Projektowanie i Analiza Algorytmów	
Kierunek *Informatyczne Systemy Automatyki*	**Termin** *Wtorek* $15^{15} - 16^{55}$
Imię, nazwisko, numer albumu *Mikołaj Nowak 280082*	**Data** *08.04.2025*
Link do projektu `https://github.com/devoid-of-though/Projektowanie_algorytmow`	

Sprawozdanie Projekt nr 1

1 Wstęp

Celem niniejszego sprawozdania jest analiza trzech algorytmów sortowania: przez kopcowanie, timsorta oraz sortowania introspektywnego. Algorytmy zostały zaimplementowane w języku C++, aby można było zbadać ich efektywność oraz złożoność obliczeniową. Algorytmy zostały przetestowane dla rozmiarów tablic: 1000, 10000, 50000, 100000, 500000, 1000000 oraz stopni posortowań tablic: 0%, 25%, 50%, 75%, 95%, 97%, 99%, 99.7%, 100% oraz 100% w odwrotnej kolejności w ilości 100 próbek na typ tablicy.

2 Opis algorytmów

2.1 Sortowanie przez kopcowanie

Algorytm sortowania przez kopcowanie wykorzystuje strukturę kopca do sortowania. Wiedząc, że największy element kopca jest jego pierwszym elementem, sortowanie przez kopcowanie buduje kopiec po czym wyjmowany jest pierwszy element na koniec tablicy, czynność ta powtarzana jest aż do osiągnięcia posortowanej tablicy. Teoretyczna złożoność obliczeniowa dla tego algorytmu to O(nlogn) dla wszystkich przypadków.

2.2 Timsort

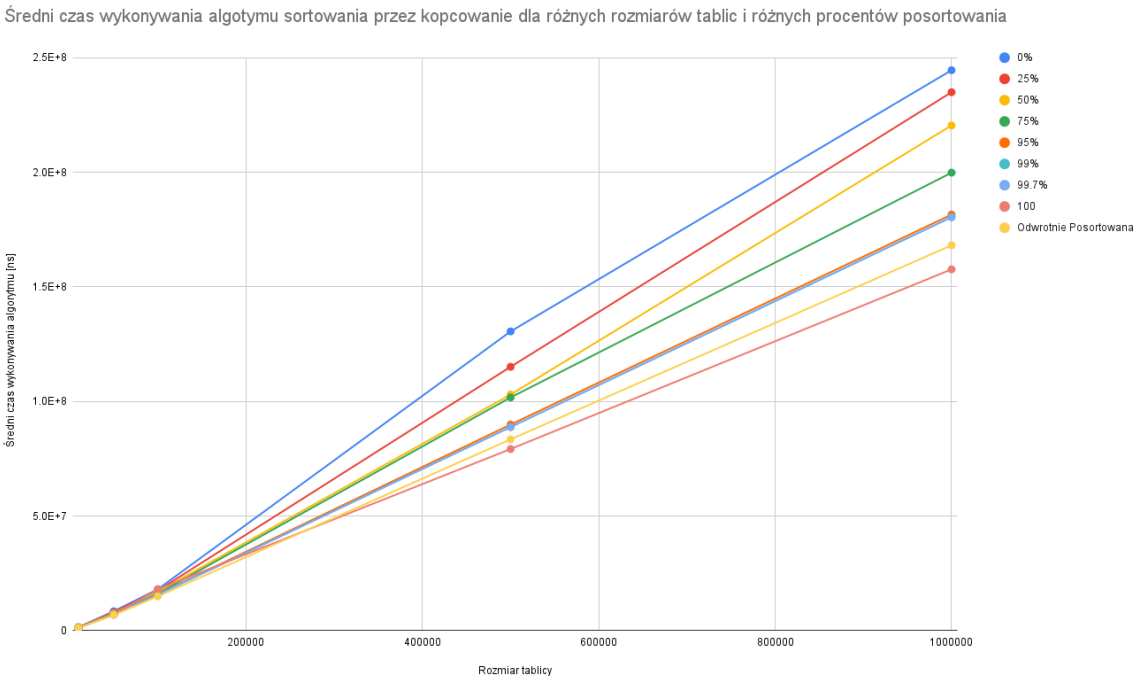
Algorytm Timsort rozpoczyna się od podzielenia tablicy na mniejsze subtablice, o danym rozmiarze i posortowanie ich za pomocą algorytmu sortowania przez wstawianie, następnie za pomocą algorytmu sortowania przez scalanie łączymy i sortujemy subtablice ze sobą, aż do uzyskania posortowanej tablicy. Teoretyczna złożoność obliczeniowa dla tego algorytmu to O(nlogn) zarówno dla najgorszego jak i średniego przypadku ze względu na oparcie go na sortowaniu przez scalanie.

2.3 Sortowanie introspektywne

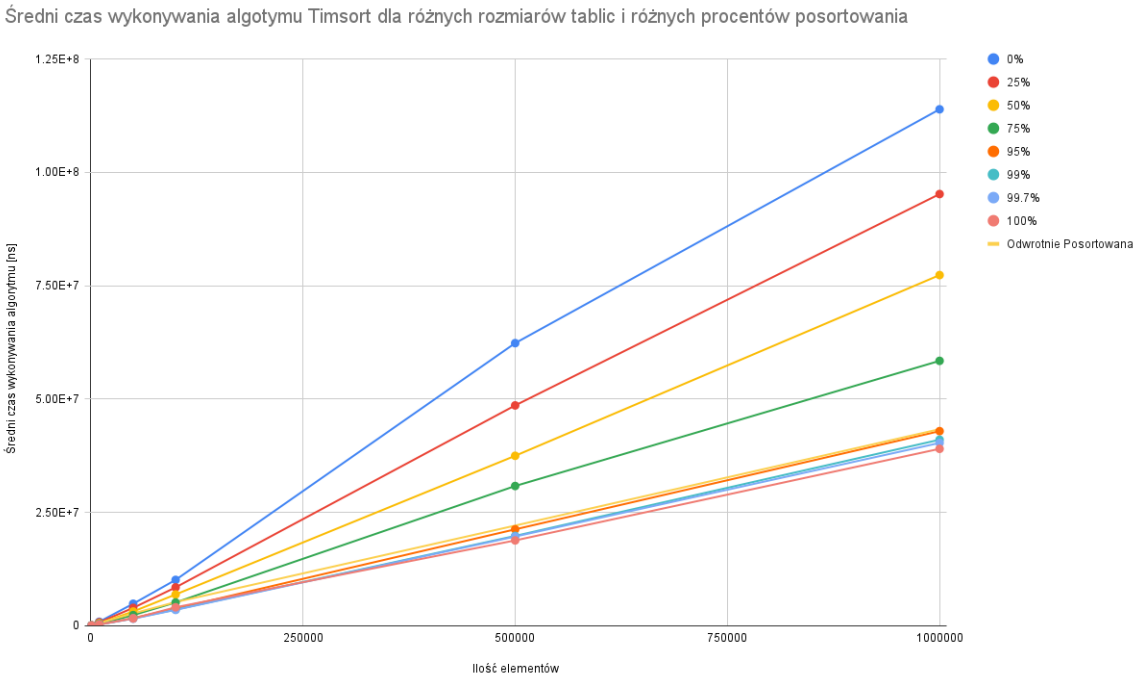
Algorytm sortowania introspektywnego wykorzystuje rekurencyjne odwołanie by podzielić tablice na mniejsze częśći, a następnie dla dostatecznie małych subtablic $n < 16$ wykorzystuje sortowanie przez wstawianie, jeśli z koleji przekroczymy pewną głębokość rekurencji wykorzystuje heapsort. Teoretyczna złożoność obliczeniowa dla tego algorytmu to O(nlogn) zarówno dla najgorszego jak i średniego przypadku ze względu na oparciu go na sortowaniu przez kopcowanie przy przekroczeniu głębokości rekurencji.

3 Analiza i weryfikacja algorytmów

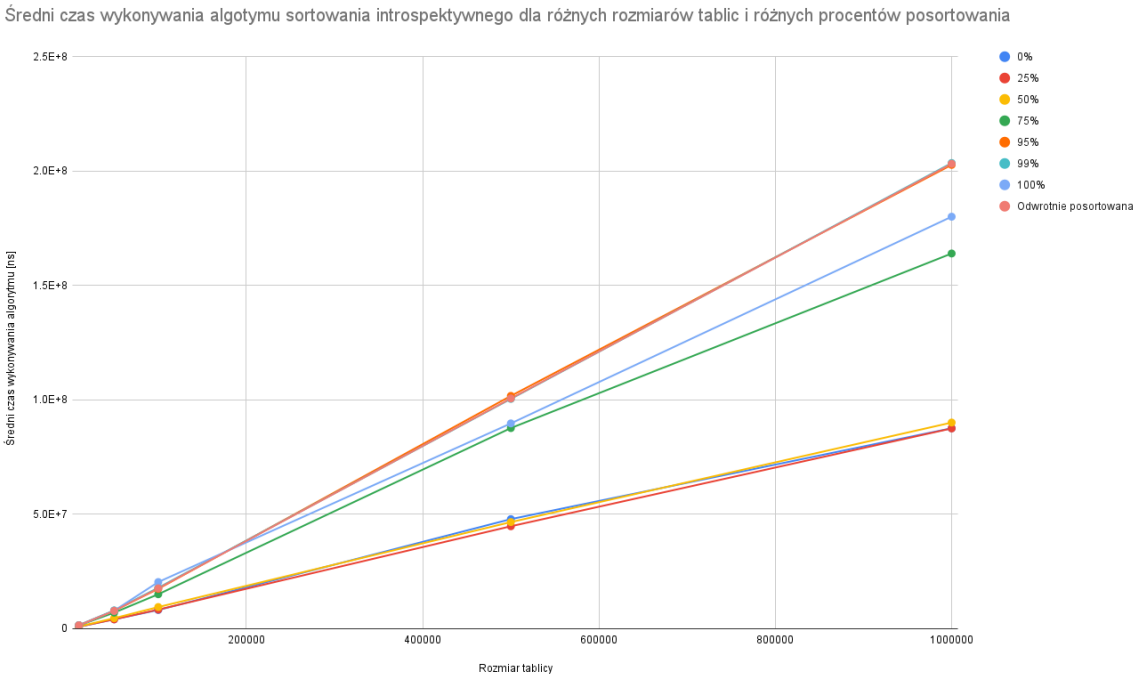
3.1 Porównanie średnich czasów wykonywania algorytmów



Rysunek 1: Wykres średniego czasu działania algorytmu sortowania przez kopcowanie dla poszczególnych tablic



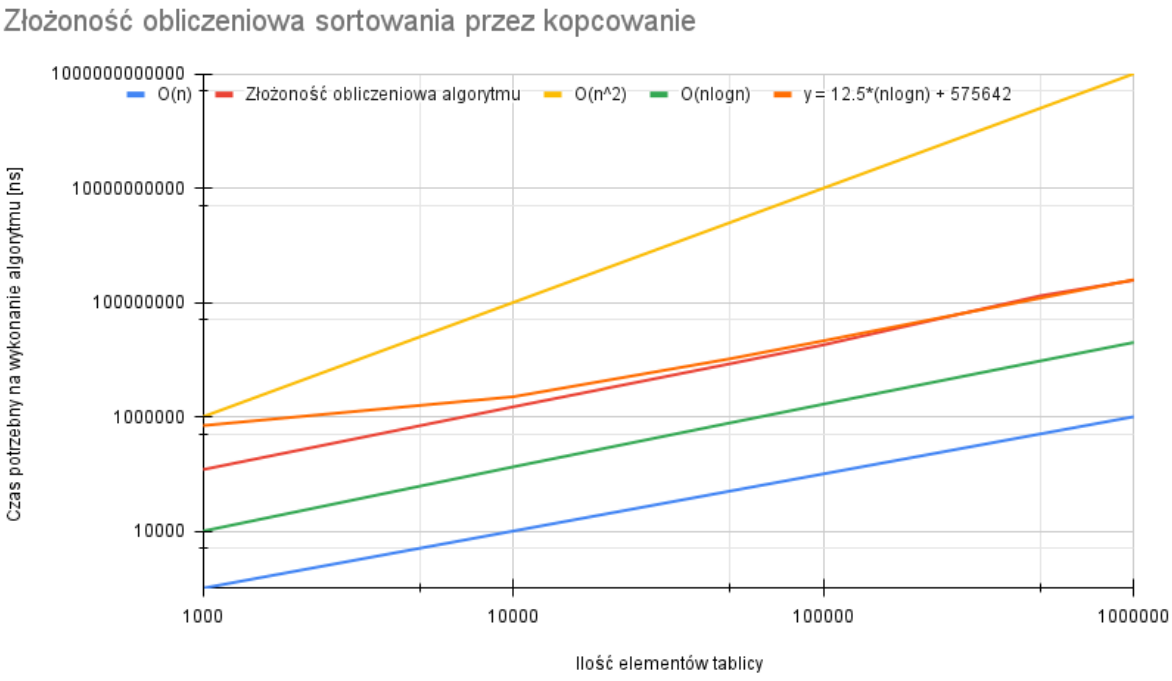
Rysunek 2: Wykres średniego czasu działania algorytmu Timsort dla poszczególnych tablic



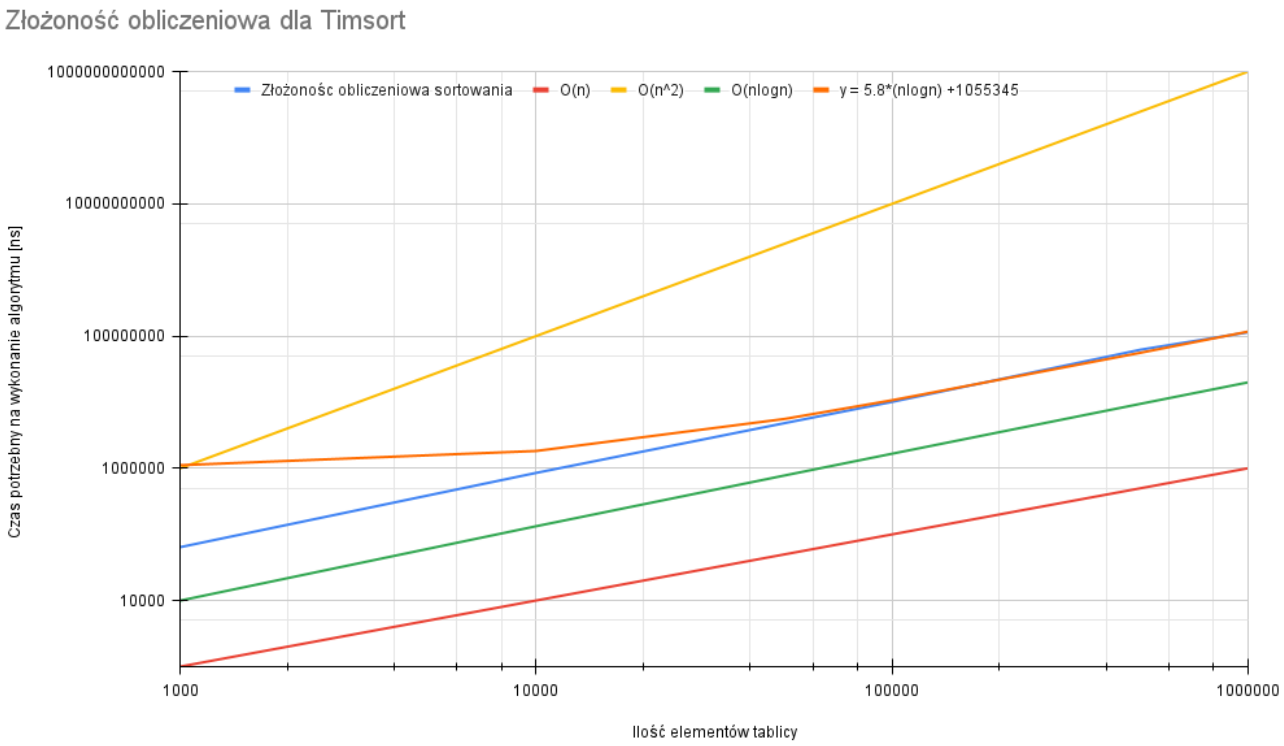
Rysunek 3: Wykres średniego czasu działania algorytmu sortowania introspektywnego dla poszczególnych tablic

3.2 Złożoność obliczeniowa dla najgorszego przypadku

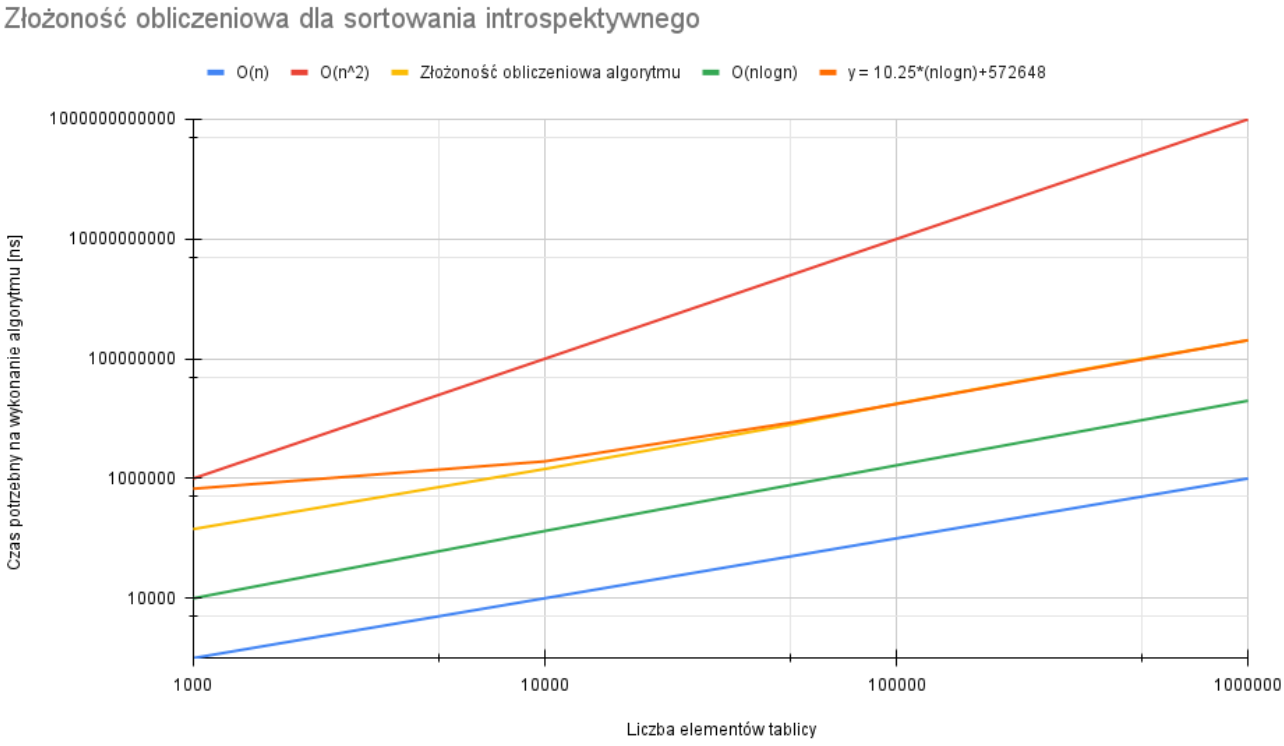
Złożoność algorytmów z przybliżoną złożonością y(n) = a*(n*logn) + b



Rysunek 4: Złożoność obliczeniowa dla sortowania przez kopcowanie dla najgorszego przypadku: tablica nieposortowana

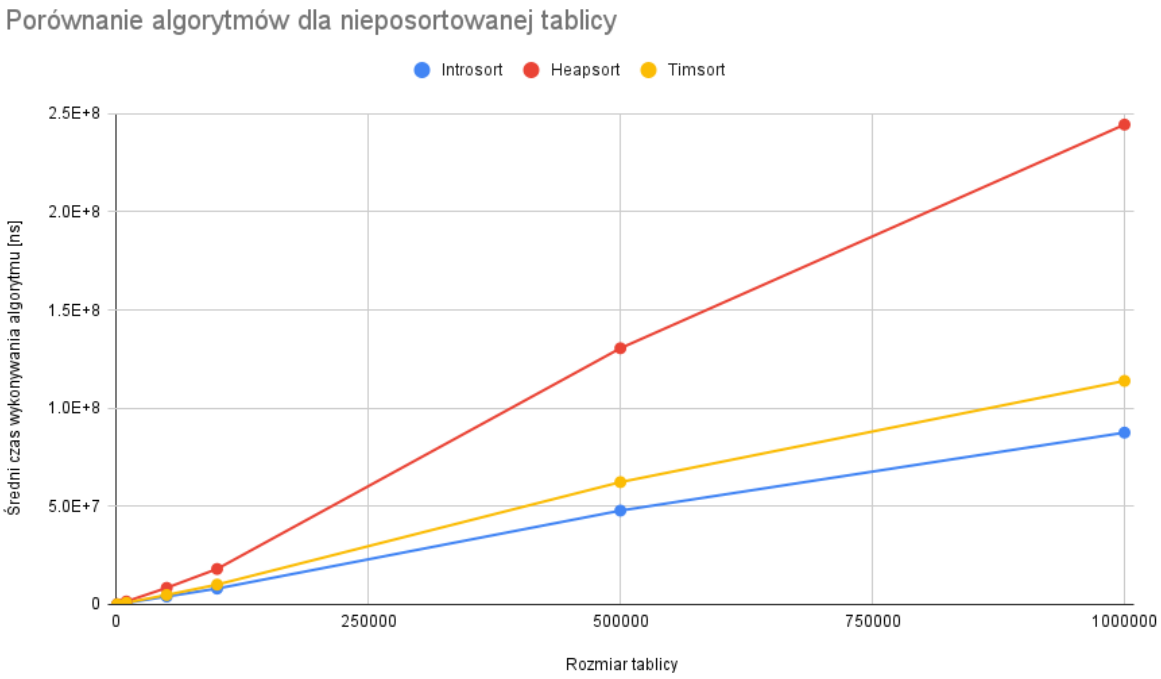


Rysunek 5: Złożoność obliczeniowa dla timsort'a dla najgorszego przypadku: tablica nieposortowana

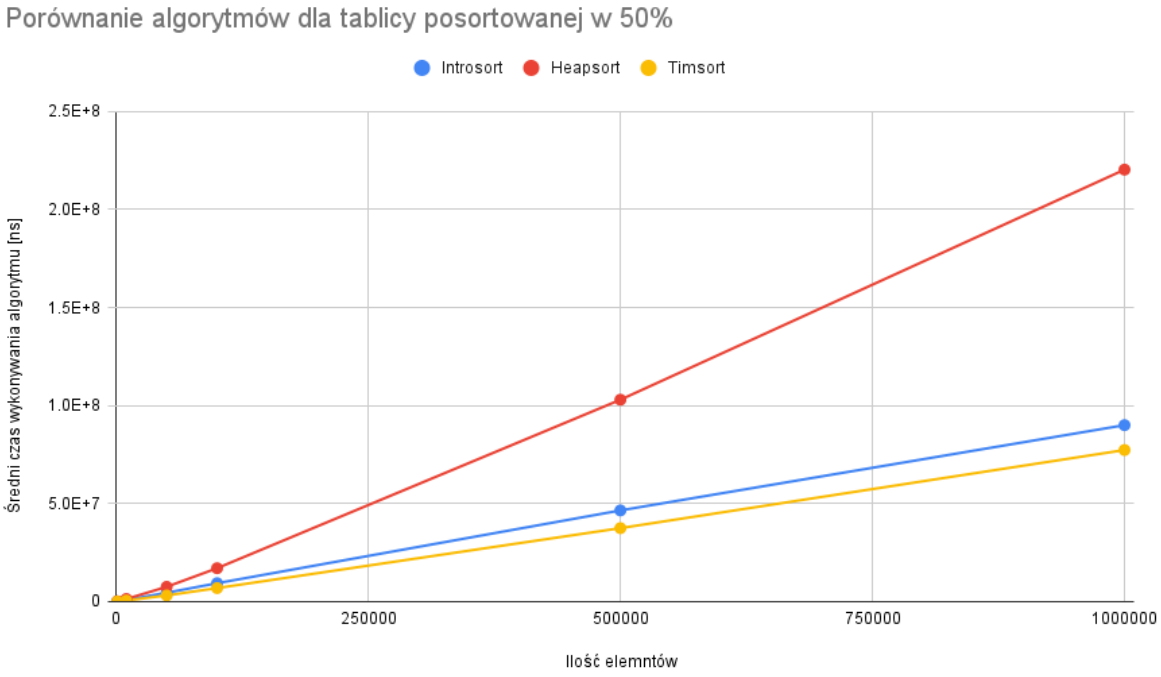


Rysunek 6: Złożoność obliczeniowa dla sortowania introspektywnego dla najgorszego przypadku: tablica posortowana w 99%

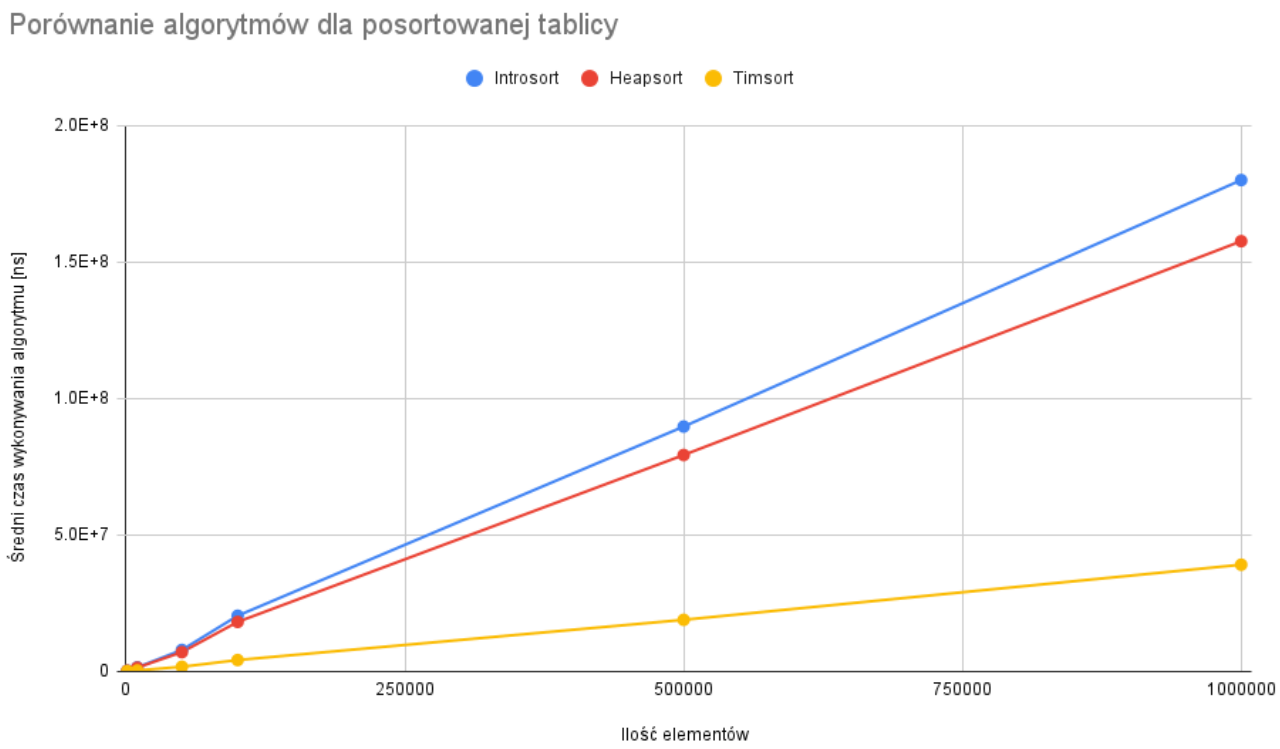
3.3 Porównanie średniego czasu wykonania pomiędzy algorytmami



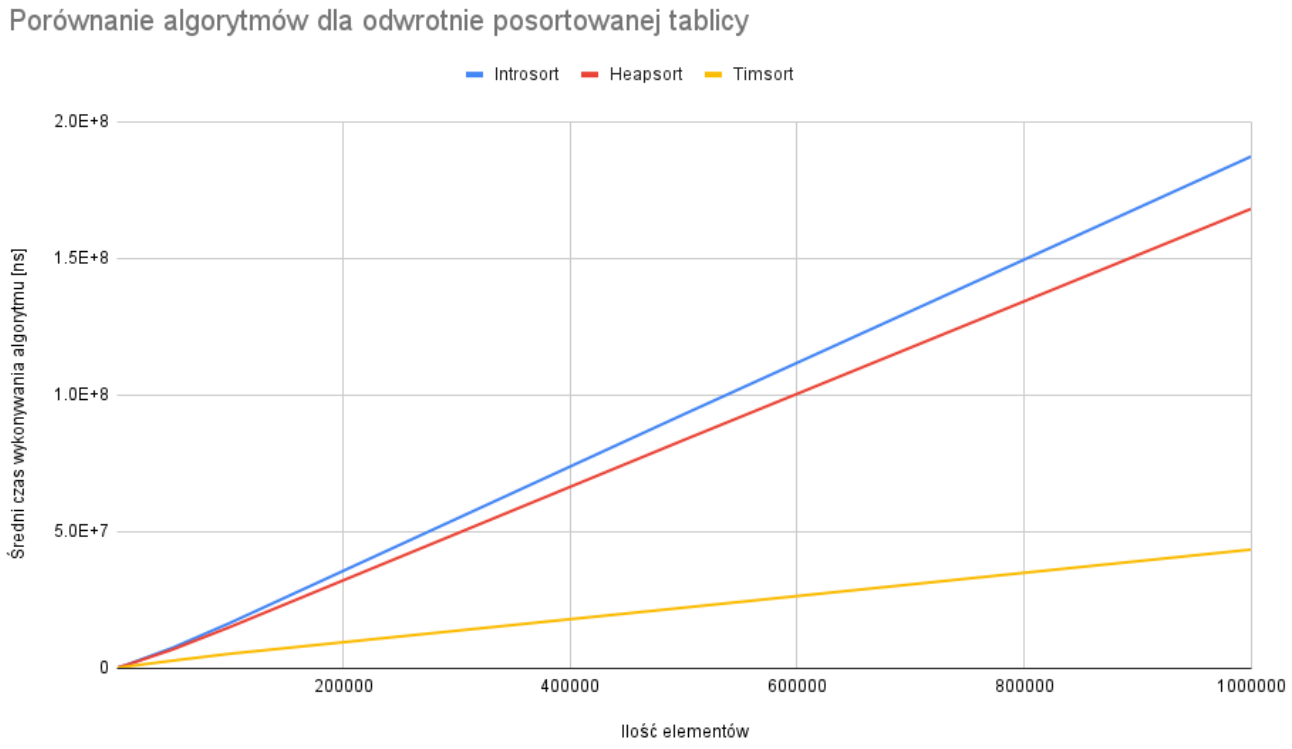
Rysunek 7: Wykres średniego czasu działania algorytmów dla nieposortowanej tablicy



Rysunek 8: Wykres średniego czasu działania algorytmów dla tablicy posortowanej w 50%



Rysunek 9: Wykres średniego czasu działania algorytmów dla posortowanej tablicy



Rysunek 10: Wykres średniego czasu działania algorytmów dla odwrotnie posortowanej tablicy

4 Wnioski

- Teoretyczne złożoności obliczeniowe zgadzają się w większości z praktyką i dla najgorszego przypadku wszystkie wyniosły O(nlogn).
- Warto zaznaczyć, że teoretyczne najgorsze przypadki różniły się od rzeczywistych, co mogło być spowodowane między innymi procesami w tle, środowiskiem lub urządzeniem na którym działał program .
- Timsort jest algorytmen o największej wydajnośći, przy czym dla nieposortowanej tablicy wypada gorzej od sortowania introspektywnego, które nie radzi sobie z tablicami o wysokim stopniu posortowania. Sortowanie przez kopcowanie jest stabilnym ale wolniejszym od sortowań hybrydowych algorytmem.
- Próba zaimplementowania dual pivot quicksort sprawiła problem poprzez zbyt dużą głębie rekurencji, co skutkowało błędęm segmentacji dla tablicy 500000 posortowanej w 100% .