Projekt 1

Algorytmy sortujące

PAMSI

Projektowanie algorytmów i metody sztucznej inteligencji

Prowadzący: **mgr. inż. Magdalena Skoczeń**
Termin zajęć: **Środa $18^{\overline{55}}$**

Piotr Niedziółka, 249023

08.04.2020

1 Wprowadzenie

Celem projektu była implementacja trzech algorytmów sortowania oraz analiza ich efektywności. Wykorzystane algorytmy w tym sprawozdaniu to: **QuickSort** - sortowanie szybkie, **MergeSort** - sortowanie przez scalanie, **IntroSort** - sortowanie introspektywne. Algorytmy zostały przetestowane na *100* tablicach o rozmiarach: *10 000, 50 000, 100 000, 500 000* oraz *1 000 000*, przy warunkach:

- wszystkie elementy losowe
- uporządkowana w *25%, 50%, 75%, 95%, 99%, 99,7%*
- tablica posortowana w odwrotnej kolejności

2 Omówienie algorytmów

a) **MergeSort** - Sortowanie przez scalanie

Algorytm sortowania stosujący zasadę **"dziel i zwyciężaj"**, sortowana tablica jest dzielona rekurencyjnie na dwie podtablice, dopóki nie uzyska się tablic jednoelementowych. Następnie tablice te są w odpowiedni sposób scalane, co pozwala na szybkie posortowanie tablicy. Jest to bardzo wydajny algorytm, stosujące dzielenie dużego problemu na mniejsze - łatwiejsze do rozwiązania.

Złożoność obliczeniowa:
Średni przypadek: ***O(n log n)***
Najgorszy przypadek: ***O(n log n)***

b) **QuickSort** - Sortowanie szybkie

Jeden z najpopularniejszych algorytmów, który słynie, jak nazwa mówi, ze swojej szybkości. Wybierany jest jeden element, a następnie tablica jest dzielona na dwie podtablice. Pierwsza z nich to elementy mniejsze od wybranego elementu, a druga z nich to elementy większe lub równe wybranemu elementowi - znanemu jako **pivot**. Proces ten jest wykonywany, aż do uzyskania tablic jednoelementowych. Sortowanie jest w pewien sposób ukryte, jednocześnie najgorszym przypadkiem będzie to, gdy będziemy wybierać za każdym razem skrajne wartości tablicy.

Złożoność obliczeniowa:
Średni przypadek: ***O(n log n)***
Najgorszy przypadek: $\mathbf{O(n^2)}$

c) **IntroSort** - Sortowanie introspektywne

Odmiana sortowania hybrydowego - połączenie dwóch sortowań: sortowanie szybkie, sortowanie przez kopcowanie. W tym algorytmie został wyeliminowany problem złożoności obliczeniowej $O(n^2)$ gdy zostanie wybrany najmniejszy / największy element jako **pivot**. I właśnie to jest głównym założeniem tego algorytmu - logarytmiczna złożoność obliczeniowa.W algorytmie IntroSort wykorzystuje się różne algorytmy sortujące, co sprawia, że program staje się skomplikowany. W trakcie implementacji można popełnić wiele błędów, dlatego trzeba być ostrożnym implementując ten algorytm.

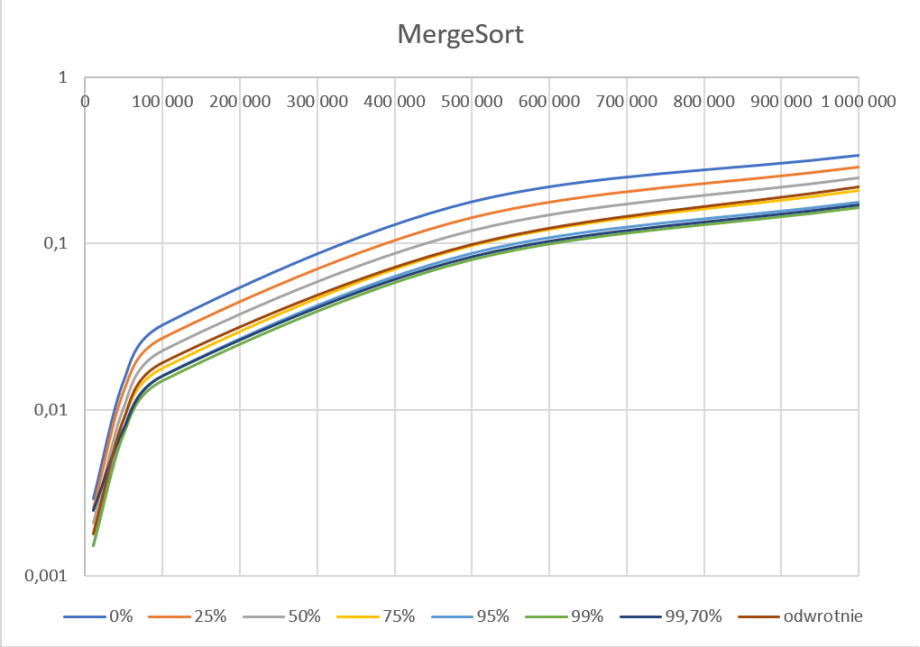
Złożoność obliczeniowa:
Średni przypadek: ***O(n log n)***
Najgorszy przypadek: ***O(n log n)***

3 Przebieg badanych algorytmów

- MergeSort

MergeSort					
	10 000	50 000	100 000	500 000	1 000 000
0%	0,0029	1,52E-02	0,03228	0,17779	0,3379
25%	0,00254	0,01316	0,02702	0,14292	0,28752
50%	0,00208	0,01054	0,02274	0,11992	0,24882
75%	0,0018	0,0089	0,01778	0,09726	0,20898
95%	0,00152	0,00778	0,016	0,087	0,17622
99%	0,00152	0,00726	0,01496	0,07992	0,1642
99,70%	0,00248	0,00782	0,01602	0,08344	0,1709
odwrotnie	0,00178	0,00894	0,01924	0,09924	0,22056

Rysunek 1: Tabela czasów MergeSort [s]

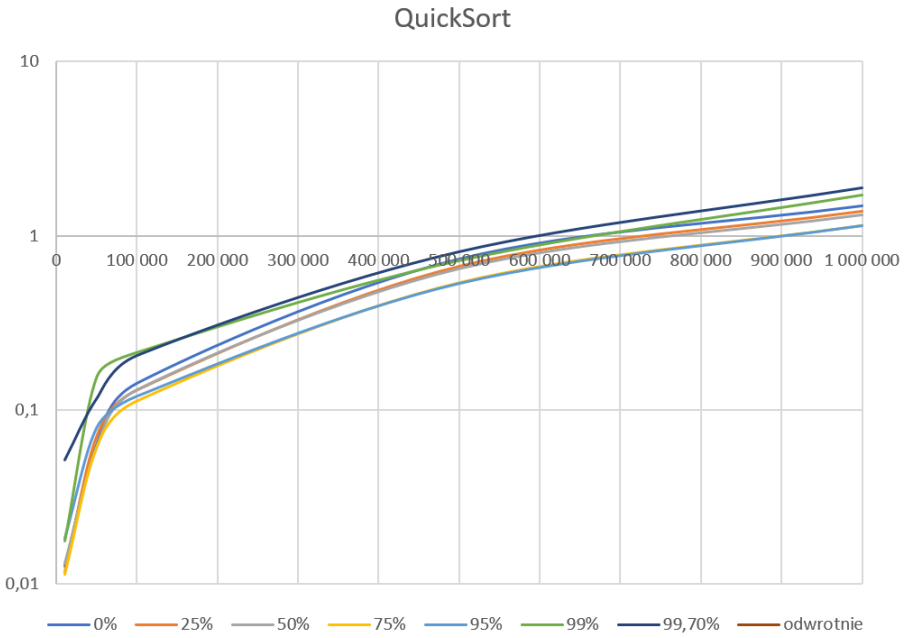


Rysunek 2: Wykres logarytmiczny MergeSort

- QuickSort

QuickSort					
	10 000	50 000	100 000	500 000	1 000 000
0%	0,01256	0,06362	0,14194	0,73403	1,48296
25%	0,01	0,07104	0,12974	0,66742	1,3827
50%	0,01306	0,06152	0,13046	0,64732	1,31898
75%	0,0113	0,06144	0,11232	0,5365	1,14028
95%	0,01838	0,07976	0,12006	0,53064	1,137
99%	0,01758	0,15526	0,21312	0,71734	1,70664
99,70%	0,0515	0,11686	0,20562	0,81235	1,89231
odwrotnie	x	x	x	x	x

Rysunek 3: Tabela czasów QuickSort [s]

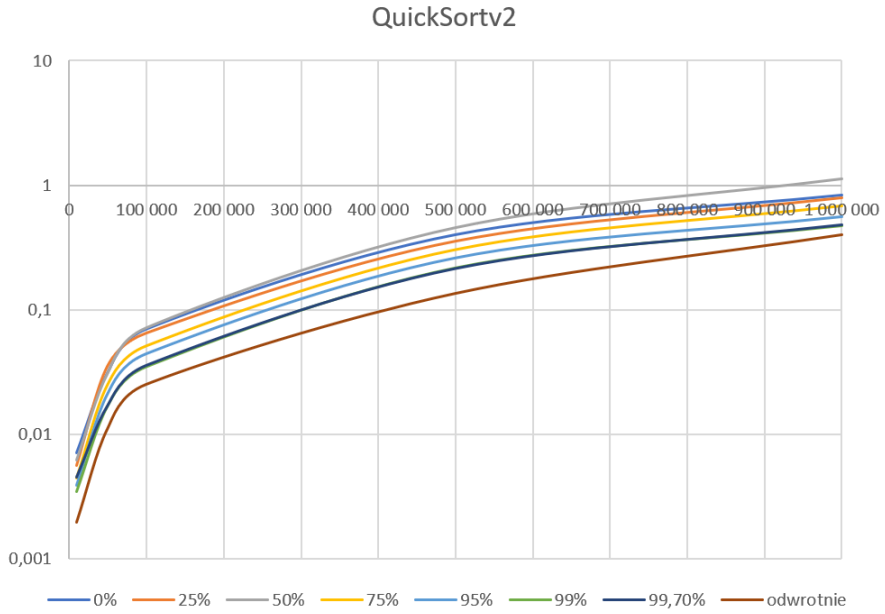


Rysunek 4: Wykres logarytmiczny QuickSort

- QuickSortv2

QuickSortv2					
	10 000	50 000	100 000	500 000	1 000 000
0%	0,00718	3,15E-02	0,07056	0,40334	0,83848
25%	0,00566	0,03564	0,06546	0,35826	0,8
50%	0,00624	0,031	0,07208	0,46046	1,13696
75%	0,0045	0,02512	0,05138	0,30294	0,67876
95%	0,00392	0,02118	0,04438	0,26022	0,55714
99%	0,0035	0,01688	0,03492	0,21596	0,4705
99,70%	0,00456	0,01708	0,03596	0,21522	0,48282
odwrotnie	0,00198	0,01114	0,02524	0,13524	0,39962

Rysunek 5: Tabela czasów QuickvSortv2 [s]

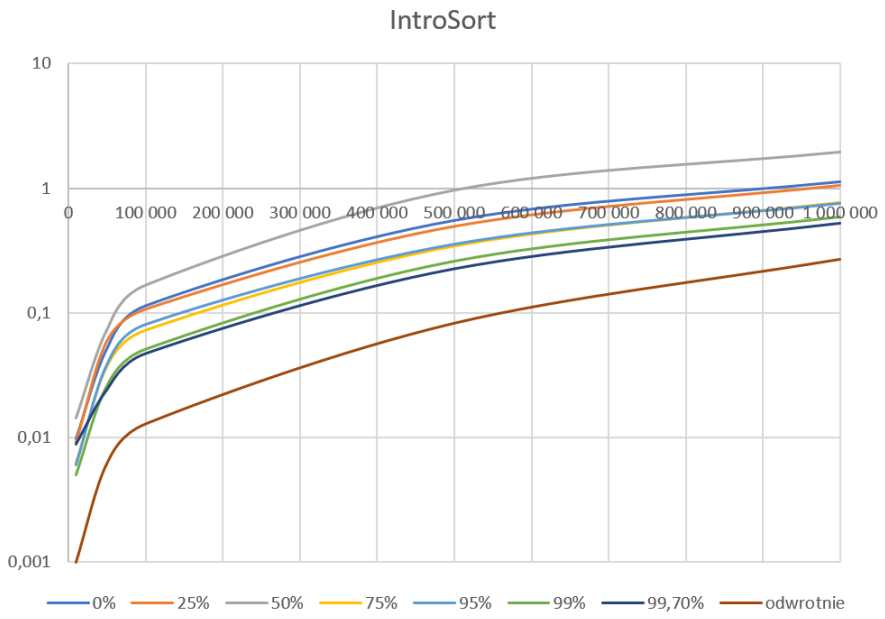


Rysunek 6: Wykres logarytmiczny QuickSortv2

- IntroSort

Introsort					
	10 000	50 000	100 000	500 000	1 000 000
0%	0,00972	5,14E-02	0,11376	0,55192	1,13E+00
25%	0,00908	0,05924	0,10688	0,49528	1,0581
50%	0,01442	0,0733	0,1671	0,96372	1,95038
75%	0,0062	0,03718	0,07238	0,34352	0,76802
95%	0,00598	0,03812	0,07996	0,35258	0,74898
99%	0,00502	0,02592	0,05088	0,25794	0,58352
99,70%	0,00876	0,02396	0,04672	0,22376	0,51948
odwrotnie	0,001	0,0062	0,01288	0,08288	0,2701

Rysunek 7: Tabela czasów IntroSort [s]



Rysunek 8: Wykres logarytmiczny IntrokSort

4 Podsumowanie

MergeSort jest najszybszym algorytmem we wszystkich testach. Drugim pod względem efektywności jest QuickSortv2, następnie IntroSort i najwolniejszy QuickSort. Jest to spowodowane, że MegrgeSort działa bardzo szybko, ale potrzebuje więcej pamięci niż pozostałe algorytmy. IntroSort jest szybszy niż QuickSort - ponieważ został tam wyeliminowany problem kwadratowej złożoności obliczeniowej w najgorszym przypadku, gdy wybierany jest za każdym razem największy lub namniejszy element tablicy. QuickSortv2 jest szybszy niż pozostałe dwa, gdyż został zaimplementowany tak, by zaczynał zarówno z początku i końca tablicy jednocześnie, co pozwala na szybsze posortowanie.

W mojej inkrementacji dla Quicksorta nie działa odwracanie, jest to spowodowane przepełnieniem pamięci.

Najszybszy czas sortowania notujemy dla tablic posortowanych malejąco poprzez algorytm IntroSort. Dla tablic rosnących najszybszym algorytmem jest MergeSort. Algorytmy najszybciej sortują tablicę, która jest uporządkowana, gdy do posortowania jest 99% lub 99,7%, zaś najwolniej dla 25%, 50%, lecz to zależne od algorytmu.

5 Bibliografia

1. `www.stackoverflow.com`
2. `en.wikipedia.org/wiki/Merge_sort`
3. `en.wikipedia.org/wiki/Quicksort`
4. `en.wikipedia.org/wiki/Introsort`
5. `www.geeksforgeeks.org/`
6. `en.cppreference.com/w/cpp/chrono/c/clock`
7. `en.cppreference.com/w/cpp/algorithm/reverse`
8. `en.cppreference.com/w/cpp/algorithm/sort`