Izabella Cwojdzińska 249001
Termin zajęć: czwartek 9:15 – 11:00

Projektowanie algorytmów i metody sztucznej inteligencji
Projekt 1

Wstęp:
Celem projektu była implementacja oraz przeanalizowanie wybranych trzech algorytmów pod względem efektywności. Testowanie algorytmów zostało przeprowadzone dla 100 tablic o rozmiarach 10 000, 50 000, 100 000, 500 000, 1 000 000. Były one sortowane za pomocą sortowania szybkiego, sortowania przez scalanie oraz sortowania introspektywnego. Zatem w projekcie musieliśmy wykonać szereg eksperymentów (dla elementów całkowitoliczbowego) takich jak:

- sortowanie wszystkich elementów tablicy losowej za pomocą 3 algorytmów
- sortowanie części początkowych elementów (25%, 50%, 75%, 95%, 99%, 99.7%) tablicy losowej za pomocą 3 algorytmów
- odwrotne sortowanie wszystkich elementów tablicy losowej za pomocą 3 algorytmów (tablica posortowana, miała nie posiadać elementów posortowanych)

Algorytmy:
Sortowanie szybkie (Quicksort):
Jest jednym z najszybszych algorytmów sortujących., ponieważ wykorzystuje technikę „dziel i zwyciężaj”. Oznacza to, że nasza tablica jest dzielona na dwie mniejsze tablice pod względem wybranego środkowego elementu. Zatem po jednej stronie są elementy mniejsze od naszego „środka” a po drugiej większe oraz równe. Następnie obie tablice sortowane są oddzielnie, do momentu uzyskania tablic zawierających dokładnie jeden element, którego nie trzeba już sortować.

Złożoność obliczeniowa Quicksorta:

- Najlepszy przypadek: $O(n \log n)$
- Standardowy przypadek: $O(n \log n)$
- Najgorszy przypadek: $O(n^2)$

Złożoność pamięciowa:

- $O(\log n)$ | $O(n)$

Wady oraz zalety:

Zalety:	Wady:
Posiada złożoność obliczeniową rzędu $O(n \log n)$	W najgorszym przypadku posiada złożoność obliczeniową rzędu $O(n^2)$
Jest łatwy w implementacji	Jest wrażliwy na błędy w implementacji
Dobrze współpracuje z różnymi typami danych	Działa z różną wydajnością, gdyż łatwo możemy manipulować nasz środkowy element
Nie potrzebuje dodatkowych tablic	

Sortowanie przez scalanie (Mergesort):
Jest kolejnym algorytmem opierającym się na technice „dziel i zwyciężaj” Jednakże w porównaniu do poprzednika ten algorytm w każdym przypadku osiąga złożoność obliczeniową równą *O(n logn).* Zatem sortowana tablica dzielona jest w sposób rekurencyjny na dwie podtablice aż do uzyskania tablic jednoelementowych. W kolejnym etapie dane podtablice są scalane w odpowiedni sposób, tak aby końcowo uzyskać posortowaną tablicę.

Złożoność obliczeniowa sortowania przez scalanie:
- Najlepszy przypadek: *O(n logn)*
- Standardowy przypadek: *O(n logn)*
- Najgorszy przypadek: *O(n logn)*

Złożoność pamięciowa:
- *O(n)*

Wady oraz zalety:

Zalety:	Wady:
W każdym przypadku posiada złożoność obliczeniową rzędu *O(n log n)*	Potrzebuje dodatkowy obszar pamięci, który służy do przechowywania kopi podtablic, używanych w trakcie scalania
Jest łatwy w implementacji	
Jest wydajny	
Jest stabilny	

Sortowanie introspektywne (Introspective sort):
Jest to algorytm hybrydowy, ponieważ składają się na niego różne sortowania takie jak: sortowanie szybkie oraz sortowanie przez kopcowanie. Dzięki temu sortowaniu możemy uniknąć najgorszego przypadku spotykanego w szybkim sortowaniu.

Złożoność obliczeniowa sortowania introspektywnego:
- Najlepszy przypadek: *O(n logn)*
- Standardowy przypadek: *O(n logn)*
- Najgorszy przypadek: *O(n logn)*

Złożoność pamięciowa:
- *O(log n)*

Wady oraz zalety:

Zalety:	Wady:
Nie potrzebuje dodatkowej struktury w celu posortowania	Jest niestabilny
Eliminuje najgorszy przypadek złożoności obliczeniowej występujący w Quicksorcie	Jest trudny w implementacji

Porównanie algorytmów:
Sortowanie szybkie (Quicksort) vs Sortowanie przez scalanie (Mergesort):

	Sortowanie szybkie (Quicksort)	**Sortowanie przez scalanie (Mergesort)**
Najlepszy przypadek złożoności obliczeniowej	*O(n log n)*	*O(n log n)*
Standardowy przypadek złożoności obliczeniowej	*O(n log n)*	*O(n log n)*
Najgorszy przypadek złożoności obliczeniowej	$O(n^2)$	*O(n log n)*
Złożoność pamięciowa:	*O(log n) | O(n)*	*O(n)*
Stabilność	Niestabilny	Stabilny
Implementacja	Łatwa	Łatwa

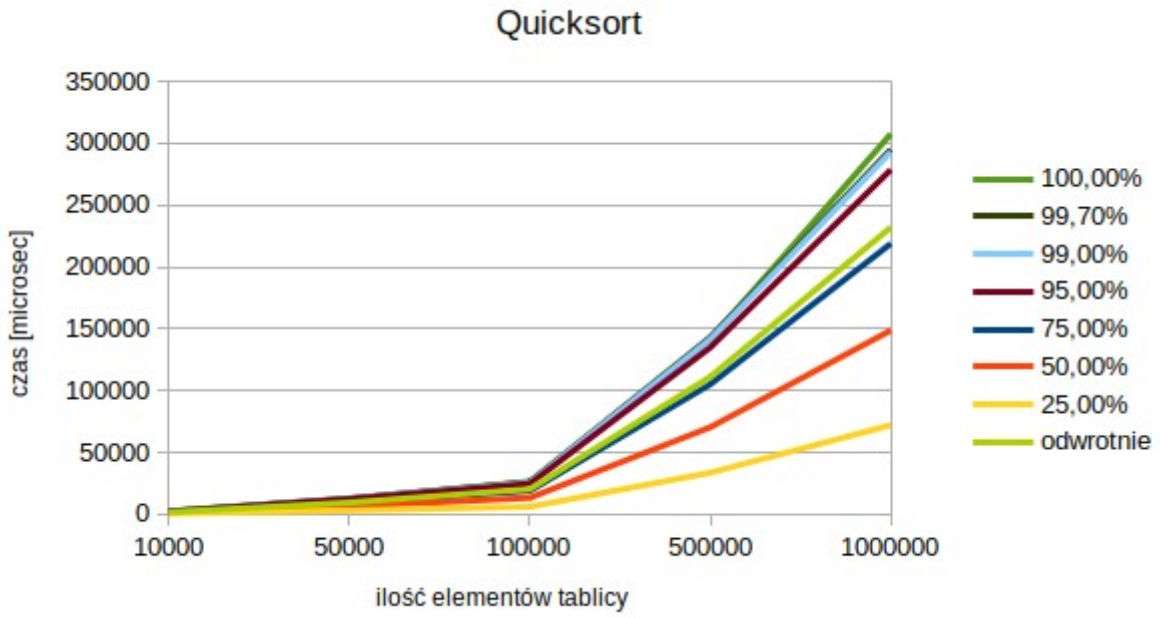
Sortowanie szybkie (Quicksort) vs Sortowanie introspektywne (Introspective sort):

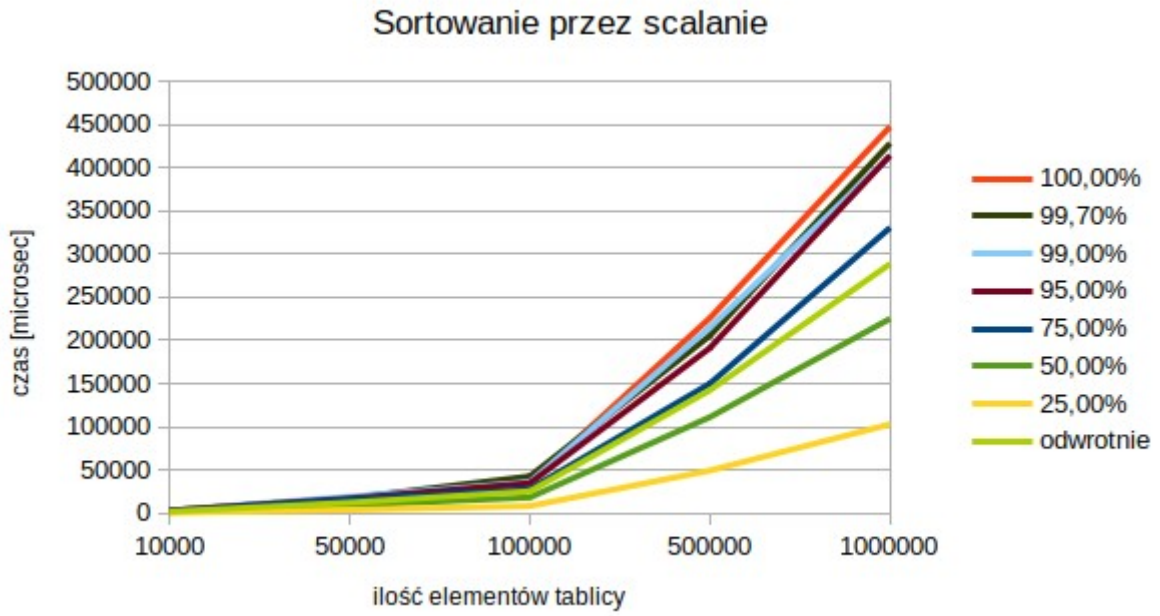
	Sortowanie szybkie (Quicksort)	**Sortowanie introspektywne (Introspective sort)**
Najlepszy przypadek złożoności obliczeniowej	*O(n log n)*	*O(n log n)*
Standardowy przypadek złożoności obliczeniowej	*O(n log n)*	*O(n log n)*
Najgorszy przypadek złożoności obliczeniowej	$O(n^2)$	*O(n log n)*
Złożoność pamięciowa:	*O(log n) | O(n)*	*O(log n)*
Stabilność	Niestabilny	Niestabilny
Implementacja	Łatwa	Trudna

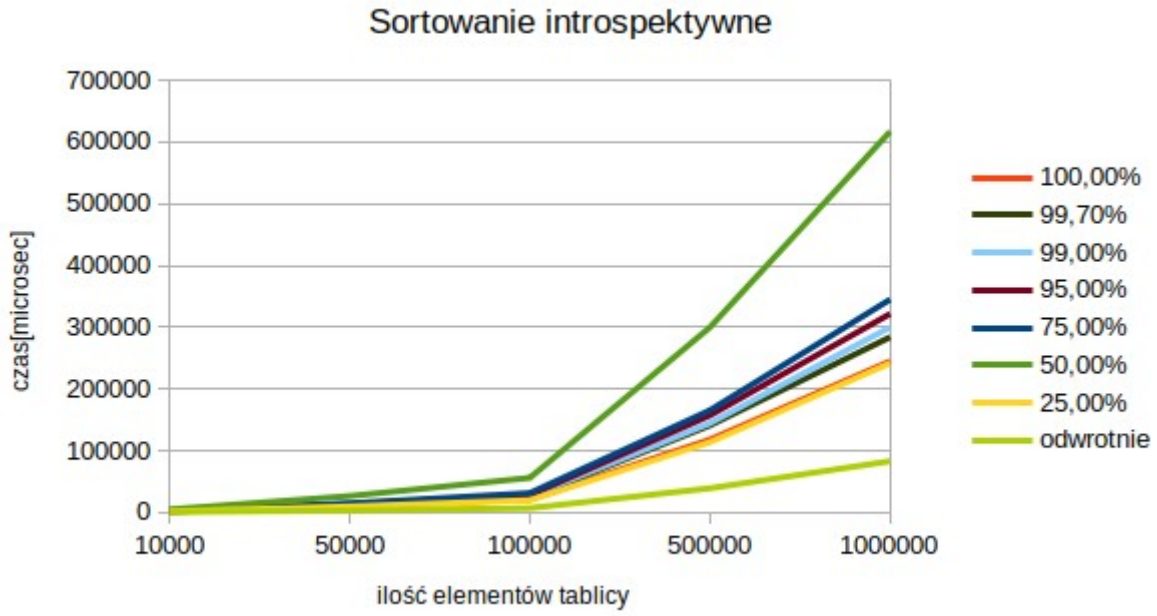
Sortowanie przez scalanie (Mergesort) vs Sortowanie introspektywne (Introspective sort):

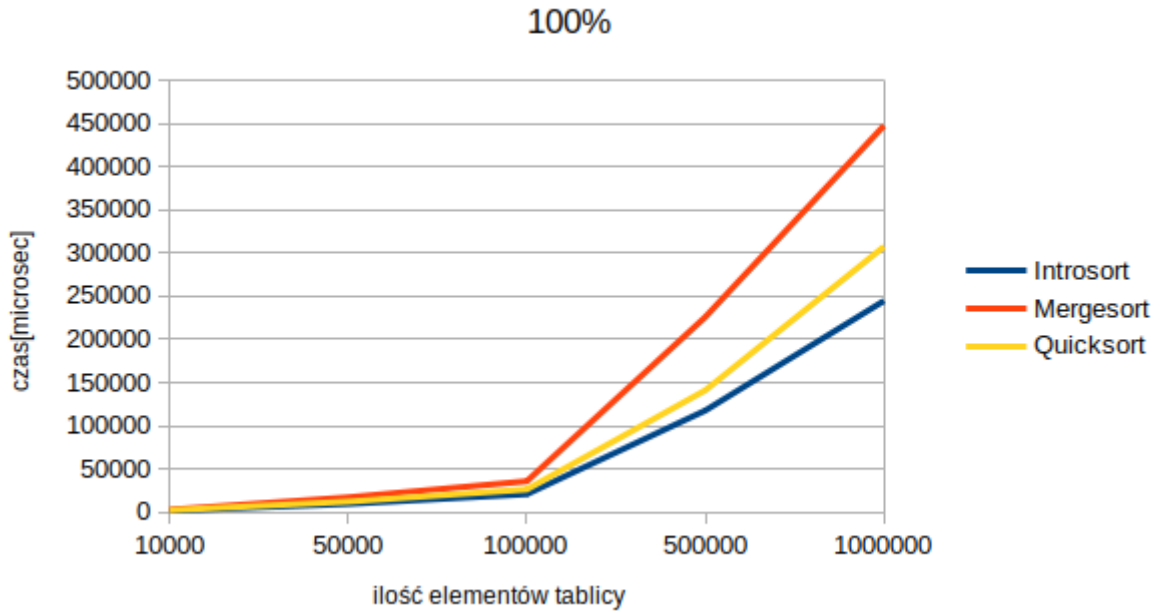
	Sortowanie przez scalanie (Mergesort)	**Sortowanie introspektywne (Introspective sort)**
Najlepszy przypadek złożoności obliczeniowej	*O(n log n)*	*O(n log n)*
Standardowy przypadek złożoności obliczeniowej	*O(n log n)*	*O(n log n)*
Najgorszy przypadek złożoności obliczeniowej	*O(n log n)*	*O(n log n)*
Złożoność pamięciowa:	O(n)	*O(n log n)*
Stabilność	Stabilny	Niestabilny
Implementacja	Łatwa	Łatwa

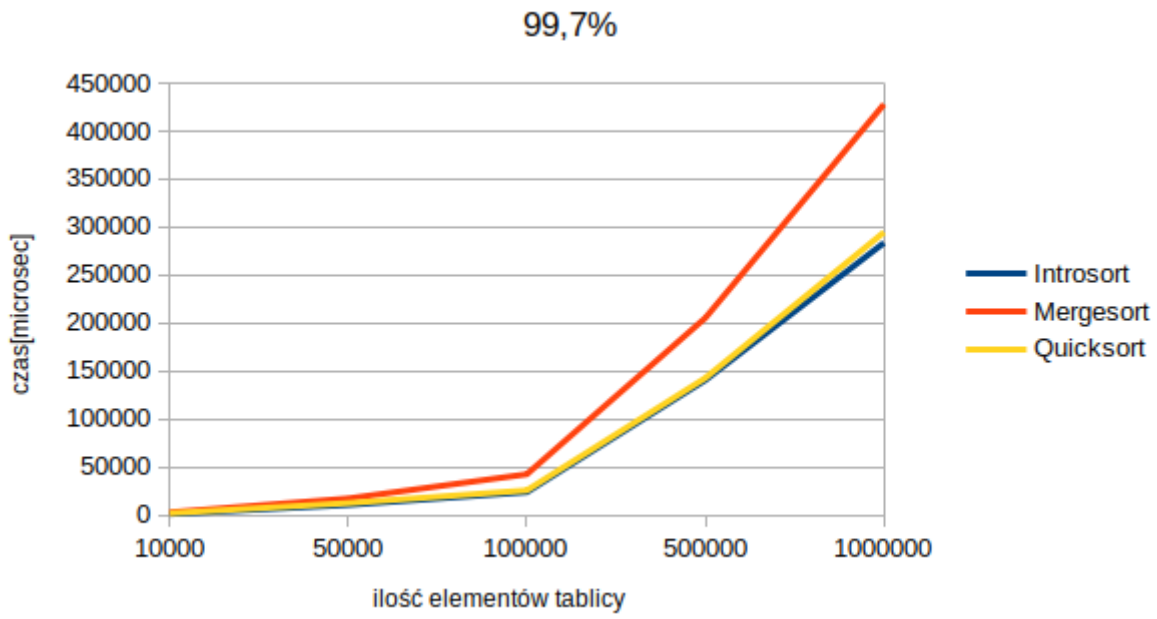
Wyniki testów:

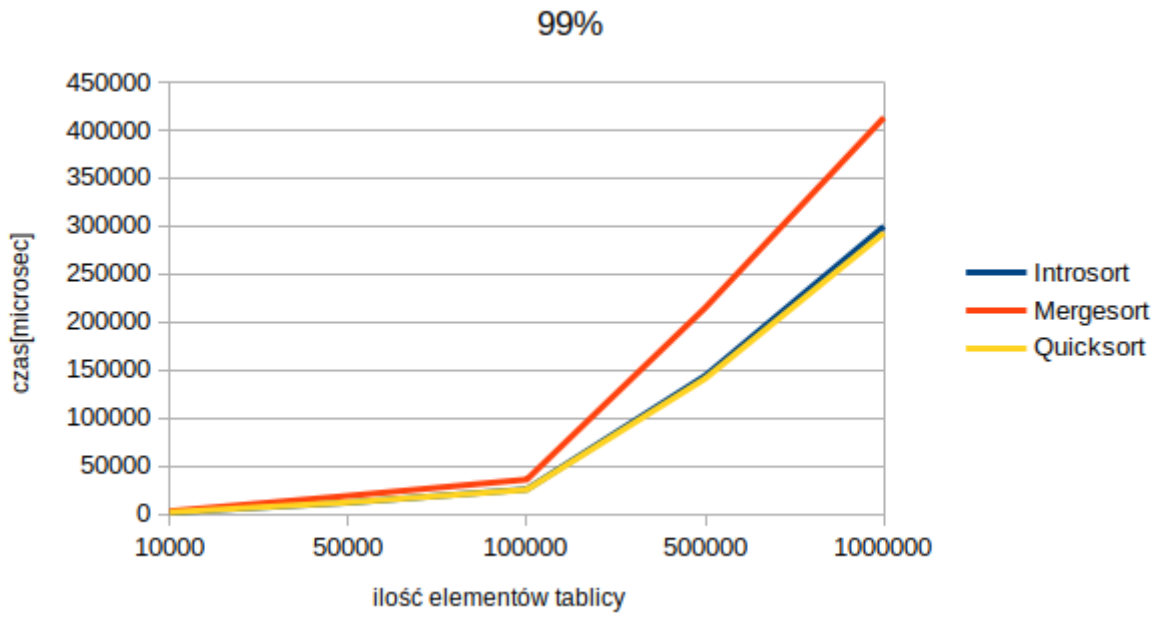


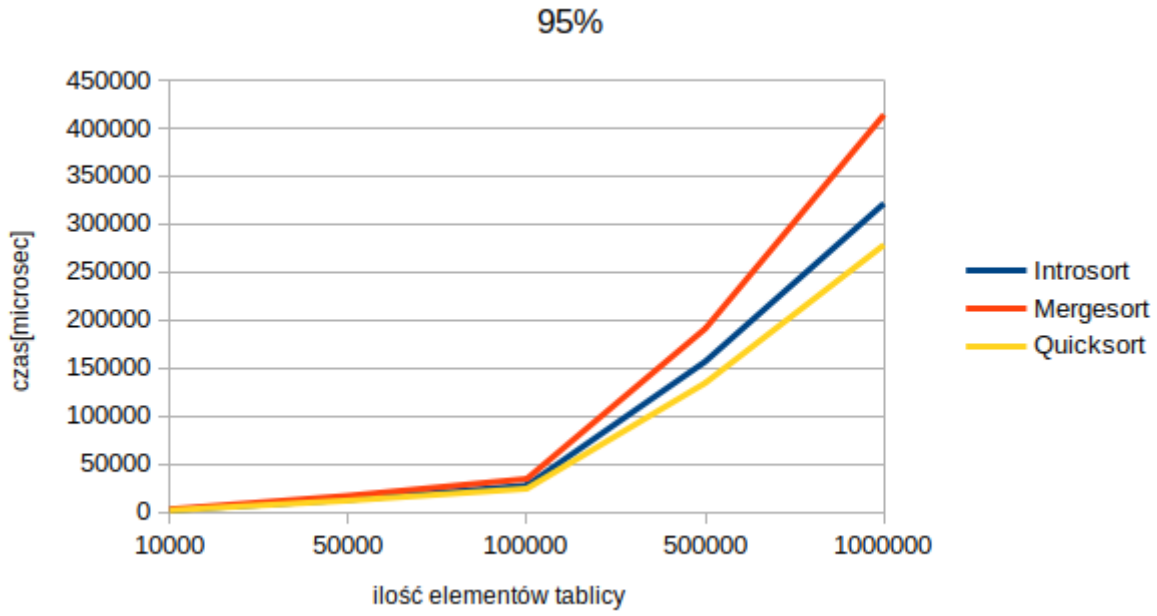


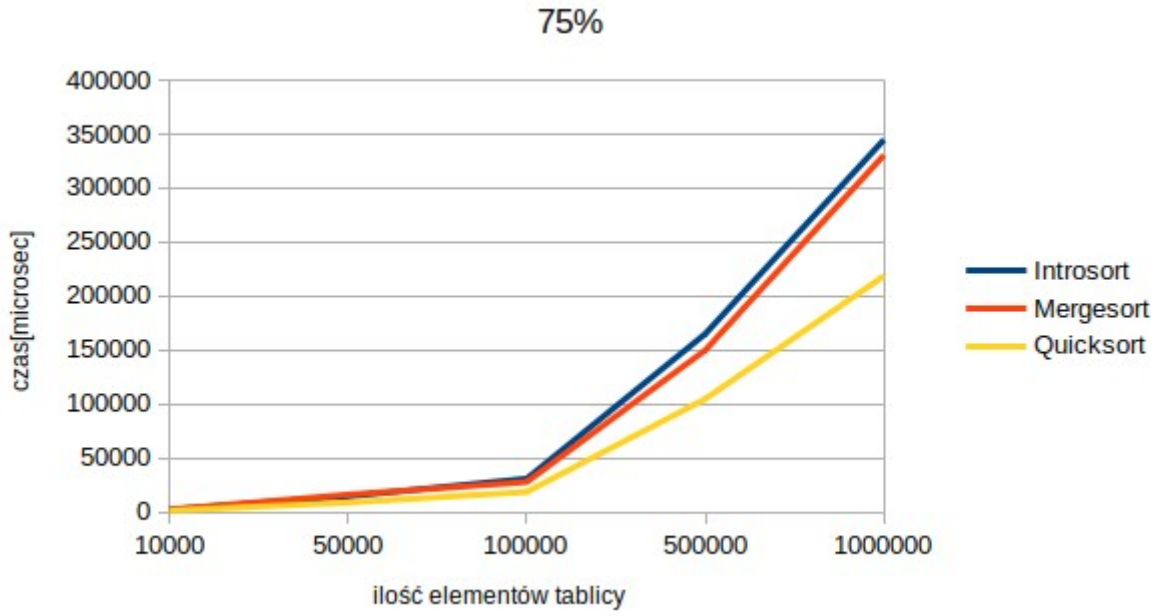

Sortowanie introspektywne
700000
600000
500000
400000
300000
200000
100000
0
czas[microsec]
10000
50000
100000
500000
1000000
ilość elementów tablicy
100,00%
99,70%
99,00%
95,00%
75,00%
50,00%
25,00%
odwrotnie

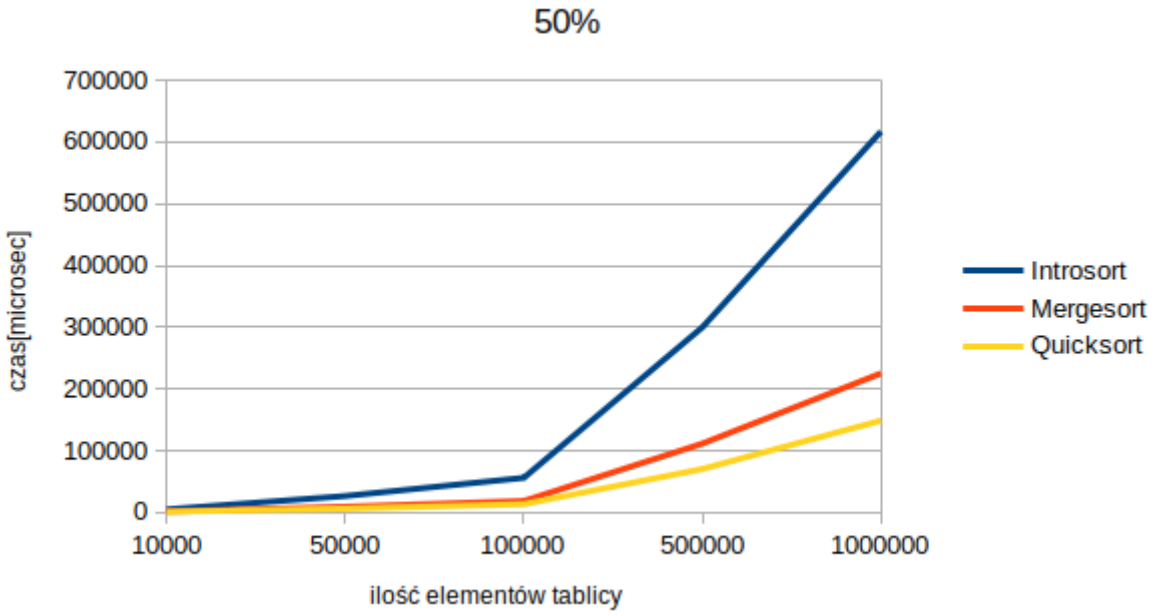

100%
500000
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
czas[microsec]
10000
50000
100000
500000
1000000
ilość elementów tablicy
Introsort
Mergesort
Quicksort

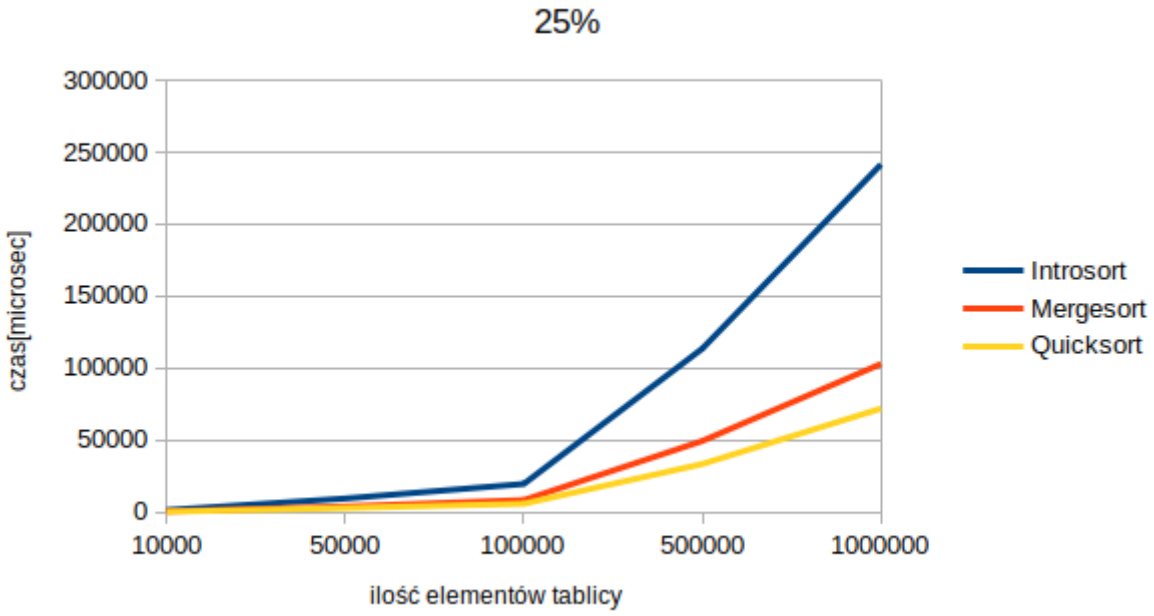

99,7%
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
czas[microsec]
10000
50000
100000
500000
1000000
ilość elementów tablicy
Introsort
Mergesort
Quicksort

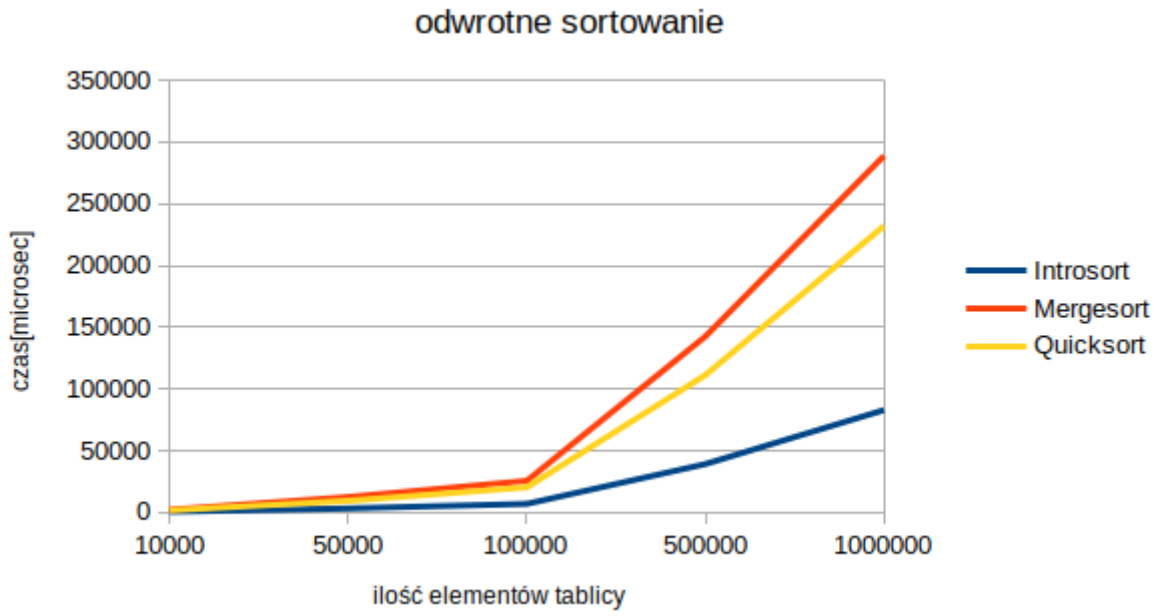

99%
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
czas[microsec]
10000
50000
100000
500000
1000000
ilość elementów tablicy
Introsort
Mergesort
Quicksort


95%
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
czas[microsec]
10000
50000
100000
500000
1000000
ilość elementów tablicy
Introsort
Mergesort
Quicksort


75%
400000
350000
300000
250000
200000
150000
100000
50000
0
czas[microsec]
10000
50000
100000
500000
1000000
ilość elementów tablicy
Introsort
Mergesort
Quicksort


50%
700000
600000
500000
400000
300000
200000
100000
0
czas[microsec]
10000
50000
100000
500000
1000000
ilość elementów tablicy
Introsort
Mergesort
Quicksort


25%
300000
250000
200000
150000
100000
50000
0
czas[microsec]
10000
50000
100000
500000
1000000
ilość elementów tablicy
Introsort
Mergesort
Quicksort



Wnioski:
Dzięki wykonanemu ćwiczeniu poznaliśmy szereg różnych algorytmów sortujących, takich jak efektywność, czy złożoność obliczeniową.
Z powyższych wykresów widzimy, że:

- Sortowanie przez scalanie jest najmniej efektywne
- W niektórych przypadkach sortowanie szybkie jest szybsze niż sortowanie introspektywne
- Sortowanie szybkie jest najlepsze dla tablic z mniejszym procentem posortowania
- Sortowanie introspektywne jest najlepsze dla tablic z większym procentem posortowania
- Wyniki testów w przybliżeniu określają nasze założenia
- Czas dla tablic do 50 000 jest zbliżony dla tych trzech sortowań

Uwagi:
Sortowanie introspektywne, które jest hybrydą 2 innych sortowań korzysta z różnych algorytmów. Dzieje się tak, ponieważ wybór metody jest zależny od rozmiaru stworzonych podtablic.
Jak wiemy Introsort powinien być w każdym przypadku szybszy od Quicksorta. Niestety implementacja sortowania introspektywnego jest trudna i skomplikowana, przez co w jej trakcie możemy popełnić wiele błędów, które mają znaczący wpływ na czas obliczeniowy.

Literatura:
Quicksort:

- https://en.wikipedia.org/wiki/Quicksort
- https://pl.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort
- http://algorytmy.ency.pl/artykul/quicksort
- https://www.geeksforgeeks.org/quick-sort/

Sortowanie przez scalanie:

- https://en.wikipedia.org/wiki/Merge_sort
- https://www.geeksforgeeks.org/merge-sort/
- http://www.algorytm.org/algorytmy-sortowania/sortowanie-przez-scalanie-mergesort.html

Sortowanie introspektywne:

- https://en.wikipedia.org/wiki/Introsort
- https://www.youtube.com/watch?v=67ta5WTjjUo
- https://aquarchitect.github.io/swift-algorithm-club/Introsort/