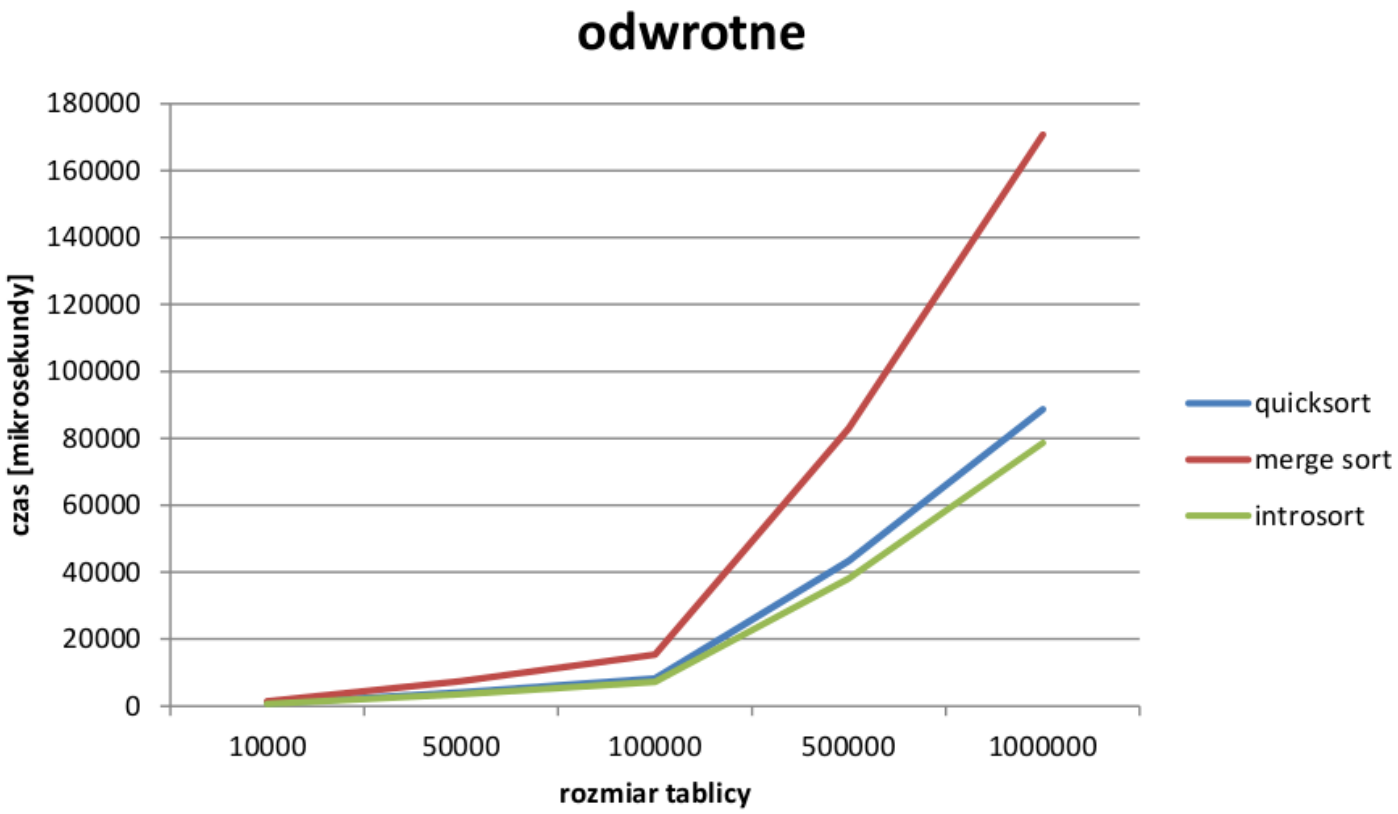


4 Wykresy porównawcze:

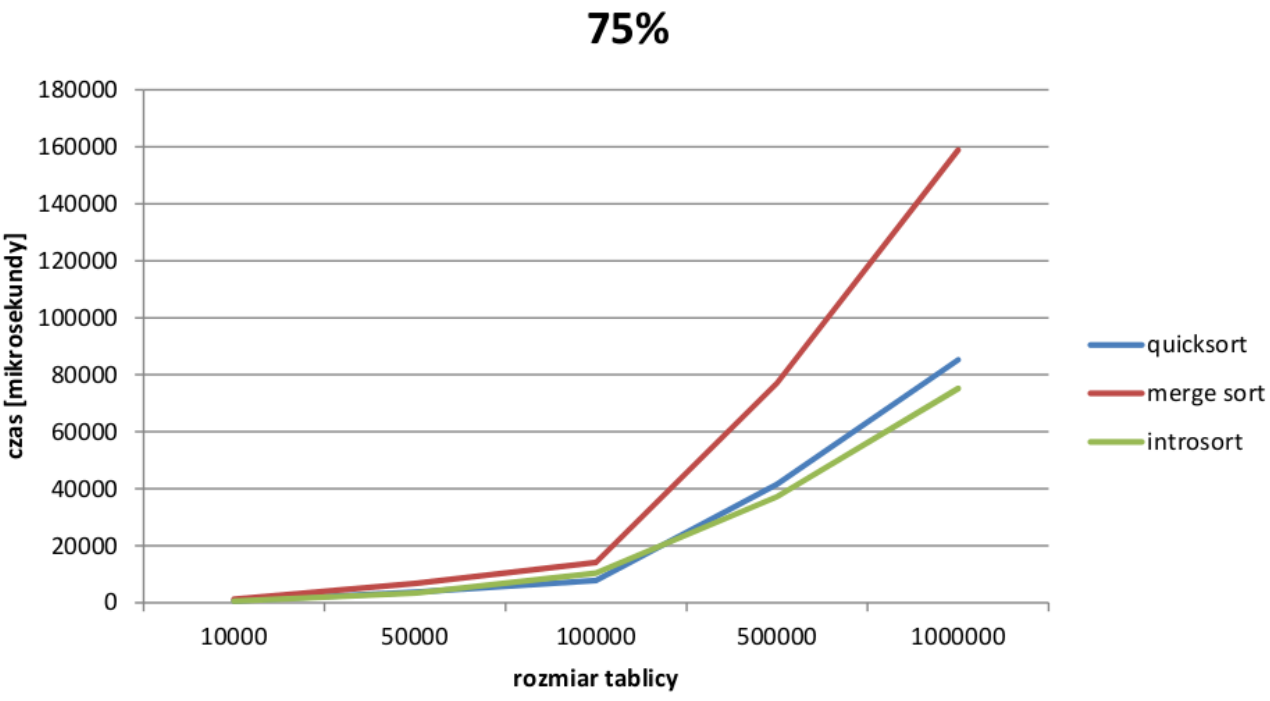
W tytule zawarta jest ilość tablicy do posortowania. Dane dotyczą obliczonego średniego czasu wykonywania jednej tablicy o danym rozmiarze.



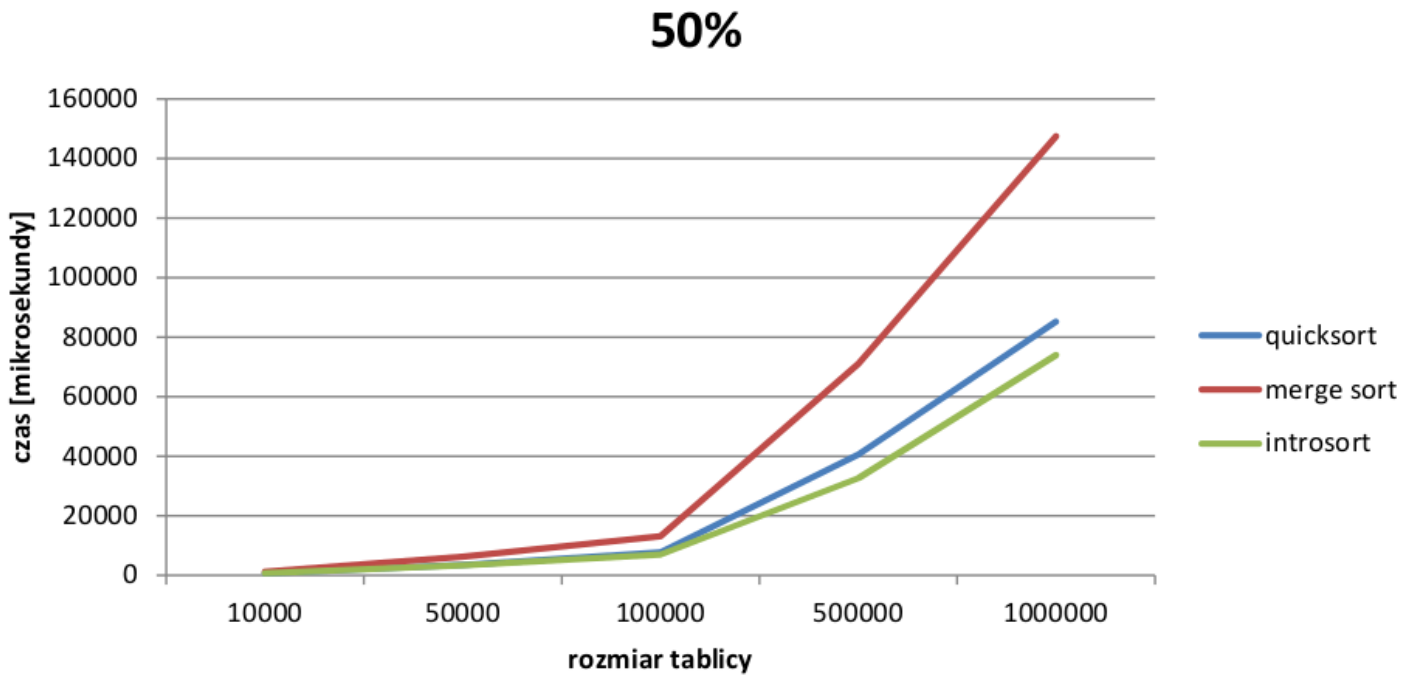
4.1 odwrotne



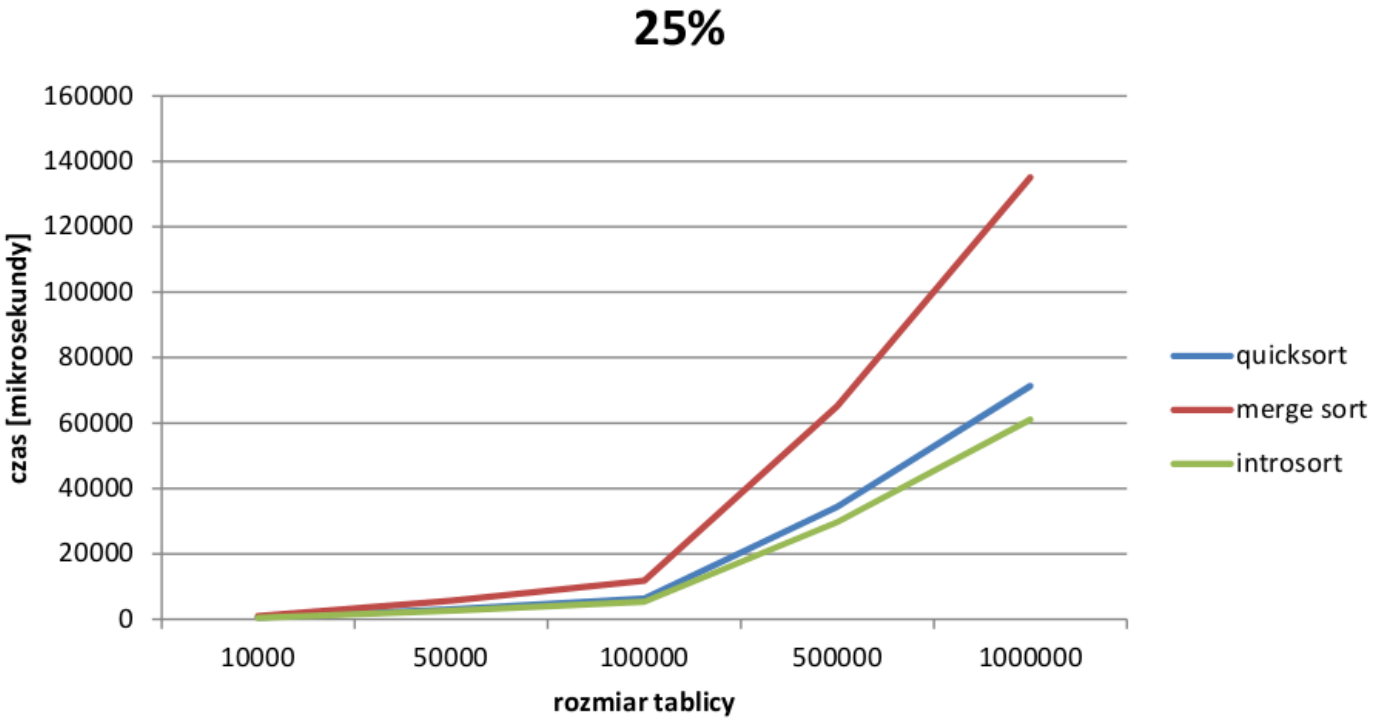
4.2 75%



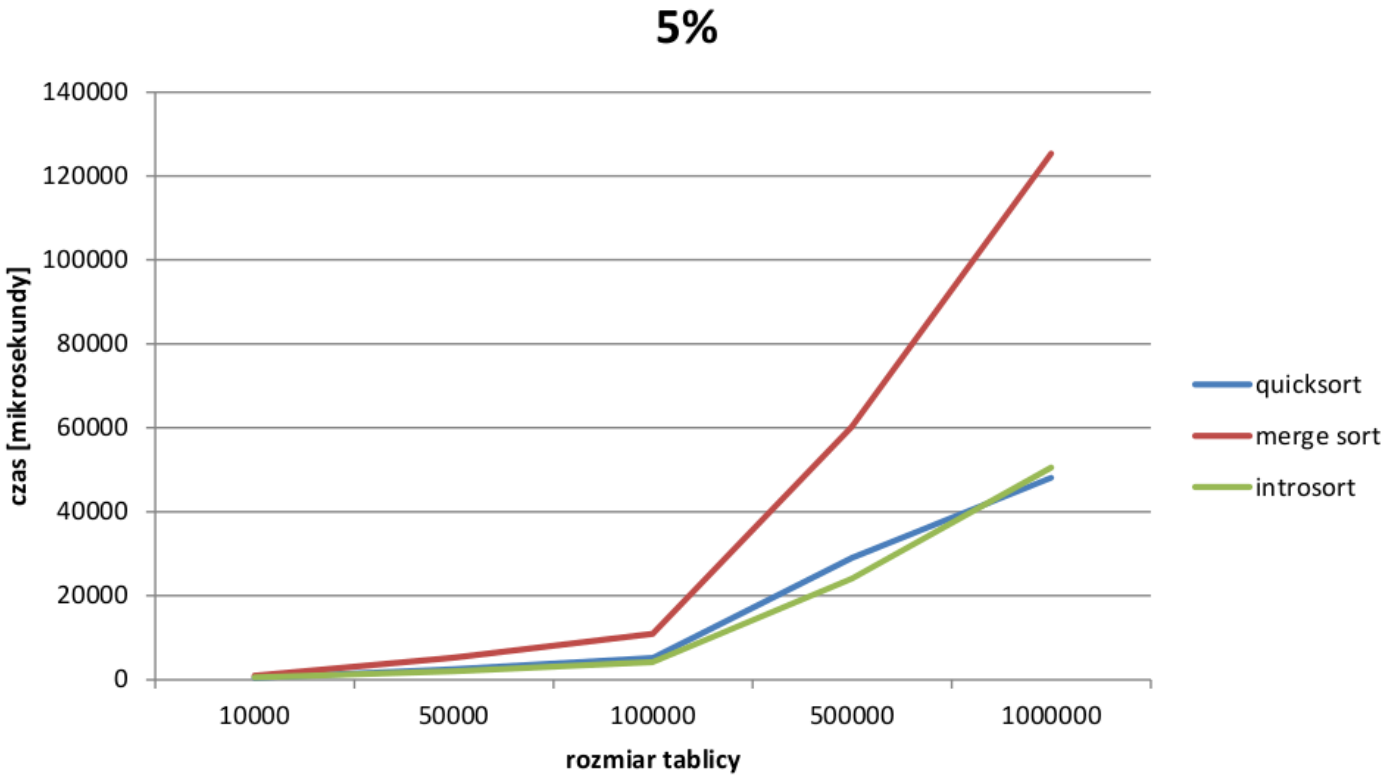
4.3 50%


50%
160000
140000
120000
100000
80000
60000
40000
20000
0
czas [mikrosekundy]
10000
50000
100000
500000
1000000
rozmiar tablicy
quicksort
merge sort
introsort

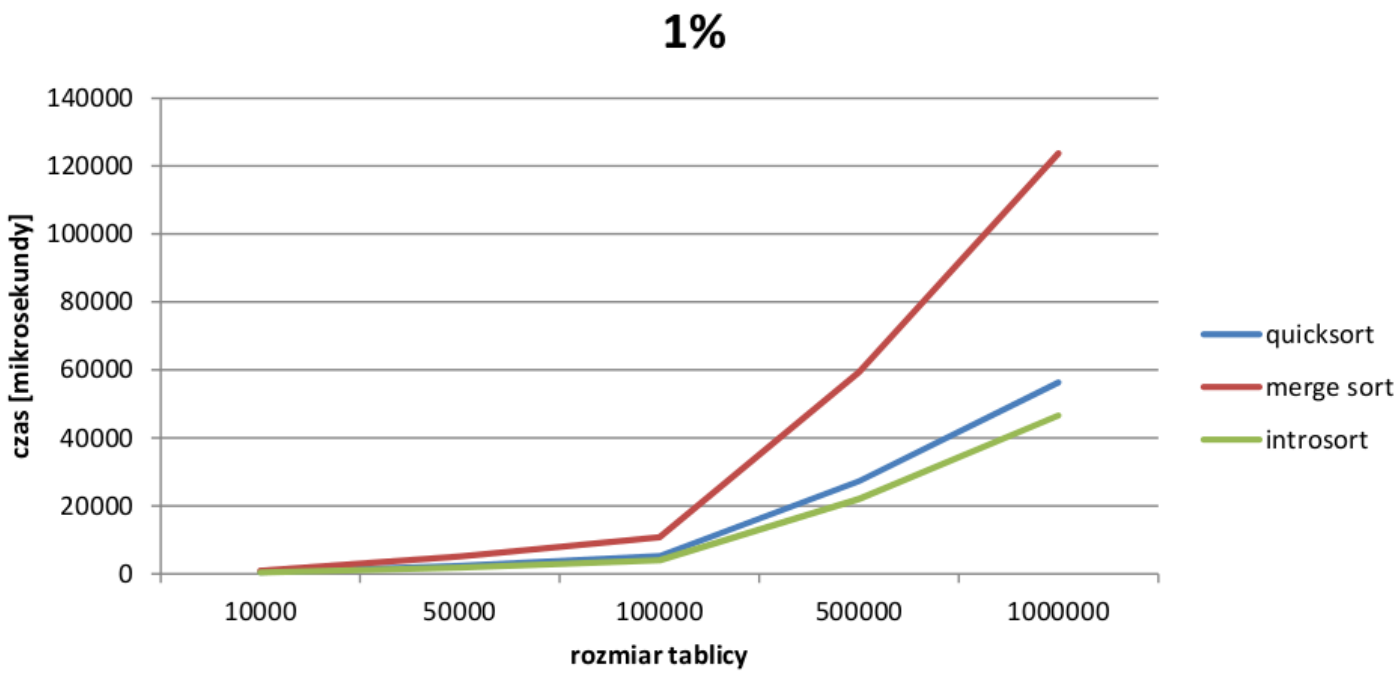
4.4 25%


25%
160000
140000
120000
100000
80000
60000
40000
20000
0
czas [mikrosekundy]
10000
50000
100000
500000
1000000
rozmiar tablicy
quicksort
merge sort
introsort

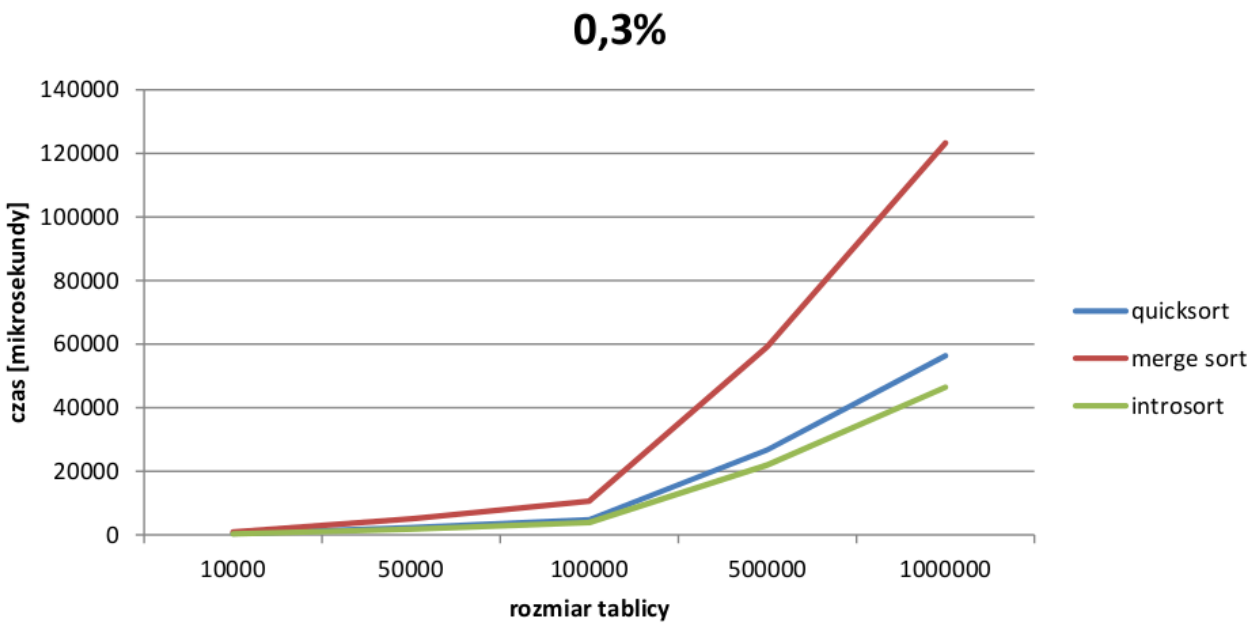
4.5 5%


5%
140000
120000
100000
80000
60000
40000
20000
0
czas [mikrosekundy]
10000
50000
100000
500000
1000000
rozmiar tablicy
quicksort
merge sort
introsort

4.6 1%


1%
140000
120000
100000
80000
60000
40000
20000
0
czas [mikrosekundy]
10000
50000
100000
500000
1000000
rozmiar tablicy
quicksort
merge sort
introsort

4.7 0.3%


0,3%
140000
120000
100000
80000
60000
40000
20000
0
czas [mikrosekundy]
10000
50000
100000
500000
1000000
rozmiar tablicy
quicksort
merge sort
introsort

5 Tabele z wynikami:

W tabeli zawarte są czasy w mikrosekundach. Dotyczą one obliczonego średniego czasu wykonywania jednej tablicy o danym rozmiarze.

5.1 Sortowanie szybkie

Wielkość tabeli	100%	odwrotne	75%	50%	25%	5%	1%	0.3%
10000	787,37	796,66	741,66	722,32	597,34	464,92	447,19	439,67
50000	4068,45	4047,77	3816,34	3397,78	3079,8	2471,11	2303,03	2301,03
100000	8292,19	8258,55	7802,28	7643,67	6367,18	5141,01	5141,01	4817,48
500000	43316,1	43383,6	41651,4	40502,8	34458,7	29018,8	27236,4	26789,4
1000000	89060,2	88643,8	85278,9	85083,2	71272,6	48107,48	56307,8	56347,8

5.2 Sortowanie przez scalanie:

Wielkość tabeli	100%	odwrotne	75%	50%	25%	5%	1%	0.3%
10000	1419,35	1422,1	1326,11	1197,81	1048,29	944,71	926,19	934,1
50000	7430,11	7480,5	6875,24	6255,49	5711,73	5200,37	5096,19	5079,57
100000	15356,8	15323,6	14163,3	13038,2	11813,4	10887	10696,5	10665,4
500000	83210,2	83006	77160	71061,5	65150,8	60260,5	59345,2	59165,4
1000000	171147	170651	158771	147484	135044	125338	123781	123199

5.3 Sortowanie introspektywne:

Wielkość tabeli	100%	odwrotne	75%	50%	25%	5%	1%	0.3%
10000	674,25	671,82	654,06	630,98	502,17	496,17	351,99	325,44
50000	3477,31	3482,66	3459,52	3358,95	2591,39	1968,31	1849,96	1833,64
100000	7280,89	7182,03	10393	6983,39	5415,61	4210,94	3910,02	3897,2
500000	38641	38111	37209,5	32565,3	29689,4	24092,3	22065,8	22042,9
1000000	78738,5	78651,5	75172,7	73879,6	61069	50530,8	46559,1	46426,4

6 Wnioski:

-Z powyższych wykresów widzimy, że najmniej efektywne jest sortowanie przez scalanie, prawdopodobnie ma to związek z parametrami komputera, ponieważ testowane na innym sprzęcie daje o wiele lepsze wyniki, jest porównywalne do quicksort.
-Z powyższych wykresów widzimy, że najbardziej efektywne jest sortowanie introspektywne
-Podobne wyniki do sortowania introspektywnego daje sortowanie szybkie

7 Uwagi

Szybkość algorytmu zależy od danych dla jakich będzie ten algorytm używany. Na ogół sortowanie introspektywne jest nieco szybsze niż sortowanie szybkie, ponieważ obsługuje najgorszy przypadek złożoności czasowej dla algorytmu szybkiego.

8 Literatura:

-http://www.algorytm.org
-http://pl.wikipedia.org/wiki/Sortowanieprzezscalanie
-http://pl.wikipedia.org/wiki/Sortowanieprzezkopcowanie
-http://pl.wikipedia.org/wiki/Sortowanieprzezwstawianie
-https://www.geeksforgeeks.org/know-your-sorting-algorithm-set-2-introsort-cs-sorting-weapon/?fbclid=IwAR0SFdix3aPmbCsm2UumfFx3XZieggEVcp3LLIMoXW7Z0rfrd0-5ajJSmmQ
-https://eduinf.waw.pl/inf/alg/003sort/m0015.php http://www.ii.uni.wroc.pl/ przemka/Dydaktyka/Heapsort.pdf
-http://www.algorytm.edu.pl/algorytmy-maturalne/quick-sort.html
-http://home.agh.edu.pl/ grzesik/Informatyka/informatykaW4.pdf
-http://smurf.mimuw.edu.pl/book/export/html/316
-https://pja.mykhi.org/3sem/ASD/asdwykady/wyklad03.pdf
-http://www.algorytm.edu.pl/algorytmy-maturalne/quick-sort.html
-http://www.algorytm.org/algorytmy/