Projektowanie Algorytmów i Metody Sztucznej Inteligencji	
Temat *Projekt 1 - sortowanie*	**Termin zajęć** *Poniedziałek 18:55*
Imię, nazwisko, numer albumu *Paweł Wójcik, 259341*	**Ocena**
Wykładowca *mgr Marta Emirsajłow*	
Kod grupy *Y03-51c*	**Data złożenia sprawozdania** 28.03.2022r.

Sprawozdanie nr. 1

Spis treści

1 Wprowadzenie

Zadaniem było zaimplementować oraz przetestować pod względem wydajności zadanych algorytmów sortowania.

Analizie było poddawane 100 tablic o rozmiarach: 10 000, 50 000, 100 000, 500 000, 1 000 000 elementów.

Oprócz różnorodności ilości elementów występowały również zmienne warunki wstępnego posortowania (przed właściwym sortowaniem):

- wszystkie elementy tablicy losowe,
- 25%, 50%, 75%, 95%, 99%, 99,7% początkowych elementów tablicy posortowanych,
- tablica posortowana w odwrotnej kolejności.

2 Opis badanych algorytmów

2.1 Wprowadzenie

W trakcie opisywania algorytmów będę się posługiwał pojęciem stabilności. Stabilne sortowanie oznacza, że elementy o równej wartości będą występowały po posortowaniu w takiej samej kolejności jaką miały w zbiorze nieposortowanym.

2.2 Sortowanie przez scalanie - mergesort

W sortowaniu przez scalanie tablica jest rodzielana na jak najmniejsze części, aż do uzyskania pojedynczych elementów. Następnie oprócz ich scalania następuje etap właściwego sortowania. Jest to przykład algorytmu stosującego zasadę "dziel i zwyciężaj".

Złożoność obliczeniowa:

- Najlepszy przypadek: $O(nlogn)$
- Średni przypadek: $O(nlogn)$
- Najgorszy przypadek: $O(nlogn)$

Złożoność pamięciowa: $O(n)$
Stabilność: TAK
Zarówno średni jak i najgorszy przypadek mają taką samą złożoność obliczeniową.

2.3 Sortowanie szybkie - quicksort

Quicksort również bazuje na zasadzie "dziel i zwyciężaj", czyli tak jak w przypadku mergesorta tablica jest rodzielana na jak najmniejsze części, aż do uzyskania pojedynczych elementów.
Tutaj podobieństwo do sortowania przez scalanie się kończy. Quicksort po wybraniu pivota przenosi elementy większe na jego prawą stronę, a elementy mniejsze na lewą stronę i tak rekursywnie aż do posortowania całości.
Złożoność obliczeniowa:

- Najlepszy przypadek: $O(nlogn)$
- Średni przypadek: $O(nlogn)$
- Najgorszy przypadek: $O(n^2)$

Złożoność pamięciowa: $O(logn)$
Stabilność: NIE
Najgorszy przypadek znacznie się różni złożonością obliczeniową, dzieje się tak w przypadku nietrafnego wyboru pivota (najmniejszy lub największy element).

2.4 Sortowanie introspektywne - introsort

Introsort to algorytm hybrydowy, który łączy sortowanie szybkie, sortowanie przez kopcowanie oraz sortowanie przez wstawianie. W standardowym przypadku introsort działa jak quicksort, oprócz newralgicznych przypadków:

- gdy dozwolona głębokość wywołań rekurencyjnych jest przekroczona używany jest heapsort,
- jeśli podtablica jest mniejsza niż 16 elementowa, to korzystaj z insertionsort.

Złożoność obliczeniowa:

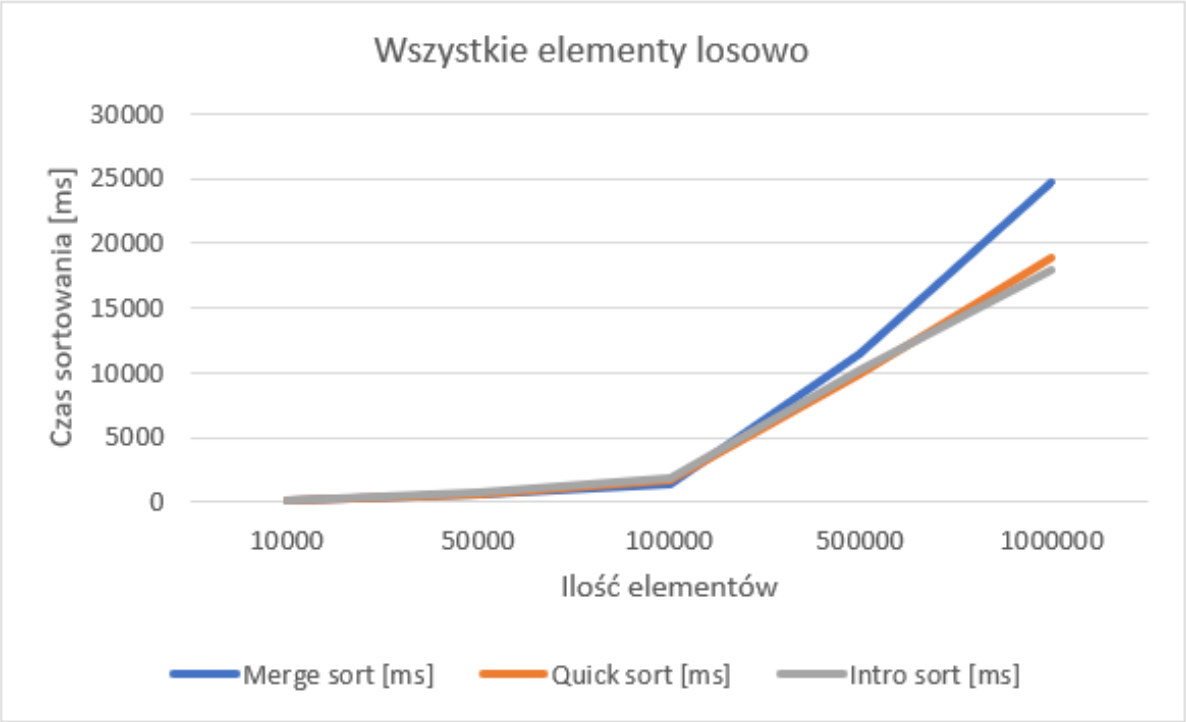
- Najlepszy przypadek: $O(nlogn)$
- Średni przypadek: $O(nlogn)$
- Najgorszy przypadek: $O(nlogn)$

Złożoność pamięciowa: $O(logn)$
Stabilność: NIE
Mimo, że introsort w dużej mierze bazuje na quicksorcie, to w najgorszym przypadku złożoność obliczeniowa jest na takim samym poziomie jak w średnim przypadku. Zawdzięcza to swojej hybrydowej strukturze.

3 Wyniki testów

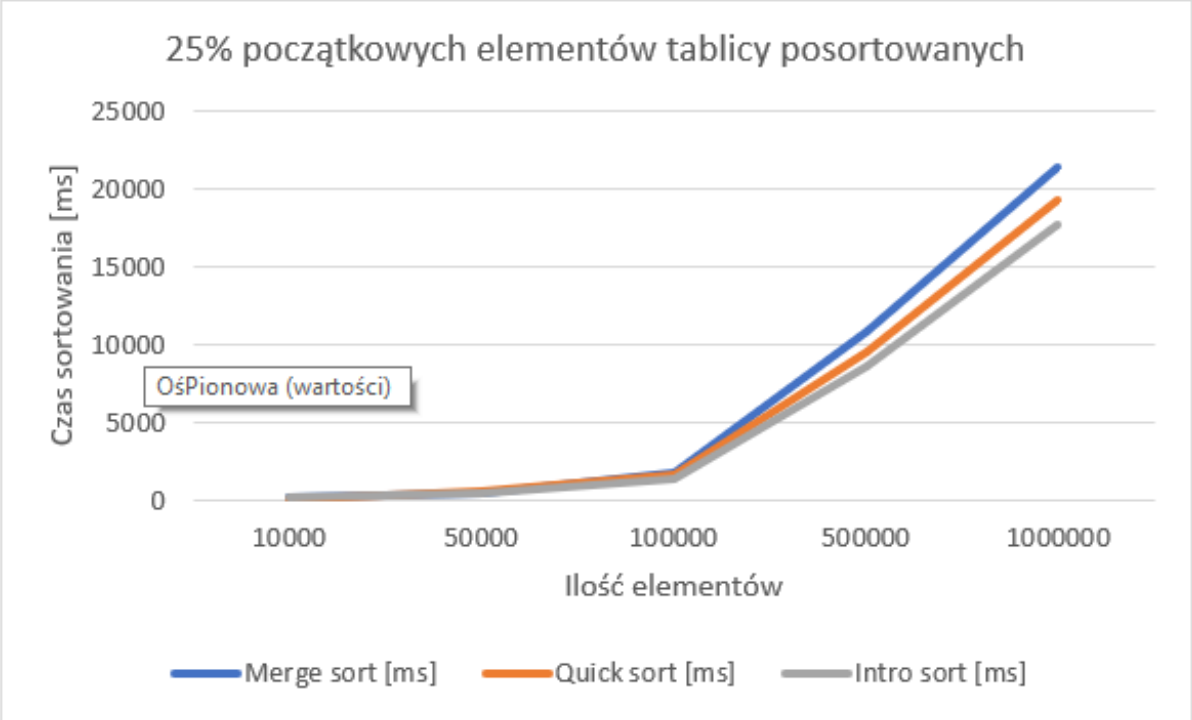
Według przewidywań sortowanie introspektywne, które jest najbardziej uniwersalne powinno być najszybsze, a sortowanie przez scalanie najwolniejsze.

Wszystkie elementy losowo					
	10000	50000	100000	500000	1000000
Merge sort [ms]	150	527	1420	11434	24708
Quick sort [ms]	154	604	1764	9878	18799
Intro sort [ms]	156	822	1914	10252	17958



Rysunek 1: Wszystkie elementy losowo

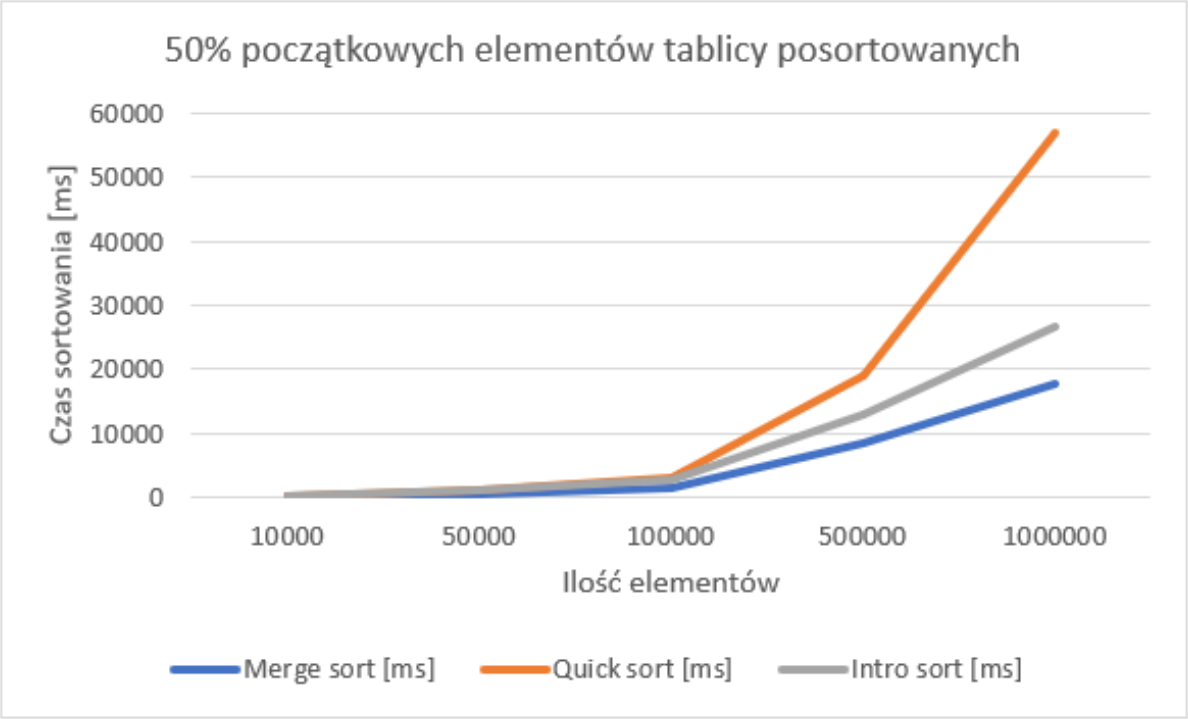
25% początkowych elementów tablicy posortowanych					
	10000	50000	100000	500000	1000000
Merge sort [ms]	101	449	1792	10898	21374
Quick sort [ms]	77	476	1574	9561	19315
Intro sort [ms]	84	417	1327	8630	17733



Rysunek 2: 25% początkowych elementów tablicy posortowanych

W powyższych dwóch przypadkach mergesort nieznacznie wolniejszy od pozostałych.

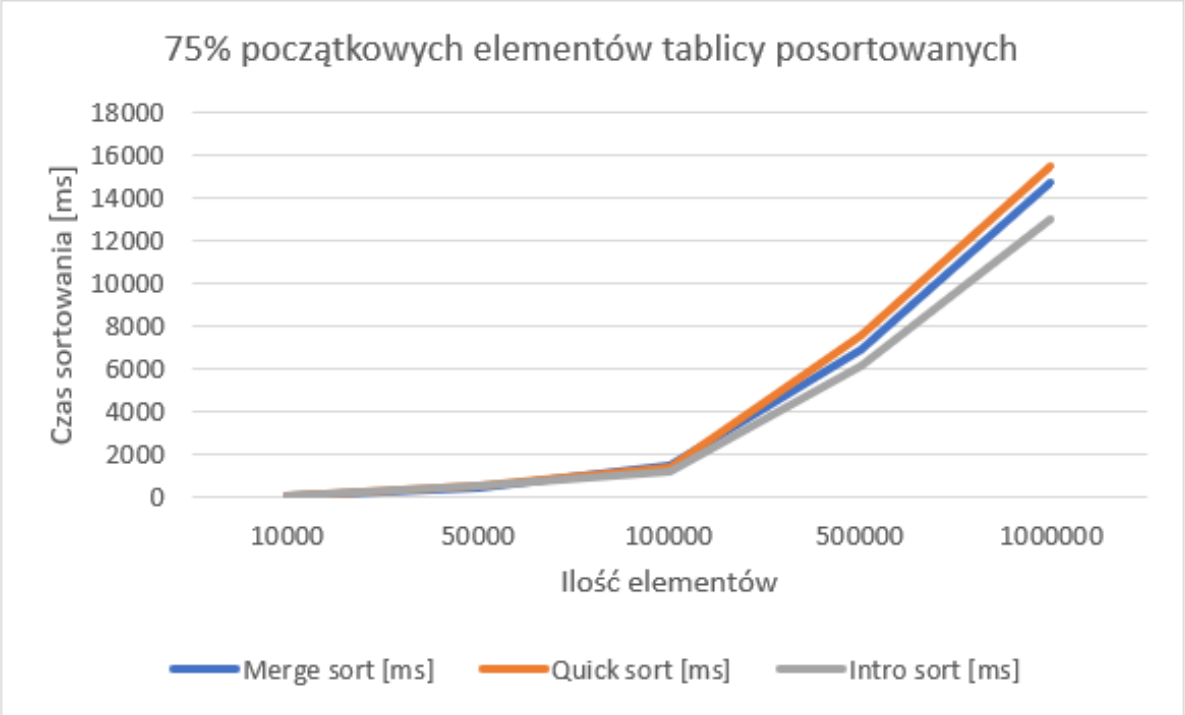
50% początkowych elementów tablicy posortowanych					
	10000	50000	100000	500000	1000000
Merge sort [ms]	53	532	1464	8613	17848
Quick sort [ms]	140	1065	3050	18872	57040
Intro sort [ms]	74	992	2740	12939	26549



Rysunek 3: 50% początkowych elementów tablicy posortowanych

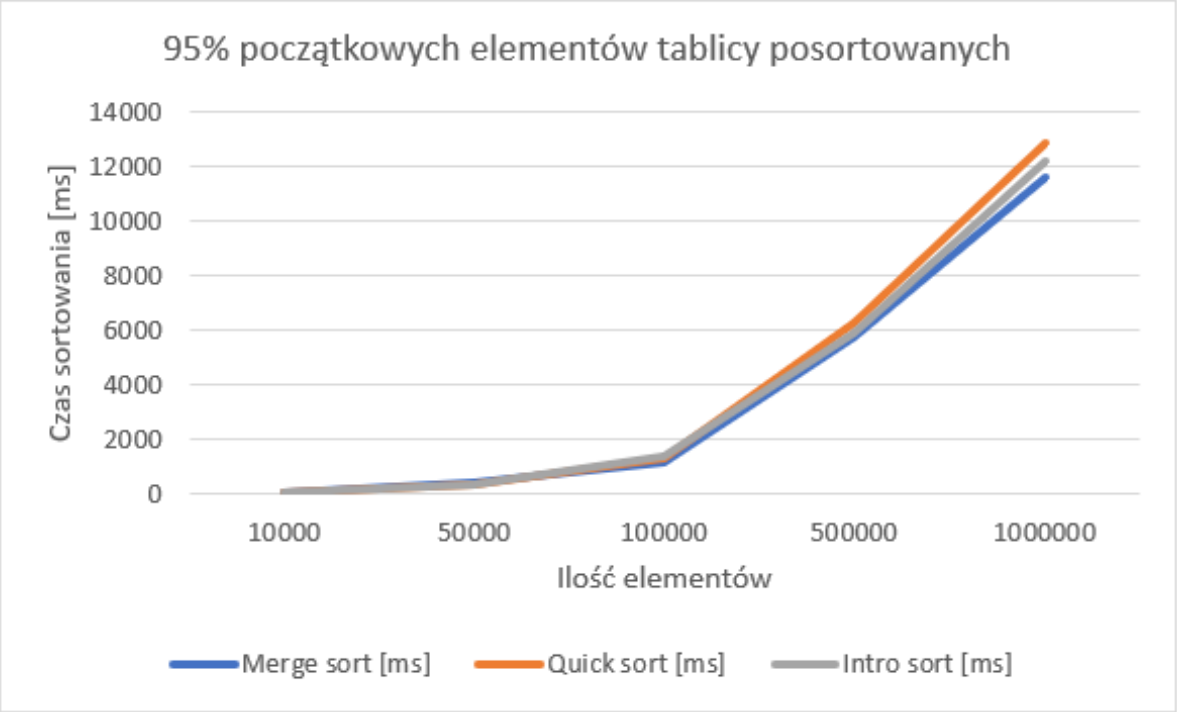
Tutaj widzimy drastyczny spadek wydajności quicksorta. Jest to spowodowane tym, że jest wybierany pivot, a ta sytuacja jest najmniej korzystna dla tego sposobu. Również introsort bazujący na powyższym zmniejszył swoją wydajność, choć dzięki temu, że jest algorytmem hybrydowym to poradził sobie z tą sytuacją znacznie lepiej niż quicksort.

75% początkowych elementów tablicy posortowanych					
	10000	50000	100000	500000	1000000
Merge sort [ms]	44	464	1474	6920	14750
Quick sort [ms]	42	500	1411	7529	15446
Intro sort [ms]	41	500	1176	6132	13003



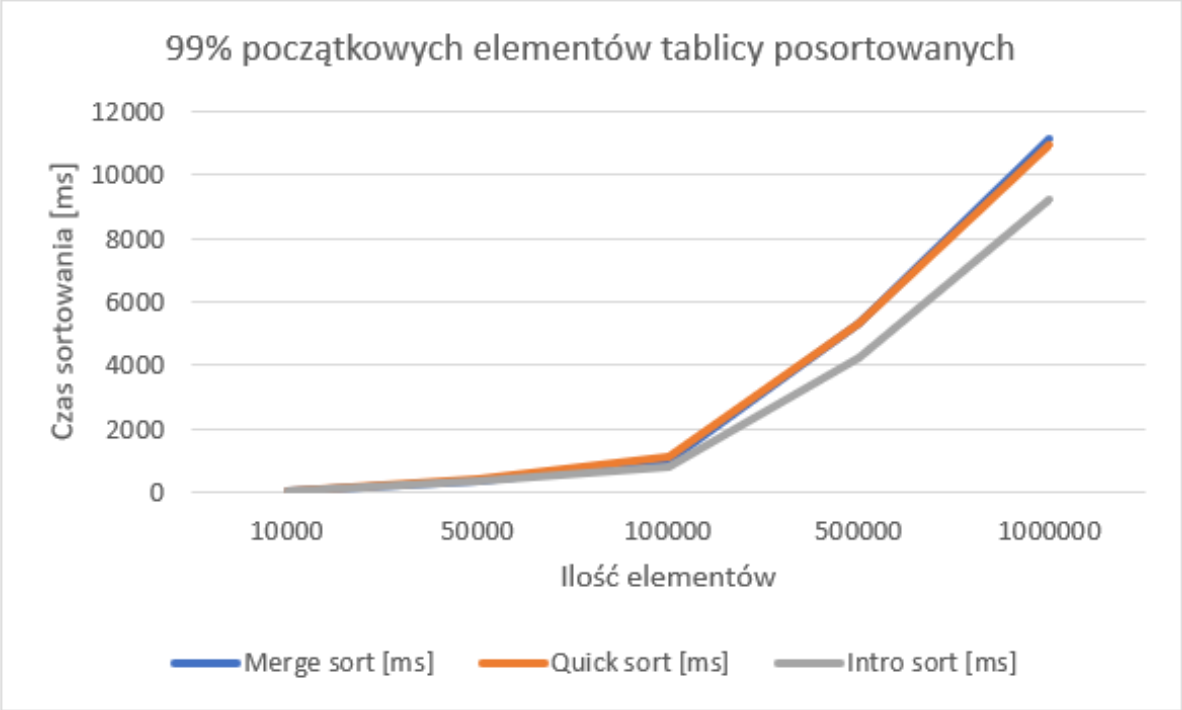
Rysunek 4: 75% początkowych elementów tablicy posortowanych

95% początkowych elementów tablicy posortowanych					
	10000	50000	100000	500000	1000000
Merge sort [ms]	23	430	1110	5750	11560
Quick sort [ms]	27	325	1278	6219	12871
Intro sort [ms]	28	335	1348	5901	12190



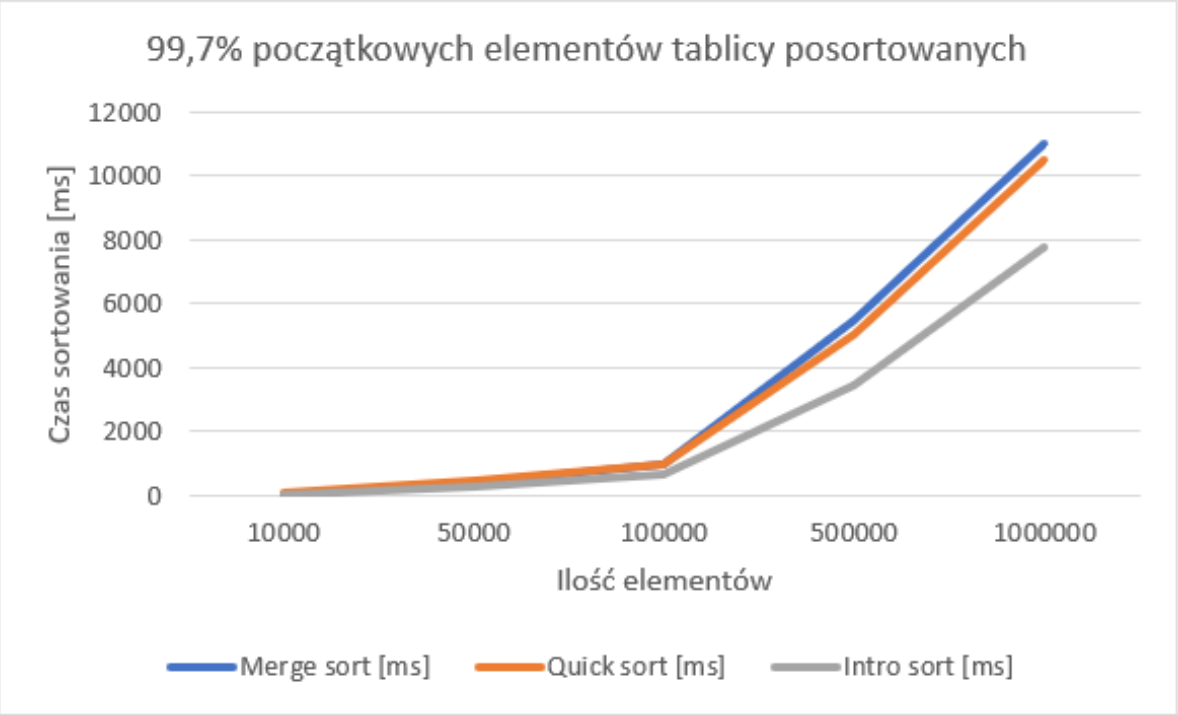
Rysunek 5: 95% początkowych elementów tablicy posortowanych

99% początkowych elementów tablicy posortowanych					
	10000	50000	100000	500000	1000000
Merge sort [ms]	24	360	983	5314	11134
Quick sort [ms]	26	397	1093	5334	10952
Intro sort [ms]	15	331	819	4276	9245



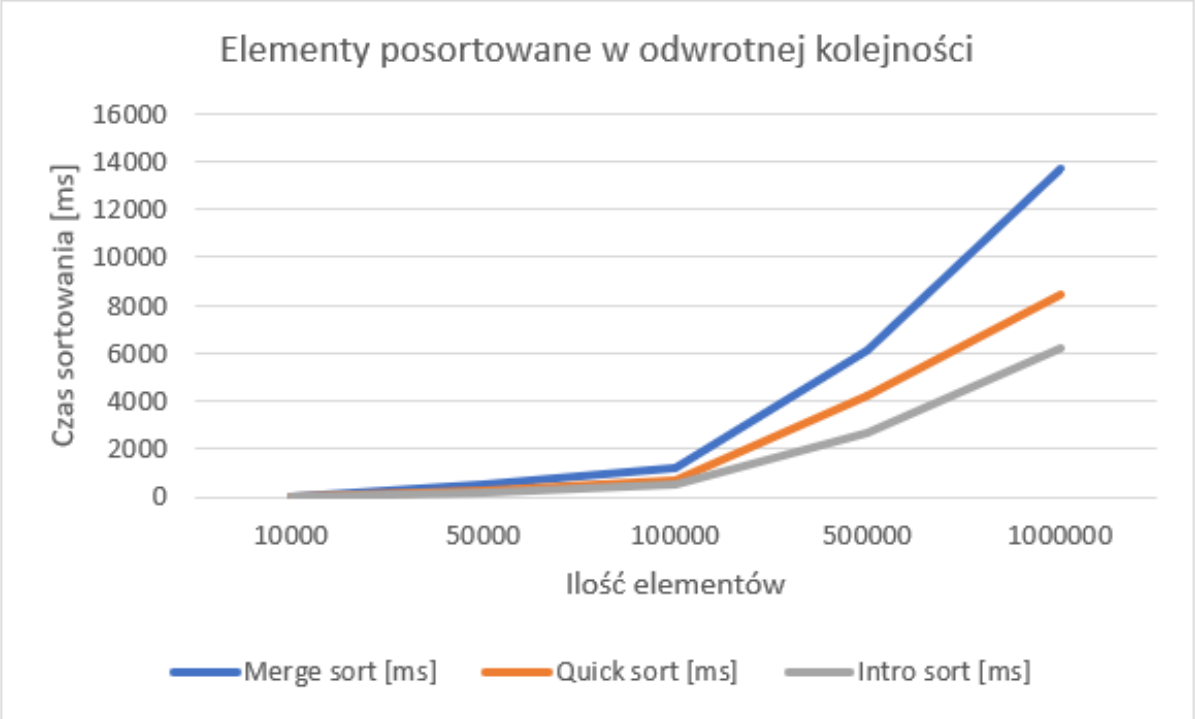
Rysunek 6: 99% początkowych elementów tablicy posortowanych

99,7% początkowych elementów tablicy posortowanych					
	10000	50000	100000	500000	1000000
Merge sort [ms]	17	425	961	5465	10986
Quick sort [ms]	55	468	984	5051	10501
Intro sort [ms]	14	262	647	3469	7737



Rysunek 7: 99,7% początkowych elementów tablicy posortowanych

Elementy posortowane w odwrotnej kolejności					
	10000	50000	100000	500000	1000000
Merge sort [ms]	23	554	1204	6102	13703
Quick sort [ms]	11	280	670	4195	8437
Intro sort [ms]	10	138	516	2645	6170



Rysunek 8: Elementy posortowane w odwrotnej kolejności

4 Wnioski

- Gdy dane są wstępnie uporządkowane, to sortowanie przez scalanie jest porównywalnie wydajny z pozostałymi algorytmami sortowania. W przypadkach całkowicie losowych danych lub uporządkowanych w odwrotnej kolejności jest zdecydowanie najgorszy.
- Tak jak przewidywano sortowanie introspektywne jest szybsze od quicksorta.
- W sytuacji 50% uporządkowanych elementów quicksort notuje drastyczny spadek wydajności, również introsort nie radzi sobie dobrze.
- Gdy popatrzymy na wartości podane w tabelach, to zauważymy, że im więcej elementów wstępnie posortowanych, tym szybsze działanie algorytmów.
- Wyniki testów są w miarę zbieżne z oczekiwaniami.

5 Bibliografia

- `http://www.algorytm.edu.pl/algorytmy-maturalne/sortowanie-przez-scalanie.html`
- `https://www.programiz.com/dsa/quick-sort`
- `https://www.programiz.com/dsa/merge-sort`

- https://www.programiz.com/dsa/insertion-sort
- https://www.programiz.com/dsa/heap-sort
- https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie
- https://pl.wikipedia.org/wiki/Sortowanie_szybkie
- https://pl.wikipedia.org/wiki/Sortowanie_introspektywne
- https://www.programmingalgorithms.com/algorithm/intro-sort/cpp/
- https://www.geeksforgeeks.org/introsort-or-introspective-sort/