Data: 25.03.2020
Imię i nazwisko: Karolina Głuszek
Nr albumu: 249034
Nazwa kursu: Projektowanie algorytmów i metody sztucznej inteligencji
Dane prowadzącego: mgr Marta Emirsajłow
Termin zajęć: piątek, 9.15

1. Wprowadzenie

Analizowane oraz zaimplementowane przeze mnie algorytmy sortowania to sortowanie przez scalanie, sortowanie szybkie oraz introspektywne. Testy efektywności algorytmów zostały wykonane dla 100 tablic zawierających elementy typu całkowitoliczbowego o rozmiarach kolejno 10 000, 50 000, 100 000, 500 000 i 1 000 000. Wyniki przedstawione na wykresach są średnią arytmetyczną 100 wykonanych pomiarów czasu. Eksperymenty zostały przeprowadzone dla tablic początkowo posortowanych w proporcjach 25%, 50%, 75%, 95%, 99%, 99,7%, a także dla wszystkich elementów tablicy posortowanych w odwrotnej kolejności i wybranych losowo.

2. Opis algorytmów

a) Sortowanie przez scalanie (merge sort)

Algorytm sortowania przez scalanie działa w oparciu o metodę „dziel i zwyciężaj". Oznacza to, że sortowana tablica najpierw jest dzielona na dwie części, a następnie każda część jest sortowana rekurencyjnie, aż do uzyskania podzbiorów wielkości 1. Następnie jest spowrotem scalana, a poszczególne elementy porównywane, co generuje posortowaną listę. Algorytm sortowania przez scalanie może zostać zobrazowany za pomocą drzewa binarnego, którego wysokość wynosi $O(\log n)$. Ilość operacji wykonywanych na każdym poziomie drzewa to $O(n)$, a zatem całkowita złożoność obliczeniowa algorytmu wynosi $O(n \log n)$ dla przypadku średniego i najgorszego.

b) Sortowanie szybkie (quicksort)

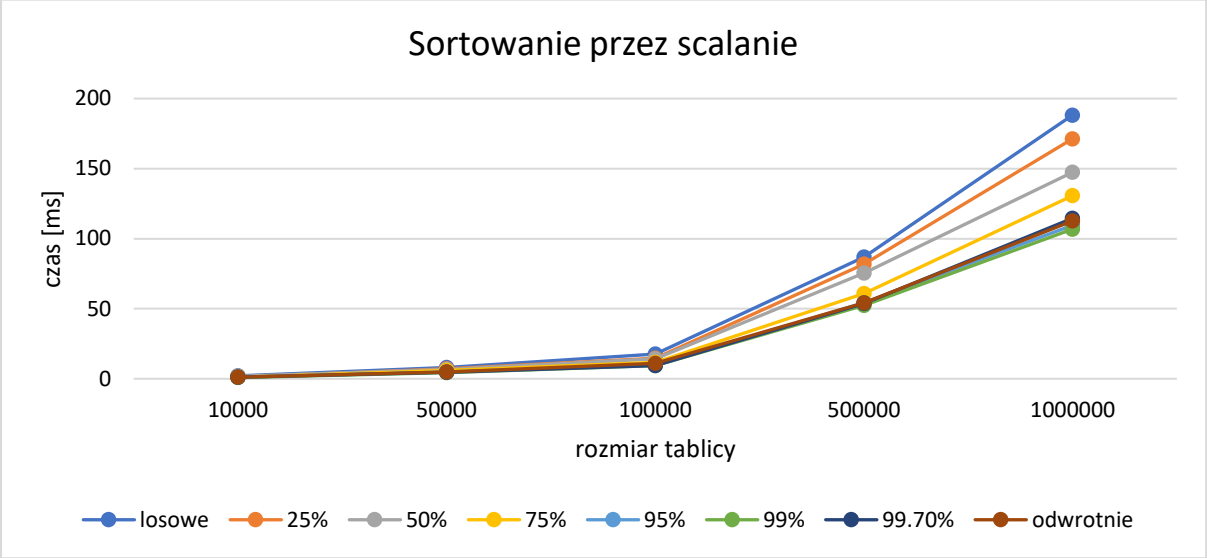
Algorytm sortowania szybkiego działa na zasadzie metody „dziel i zwyciężaj". Wybierany jest losowy element tablicy zwany piwotem, a następnie sortowana tablica zostaje podzielona na dwie podtablice elementów mniejszych oraz większych od piwota. Algorytm wywoływany jest rekurencyjnie na obu podtablicach, aż do uzyskania fragmentów jednoelementowych, które następnie zostają scalane w posortowaną tablicę. Wysokość binarnego drzewa sortowania szybkiego wynosi $O(\log n)$, a na każdym poziomie drzewa wykonuje się n obliczeń. W związku z tym średnia złożoność obliczeniowa algorytmu wynosi $O(n \log n)$. Jeżeli jednak jako piwot zawsze wybrany zostanie element skrajny (najmniejszy lub największy) pierwsza z podtablic będzie miała rozmiar 0, a druga n. W tym pesymistycznym przypadku złożoność obliczeniowa będzie proporcjonalna do sumy $n + (n-1) + \cdots + 1$ i wyniesie $O(n^2)$.

c) Sortowanie introspektywne (introsort)

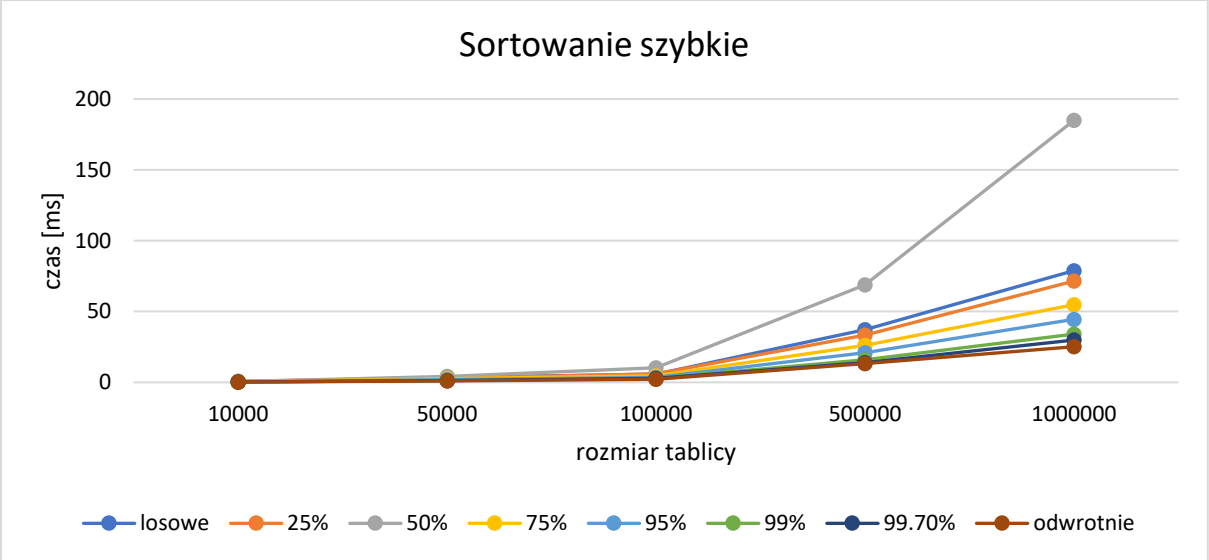
Sortowanie intospektywne jest hybrydowym algorytmem sortowania korzystającym z algorytmu sortowania szybkiego i przez kopcowanie. Dla tablicy o wielkości n wyznaczona zostaje stała o wartości $M = \log_2 n$ oznaczająca maksymalną dozwoloną głębokość wywołań rekurencyjnych. Domyślnie wykonywane jest sortowanie szybkie. Jeżeli natomiast liczba jego wywołań przekroczy stałą M na sortowanej części tablicy wykonywane jest sortowanie przez kopcowanie. Dzięki temu udaje się uniknąć pesymistycznej złożoności obliczeniowej sortowania szybkiego i średnia złożoność wynosi $O(n \log n)$. W przypadku pesymistycznym, czyli wtedy, gdy sortowanie szybkie zostaje przerwane i wywołane zostaje sortowanie przez kopcowanie mające pesymistyczną złożoność obliczeniową $O(n \log n)$, cały algorytm ma również złożoność równą $O(n \log n)$.

3. Wyniki

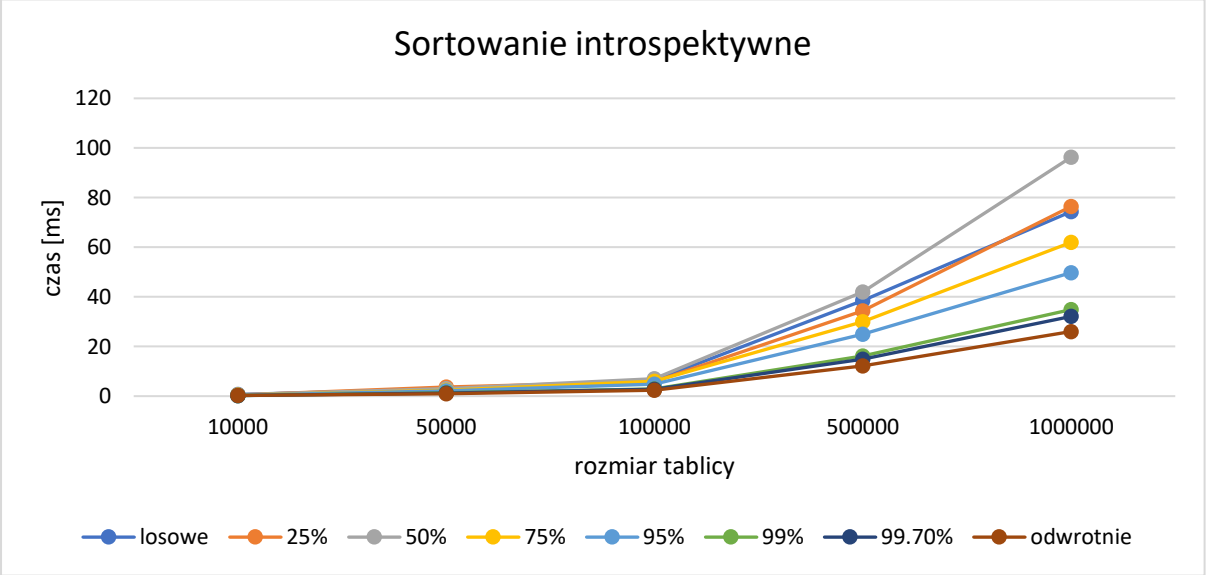
Wykres 1



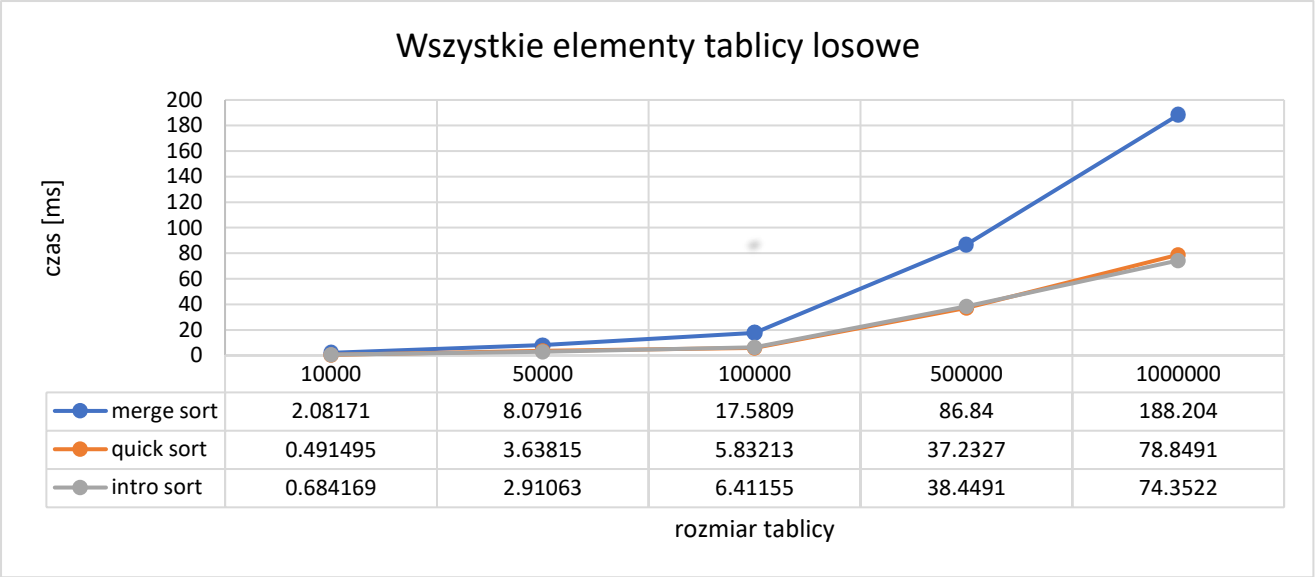
Wykres 2



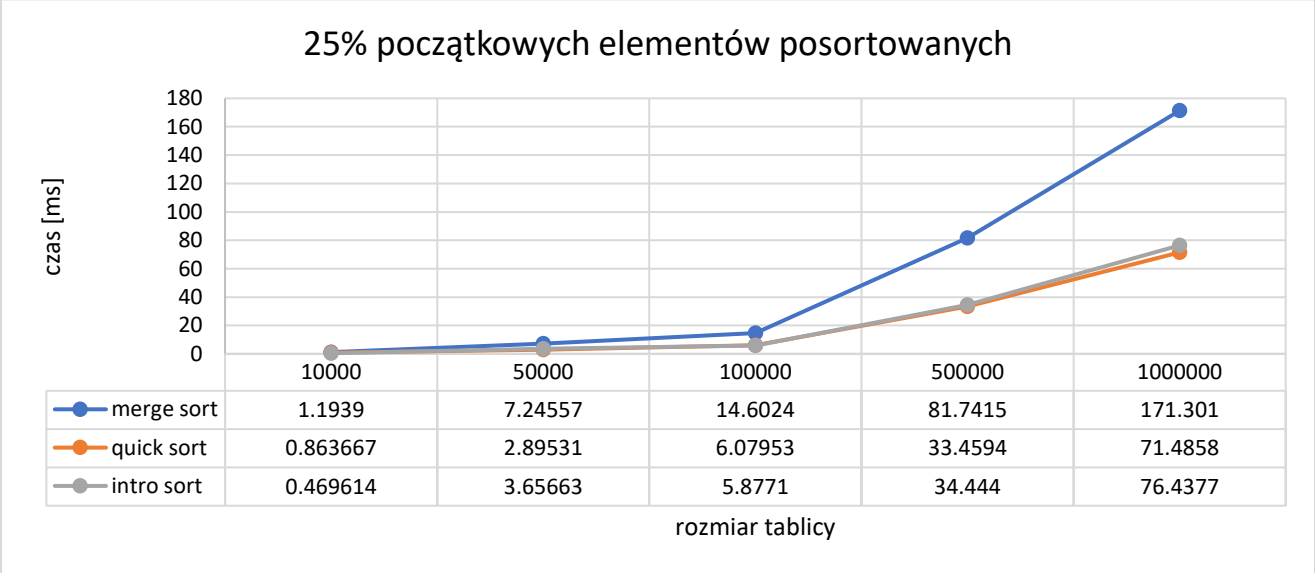
Wykres 3



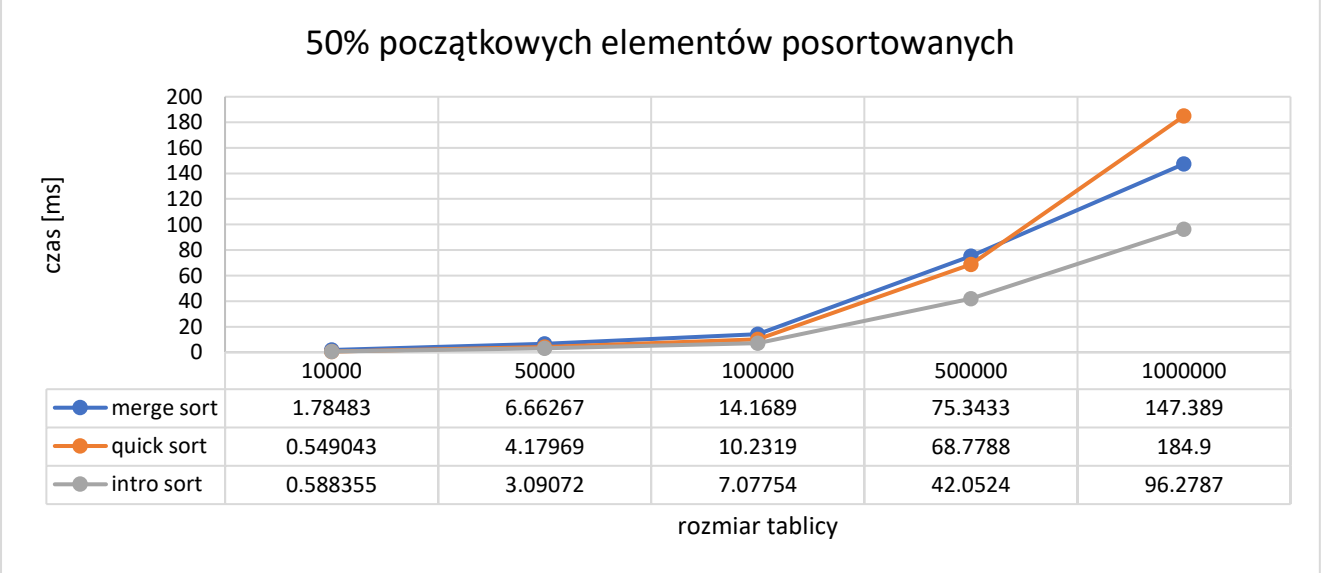
Wykres 4



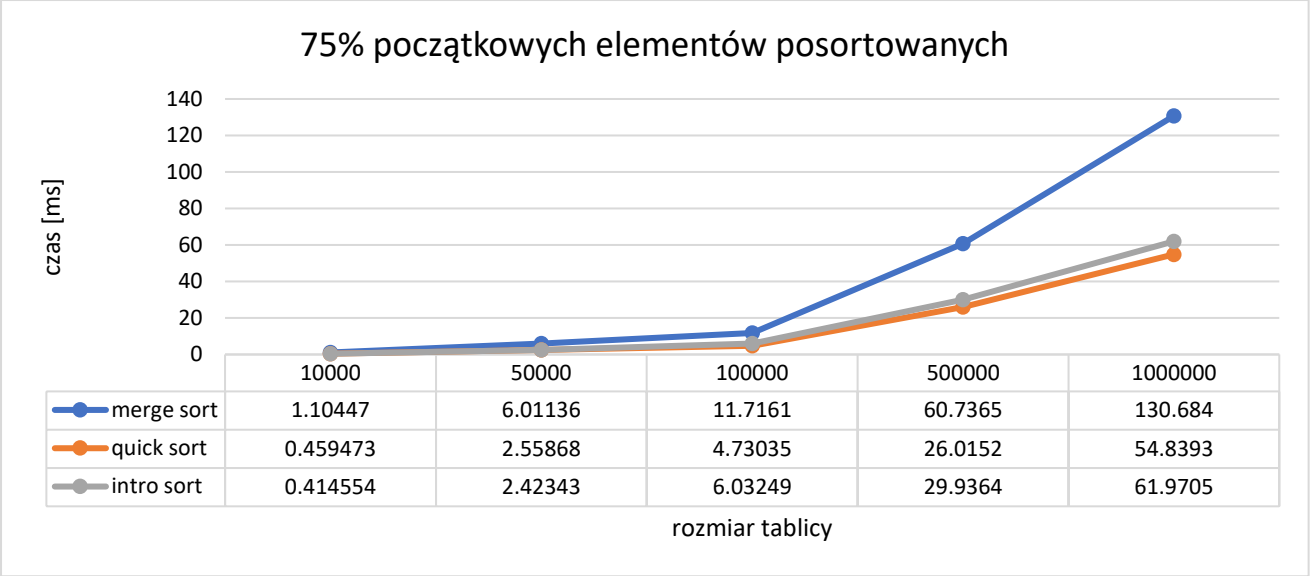
Wykres 5



Wykres 6

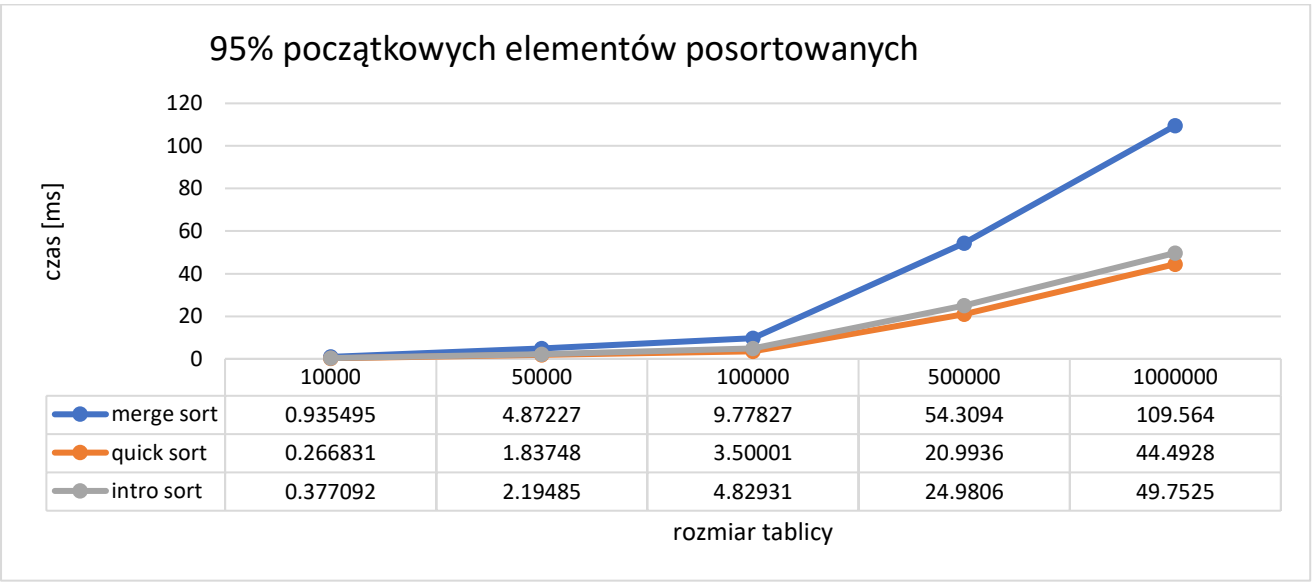


Wykres 7



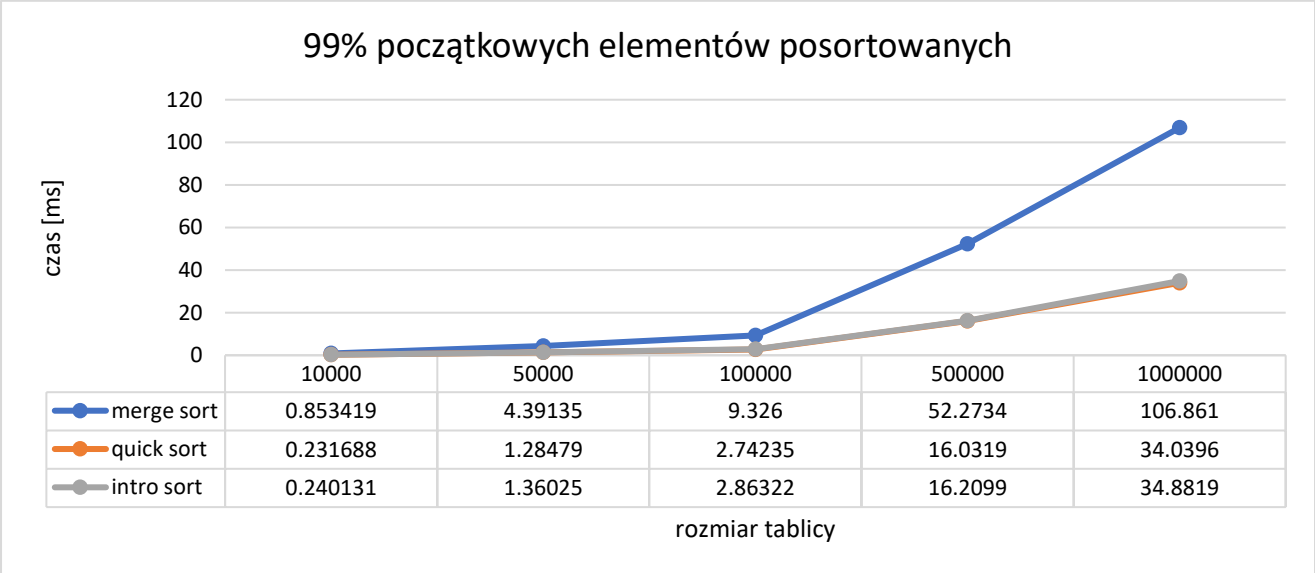
	10000	50000	100000	500000	1000000
merge sort	1.10447	6.01136	11.7161	60.7365	130.684
quick sort	0.459473	2.55868	4.73035	26.0152	54.8393
intro sort	0.414554	2.42343	6.03249	29.9364	61.9705

Wykres 8



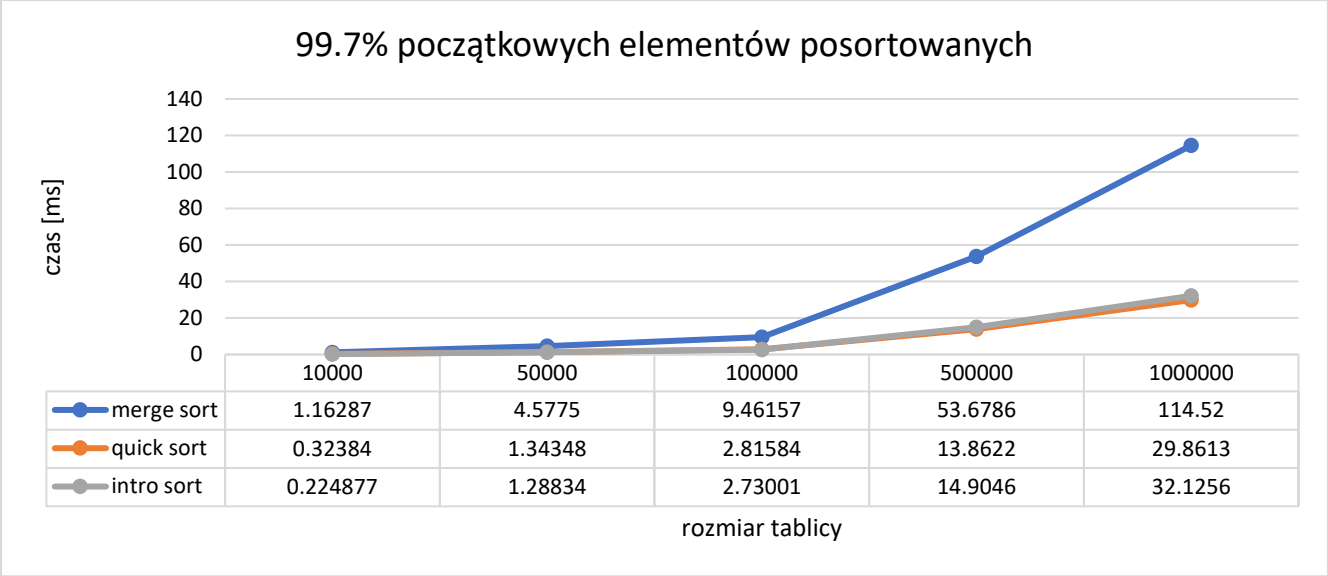
	10000	50000	100000	500000	1000000
merge sort	0.935495	4.87227	9.77827	54.3094	109.564
quick sort	0.266831	1.83748	3.50001	20.9936	44.4928
intro sort	0.377092	2.19485	4.82931	24.9806	49.7525

Wykres 9



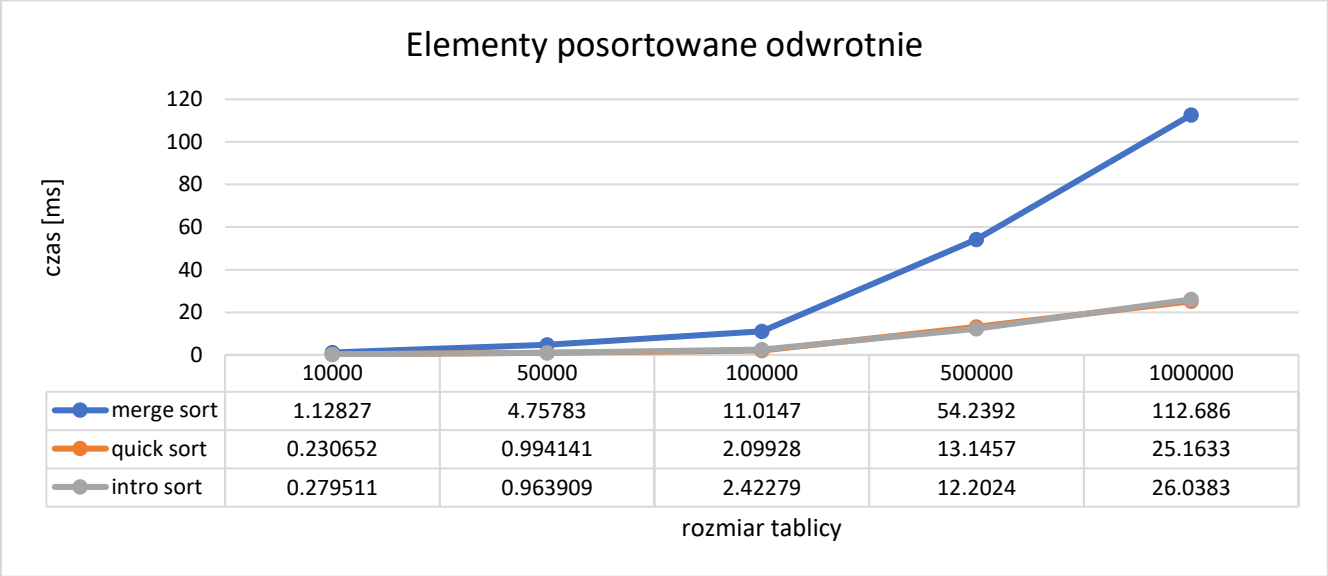
	10000	50000	100000	500000	1000000
merge sort	0.853419	4.39135	9.326	52.2734	106.861
quick sort	0.231688	1.28479	2.74235	16.0319	34.0396
intro sort	0.240131	1.36025	2.86322	16.2099	34.8819

Wykres 10



	10000	50000	100000	500000	1000000
merge sort	1.16287	4.5775	9.46157	53.6786	114.52
quick sort	0.32384	1.34348	2.81584	13.8622	29.8613
intro sort	0.224877	1.28834	2.73001	14.9046	32.1256

Wykres 11



	10000	50000	100000	500000	1000000
merge sort	1.12827	4.75783	11.0147	54.2392	112.686
quick sort	0.230652	0.994141	2.09928	13.1457	25.1633
intro sort	0.279511	0.963909	2.42279	12.2024	26.0383

4. Podsumowanie i wnioski

Na wykresach 1 – 3 przedstawiono zależności czasu sortowania tablic o różnym stopniu początkowego posortowania od ich rozmiarów. Z wykresów tych można zauważyć, że dla każdego z badanych algorytmów sortowania ich działanie było tym szybsze, im większy procent elementów tablicy został wcześniej posortowany. Jedne z najmniejszych wyników uzyskano również dla tablic początkowo posortowanych w odwrotnej kolejności. Jedyny pomiar niespełniający tej zależności to przypadek występujący na wykresie 2, gdzie dla quicksorta najwolniejsze okazało się sortowanie tablicy początkowo posortowanej w 50%. Wynika to z faktu, że zaimplementowany algorytm wybiera piwot jako środkowy element tablicy, który w tym przypadku jest również elementem skrajnym, przez co mamy do czynienia z pesymistyczną złożonością obliczeniową algorytmu wynoszącą $O(n^2)$.

Na wykresach 4 – 11 poszczególne czasy działania algorytmów sortowania są porównywane między sobą. Wśród badanych algorytmów najwolniejszy okazało się sortowanie przez scalanie. Czasy działania algorytmów sortowania szybkiego i introspektywnego dla większości przypadków pokrywają się lub są bardzo zbliżone. Jedynym wyjątkiem jest wspomniany wyżej pomiar działania sortowania szybkiego dla tablic posortowanych w 50% przedstawiony na wykresie 6, który jest większy niż dla obu pozostałych algorytmów.

5. Literatura

P. Wróblewski, „Algorytmy, struktury danych i techniki programowania”

https://en.wikipedia.org/wiki/Merge_sort

https://aquarchitect.github.io/swift-algorithm-club/Merge%20Sort/

https://en.wikipedia.org/wiki/Quicksort

https://en.wikipedia.org/wiki/Heapsort

https://en.wikipedia.org/wiki/Introsort

https://aquarchitect.github.io/swift-algorithm-club/Introsort/