

18.03.2020r.
Emilia Starczyk
249005

PROJEKTOWANIE ALGORYTMÓW I METODY SZTUCZNEJ INTELIGENCJI

Projekt 1

Dr inż. Łukasz Jeleń
Czwartek 9.15 - 11.00

1. Wstęp

Tematem projektu było zaimplementowanie trzech wybranych algorytmów sortowania oraz zbadanie ich złożoności obliczeniowej. Wybrane przeze mnie sortowania: sortowanie przez scalanie, sortowanie szybkie oraz sortowanie introspektywne. Algorytmy tych sortowań można implementować w różny sposób. Są to algorytmy, które mają złożoność obliczeniową $O(n \cdot \log n)$. Jedynym wyjątkiem jest sortowanie szybkie, które posiada pesymistyczny przypadek o złożoności $O(n^2)$.

2. Opis algorytmów sortowania

2.1. Sortowanie przez scalanie

Sortowanie przez scalanie jest sortowaniem rekurencyjnym, wykorzystuje metodę „dziel i zwyciężaj”. W myśl tej zasady, problemy dzieli się na mniejsze podproblemy, aż do momentu, kiedy są one proste do rozwiązania. Po rozwiązaniu danych zagadnień scala się je, rozwiązując, coraz to „większe” problemy.

Sortowanie przez scalanie odbywa się w kilku podstawowych krokach:

1. Podział tablicy na dwie części.
2. Zastosowanie do obu części sortowania przez scalanie (chyba, że został już tylko jeden element).
3. Scalenie elementów, w posortowany ciąg.

Sortowanie przez scalanie jest sortowaniem stabilnym, o złożoności obliczeniowej $O(n \cdot \log n)$, niezależnie od przypadku. Wadą tego sortowania jest potrzeba zastosowania pomocniczej struktury, co w przypadku sortowania małych ilości danych nie ma większego znaczenia. Efektywność algorytmu maleje wraz ze wzrostem liczby danych. Duże znaczenie będzie miał tutaj również typ danych.

2.2. Sortowanie szybkie

Sortowanie szybkie, jak sortowanie przez scalanie, również jest sortowaniem rekurencyjnym oraz wykorzystuje metodę „dziel i zwyciężaj”. Sortowanie, tak naprawdę, sprowadza się do dwóch części: części służącej do właściwego sortowania (czyli tak naprawdę wykonywania funkcji przez samą siebie) oraz części do rozdzielania elementów względem pewnej wartości, tzn. elementu osiowego. Istnieją różne metody obierania elementu osiowego. W niektórych implementacjach będzie to pierwszy element danego fragmentu, mediana elementów: pierwszego, ostatniego i środkowego lub też po prostu element środkowy danego fragmentu, co osobiście wykorzystuje w mojej implementacji.

Sortowanie szybkie opiera się na wybraniu elementu osiowego, „podzieleniu” tablicy na dwie części, z których, pierwsza zawiera elementy mniejsze od elementu osiowego, a druga większe. Następnie funkcja sortowania wywoływana jest zarówno dla pierwszej i drugiej części. W ten sposób uzyskuje się tablicę posortowaną.

Sortowanie szybkie jest sortowaniem niestabilnym o złożoności $O(n \cdot \log n)$ w przypadku przeciętnym. Wadą tego typu algorytmu jest przypadek pesymistyczny, kiedy jako element osiowy obierzemy najmniejszy (lub największy) element danego fragmentu tablicy. W takiej sytuacji złożoność obliczeniowa jest kwadratowa, ponieważ taki wybór prowadzi do wielu porównań, zamian i rekurencji.

2.3. Sortowanie introspektywne

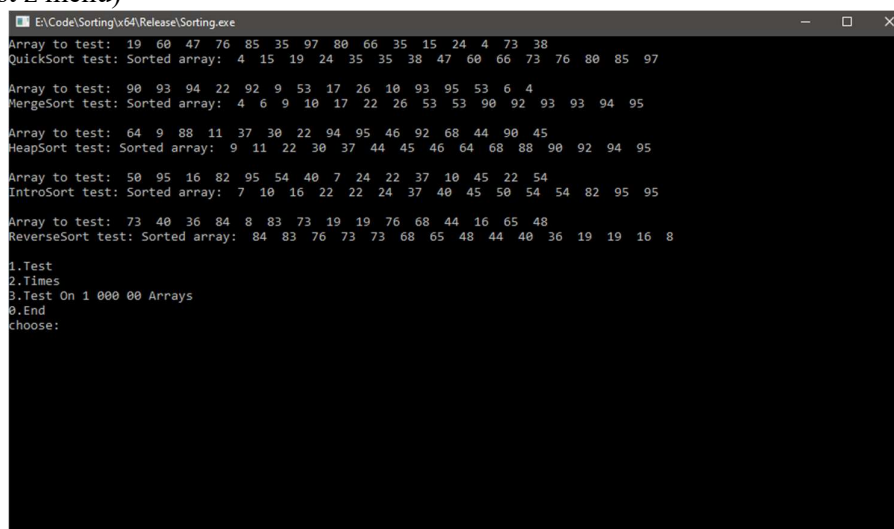
Sortowanie introspektywne jest sortowaniem hybrydowym, co oznacza, że wykorzystuje więcej niż jeden algorytm sortowania. Z reguły, sortowanie to, opiera się na sortowaniu szybkim oraz przez kopcowanie. Istnieją również implementacje wykorzystujące dodatkowo sortowanie przez wstawianie. Sortowanie introspektywne pozwala na obsługę najgorszego przypadku algorytmu sortowania szybkiego, poprzez badanie głębokości rekurencji. Sortowanie introspektywne można opisać w kilku krokach.

Pierwszym krokiem jest obliczenie maksymalnej głębokości rekurencji. Głębokość rekurencji można dobrać dowolnie, jednak standardowym wyborem jest wyliczenie głębokości na podstawie wzoru $2 \cdot \log_2 n$ (gdzie n oznacza ilość elementów). W następnym kroku uruchamia się właściwe sortowanie. W tym algorytmie początkowo sortowanie wykonuje się tak, jak w sortowaniu szybkim, stopniowo zmniejszając stopień głębokości. Gdy jest on równy zero i rozmiar tablicy jest większy od 1, wykonywane jest sortowanie przez kopcowanie. Takie zastosowanie obu sortowań pozwala na uniknięcie pesymistycznego przypadku sortowania szybkiego. Sortowanie szybkie uruchamiane jest dla większych partii tablicy, zostawiając mniejsze fragmenty sortowaniu przez kopcowanie.

Sortowanie introspektywne jest sortowaniem niestabilnym o złożoności $O(n \cdot \log n)$, w przypadku pesymistycznym, co jest zarówno złożonością dla przypadku średniego.

3. Przebieg badań

Przygotowanie programu do testów zaczęłam od napisania szablonu klasy zawierającej tablice potrzebne do przeprowadzenia badań. Zaimplementowałam również podstawowe metody pozwalające na dostęp do tablic. Do generowania tablic losowych wykorzystałam funkcję `rand()`. Następnie zaimplementowałam sortowania oraz funkcje, pozwalające na testowanie danych sortowań. Dodałam funkcję sprawdzającą czy dana tablica jest posortowana. Dodałam również sortowanie malejąco i sortowanie fragmentu tablicy tak, aby można było przeprowadzić wszystkie testy. Na początku przeprowadziłam małe testy na tablicach 15-elementowych: (punkt 1. Test z menu)



```
E:\Code\Sorting\vs64\Release\Sorting.exe
Array to test: 19 60 47 76 85 35 97 80 66 35 15 24 4 73 38
QuickSort test: Sorted array: 4 15 19 24 35 35 38 47 60 66 73 76 80 85 97

Array to test: 90 93 94 22 92 9 53 17 26 10 93 95 53 6 4
MergeSort test: Sorted array: 4 6 9 10 17 22 26 53 53 90 92 93 93 94 95

Array to test: 64 9 88 11 37 30 22 94 95 46 92 68 44 90 45
HeapSort test: Sorted array: 9 11 22 30 37 44 45 46 64 68 88 90 92 94 95

Array to test: 50 95 16 82 95 54 40 7 24 22 37 10 45 22 54
IntroSort test: Sorted array: 7 10 16 22 22 24 37 40 45 50 54 54 82 95 95

Array to test: 73 40 36 84 8 83 73 19 19 76 68 44 16 65 48
ReverseSort test: Sorted array: 84 83 76 73 73 68 65 48 44 40 36 19 19 16 8

1. Test
2. Times
3. Test On 1 000 00 Arrays
0. End
choose:
```

Funkcja *issorted(...)* wyświetla informację tylko, kiedy jest błąd w sortowaniu tablicy, zatem testy na 15-elementowych tablicach były zakończone powodzeniem.

Następnie uruchomiłam funkcję Times z menu, dla zebrania informacji o czasach sortowań 100 tablic o różnych rozmiarach. Dane te zapisane zostały w pliku *Dane.txt*, następnie odpowiednio obrobione i przedstawione w kolejnym punkcie.

Fragment pliku Dane.txt:

Percent of sorted array;Name;Size;Time[micros]

0;QuickSort;10000;62266

0.25;QuickSort;10000;56294

0.5;QuickSort;10000;57721

0.75;QuickSort;10000;44691

0.95;QuickSort;10000;30900

0.99;QuickSort;10000;24378

0.997;QuickSort;10000;24142

Reverse;QuickSort;10000;12227

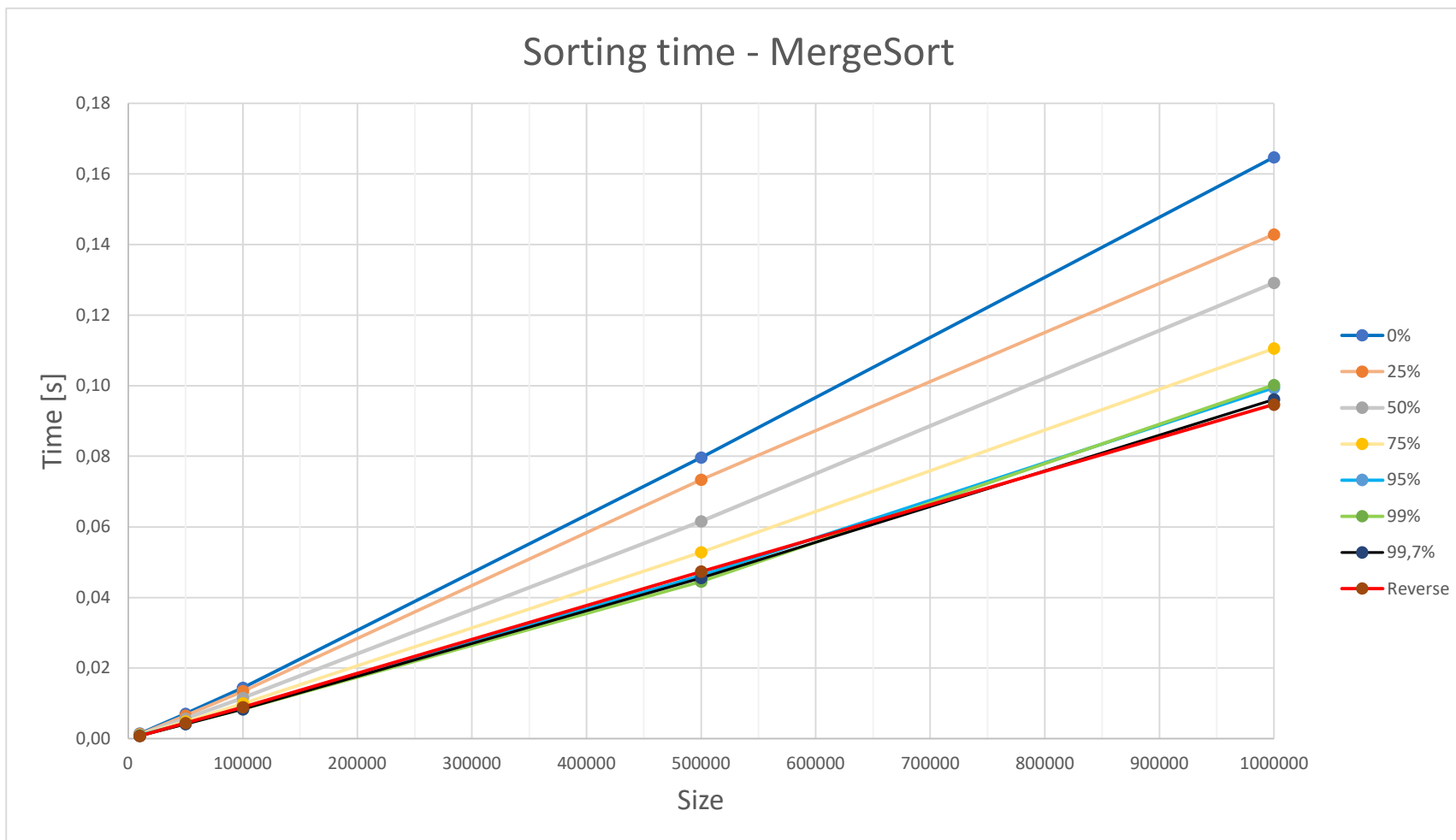
0;MergeSort;10000;140188

0.25;MergeSort;10000;129190

4. Wyniki

Tabela 1 Czasy sortowania tablic przy użyciu sortowania przez scalanie

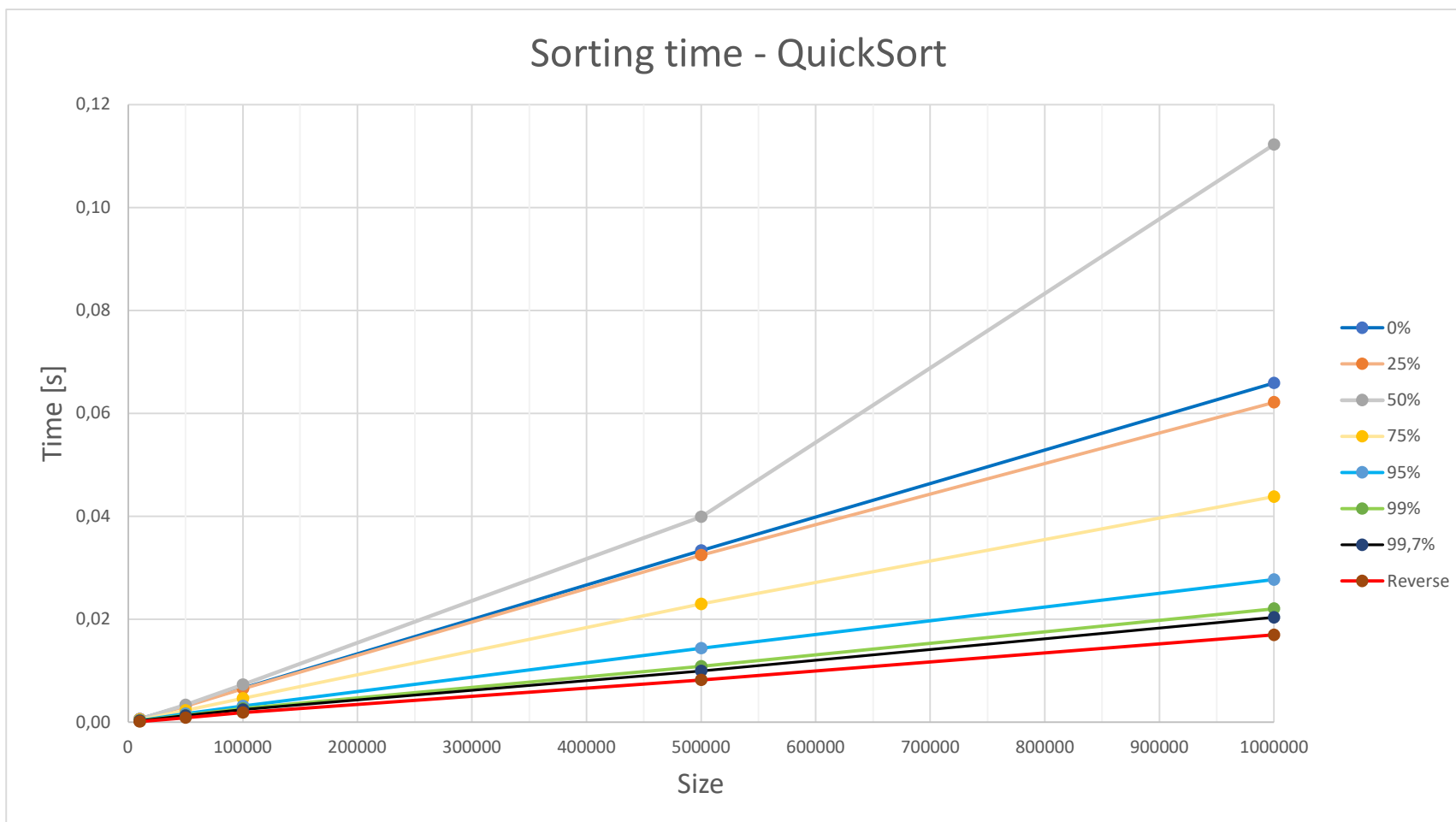
Percent of sorted array	Name	Size	Time[s]	Size	Time[s]	Size	Time[s]	Size	Time[s]	Size	Time[s]
0%	MergeSort	10000	0,00140	50000	0,0070	100000	0,0144	500000	0,080	1000000	0,165
25%			0,00129		0,0065		0,0135		0,073		0,143
50%			0,00109		0,0056		0,0116		0,062		0,129
75%			0,00115		0,0049		0,0099		0,053		0,111
95%			0,00101		0,0043		0,0087		0,046		0,099
99%			0,00085		0,0042		0,0085		0,045		0,100
99,7%			0,00081		0,0041		0,0084		0,046		0,096
100% Reverse			0,00081		0,0044		0,0090		0,047		0,095



Wykres 1 Czasy sortowania tablic przy użyciu sortowania przez scalanie

Tabela 2 Czasy sortowania tablic przy użyciu sortowania szybkiego

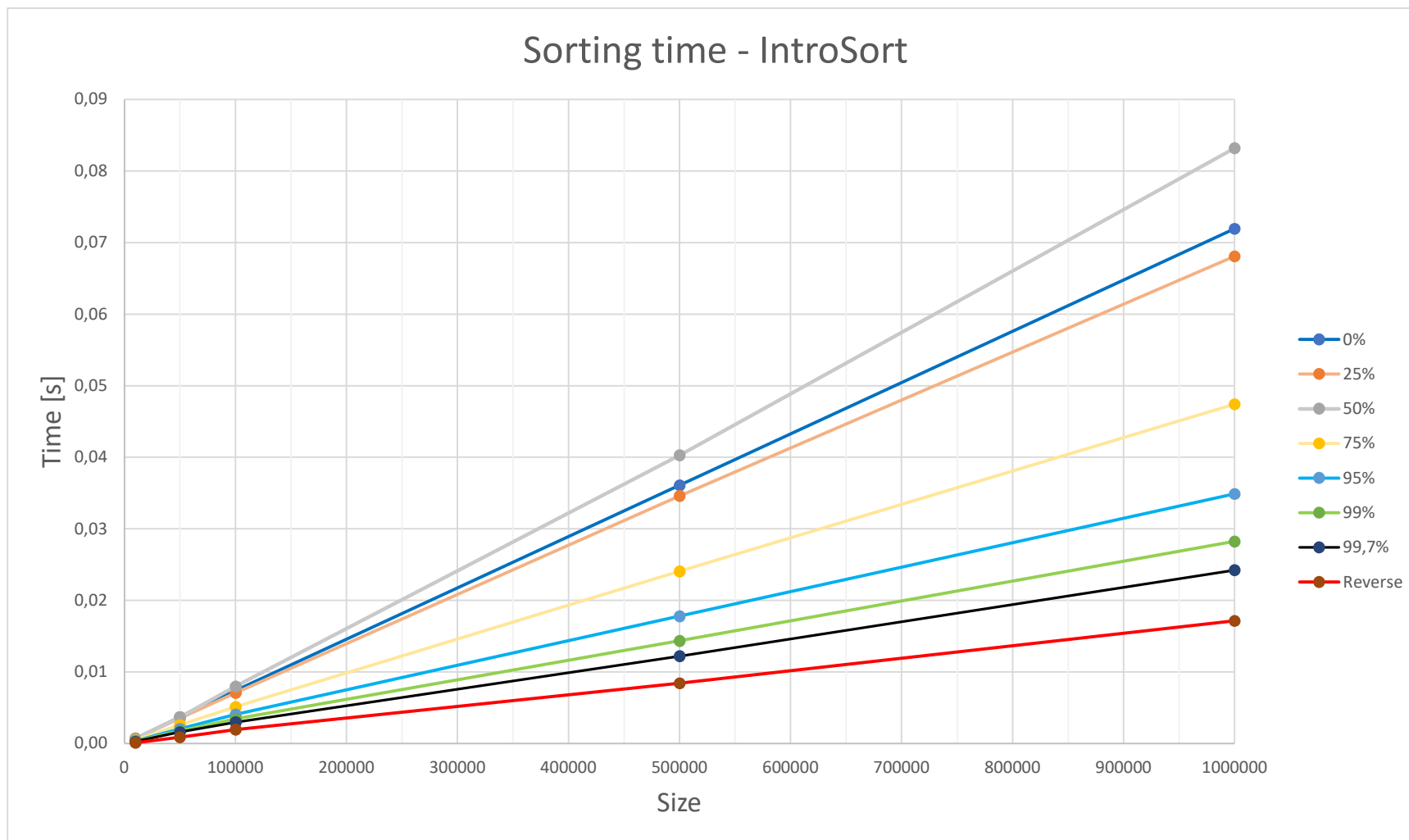
Percent of sorted array	Name	Size	Time[s]	Size	Time[s]	Size	Time[s]	Size	Time[s]	Size	Time[s]
0%	QuickSort	10000	0,00062	50000	0,0033	100000	0,0066	500000	0,033	1000000	0,066
25%			0,00056		0,0031		0,0065		0,032		0,062
50%			0,00058		0,0033		0,0073		0,040		0,112
75%			0,00045		0,0023		0,0046		0,023		0,044
95%			0,00031		0,0017		0,0032		0,014		0,028
99%			0,00024		0,0013		0,0027		0,011		0,022
99,7%			0,00024		0,0012		0,0024		0,010		0,020
100% Reverse			0,00012		0,0009		0,0019		0,008		0,017



Wykres 2 Czasy sortowania tablic przy użyciu sortowania szybkiego

Tabela 3 Czasy sortowania tablic przy użyciu sortowania introspektywnego

Percent of sorted array	Name	Size	Time[s]	Size	Time[s]	Size	Time[s]	Size	Time[s]	Size	Time[s]
0%	IntroSort	10000	0,00071	50000	0,0037	100000	0,0074	500000	0,0361	1000000	0,072
25%			0,00067		0,0035		0,0071		0,0346		0,068
50%			0,00068		0,0037		0,0080		0,0403		0,083
75%			0,00048		0,0026		0,0051		0,0241		0,047
95%			0,00038		0,0021		0,0040		0,0178		0,035
99%			0,00031		0,0017		0,0034		0,0143		0,028
99,7%			0,00027		0,0016		0,0030		0,0122		0,024
100% Reverse			0,00012		0,0009		0,0019		0,0084		0,017



Wykres 3 Czasy sortowania tablic przy użyciu sortowania introspektywnego

5. Podsumowanie

Z wykresów wynika, że złożoność każdego z algorytmów jest zbliżona do założonej, czyli $O(n \cdot \log n)$. Wykresy dla sortowania szybkiego i sortowania introspektywnego różnią się nieznacznie, ponieważ ich działanie zależy od danych wejściowych. Algorytmy te mają zbliżone czasy działania, ponieważ działają na podobnej zasadzie. Czas sortowania dla 50% posortowanej tablicy jest dłuższy niż dla tablicy losowej. Dzieje się tak, ponieważ jako element osiowy wybieram element środkowy danego fragmentu, w tym wypadku element środkowy tablicy. Element ten po uprzednim posortowaniu 50% tablicy jest największym z możliwych z tej połowy. Trafiamy więc na pesymistyczny przypadek sortowania szybkiego. Sytuacja ta jest mniej „dotkliwa” w przypadku sortowania introspektywnego. Nie ma aż tak dużej różnicy czasowej, jak w przypadku sortowania szybkiego. Ciekawym przypadkiem dla sortowania introspektywnego jest sytuacja, gdy zamiast kończenia sortowania (gdy rozmiar jest mniejszy bądź równy 1), wykonamy sortowanie szybkie przy rozmiarze ≤ 16 . Otrzymamy wtedy lepsze czasy sortowania.

Wyniki pomiarów czasów dla sortowania przez scalanie nie wykazują żadnych odstępstw. Wynika to z faktu, że dane wejściowe nie mają wpływu na ten algorytm. Wraz ze wzrostem procentu posortowania tablicy, czas sortowania maleje.

Do pomiarów czasów skorzystałam z biblioteki standardowej `std::chrono`. Wszystkie czasy mierzyłam w mikrosekundach, jednak biblioteka ta pozwala na dokładniejsze pomiary.

6. Bibliografia

- <https://en.wikipedia.org/wiki/Introsort>
- <https://en.wikipedia.org/wiki/Quicksort>
- <https://en.wikipedia.org/wiki/Heapsort>
- https://en.wikipedia.org/wiki/Merge_sort
- <http://www.algorytm.edu.pl/algorytmy-maturalne/quick-sort.html>
- Piotr Wróblewski, *Algorytmy, struktury danych i techniki programowania. Wydanie V*, Helion, 2015.