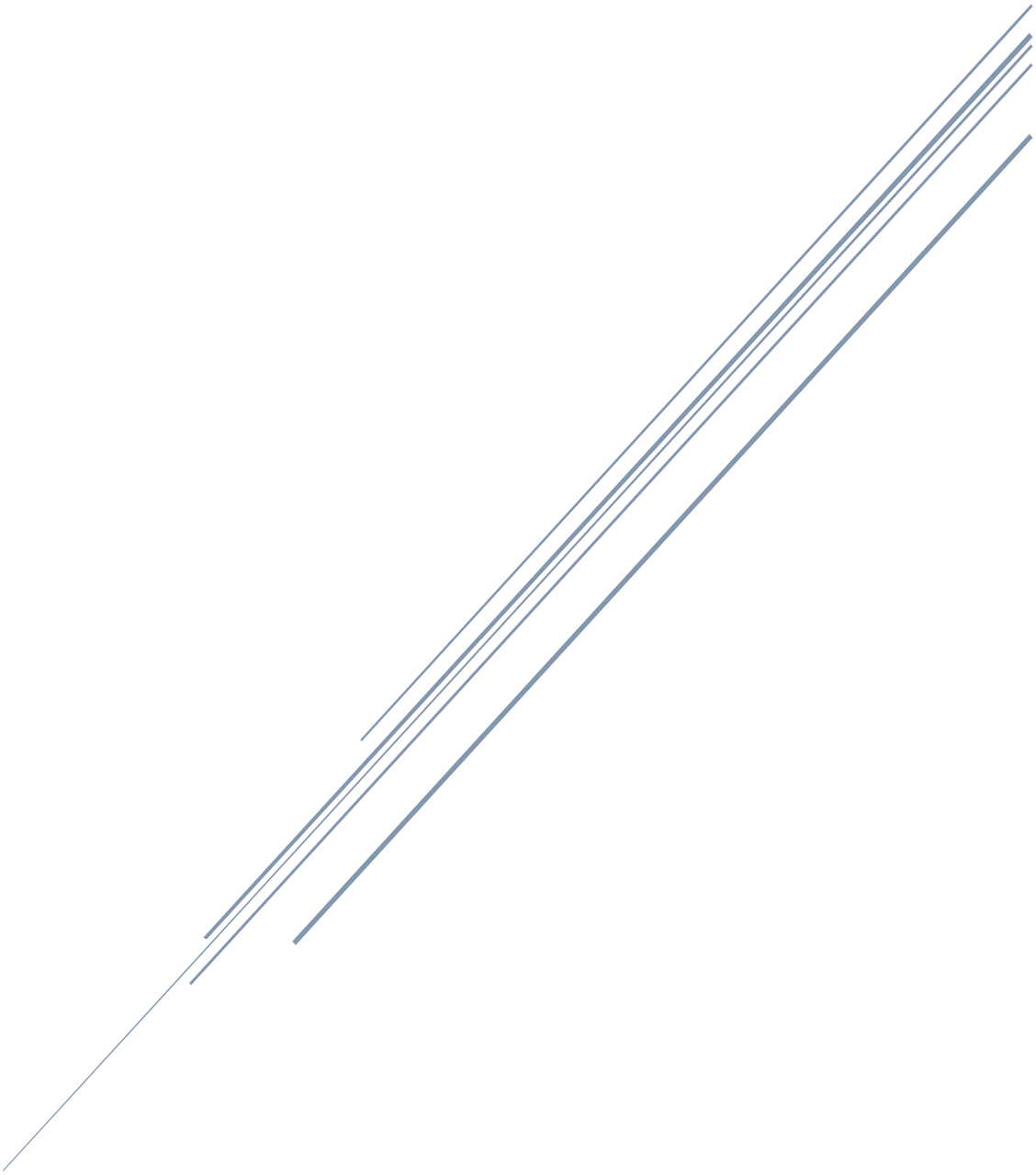
PROJEKT 2

Kacper Kaciłowicz 248951



Mgr inż. Magda Skoczeń

Projektowanie algorytmów i metody sztucznej inteligencji

1. Wprowadzenie

Celem projektu była analiza przebiegu 3 różnych sortowań zaimplementowanych w języku C++. Istotną cechą algorytmu sortującego jest czasowa złożoność obliczeniowa. Określa ona czas wykonania algorytmu, który zależy od ilości danych wejściowych. Dzięki analizie czasowej złożoności obliczeniowej możliwe jest ustalenie klasy złożoności obliczeniowej O(), która służy do określenia wydajności działania algorytmu niezależnie od sprzętu na którym korzysta się z algorytmu. W ramach projektu zostały zrealizowane analizy przebiegu sortowania szybkiego, introspektywnego oraz sortowania przez scalania. Te 3 algorytmy zalicza się do algorytmów szybkich, czyli takich, których klasa czasowej złożoności obliczeniowej wynosi $O(nlogn)$ lub jest nawet lepsza. W celu realizacji badań poddany został analizie czas sortowań po 100 tablic o 5 różnych rozmiarach, jeśli chodzi o liczbę elementów: 10 000, 50 000, 100 000, 500 000, 1 000 000. Wszystkie elementy były rozmiaru całkowitoliczbowego. Ponadto rozpatrzone były różne przypadki:

- Wszystkie elementy tablicy losowe
- Posortowane 25%, 50%, 75%, 95%, 99%, 99.7% początkowych elementów tablicy
- Wszystkie elementy posortowane w odwrotnej kolejności

2. Opis algorytmów sortujących

2.1 Sortowanie szybkie

Jest to algorytm wykorzystujący strategię „divide and conquer", która polega na dzieleniu problemu na mniejsze, a następnie rozwiązywaniu każdego z nich. W tym przypadku do realizacji sortowania niezbędne jest wyznaczenie elementu dzielącego, tzw. „pivota", który rozdzielna zbiór na 2 podzbiory – większe oraz mniejsze od pivota. Następnie zostaje wykonane rekurencyjne wywołanie funkcji dla kolejnych podzbiorów. Złożoność obliczeniowa tego algorytmu zależy przede wszystkim od wybrania elementu dzielącego. Najlepszy przypadek występuje, gdy pivot przy każdym wywołaniu jest medianą sortowanego fragmentu tablicy, a jego złożoność obliczeniowa wynosi $O(nlogn)$. Przypadek średni posiada złożoność obliczeniową $1{,}39nlogn$, zatem można dojść do wniosku, że jest mu bliżej do przypadku najlepszego, niż najgorszego, który może mieć miejsce, gdy pivot za każdym razem jest elementem największym, bądź najmniejszym, a jego złożoność obliczeniowa jest równa $O(n^2)$. W celu zmniejszenia prawdopodobieństwa wystąpienia pesymistycznego przypadku stosuje się m .in. algorytm wyboru pivota pt. „Median of three", polegający na pobraniu 3 elementów tablicy (pierwszego, ostatniego i środkowego), ustawieniu każdego z nich wg wybranej hierarchii, czyli przykładowo najmniejszy na początku, średni w środku oraz największy na końcu i wybraniu elementu rozdzielającego, który jest właśnie medianą z tych trzech elementów.

2.2 Sortowanie przez scalanie

Podobnie jak algorytm sortowania szybkiego, algorytm sortowania przez scalanie wykorzystuje strategię „divide and conquer". Polega on na rekurencyjnym podziale tablicy, aż do uzyskania jednoelementowych zbiorów, które są uznawane za posortowane, a następnie scalania ich z powrotem do tablicy wraz z dokonywaniem operacji porównania zbiorów w celu zapisania danych w poprawnej kolejności. Sortowanie przez scalanie posiada taką samą złożoność obliczeniową dla wszystkich przypadków i wynosi ona $O(nlogn)$. Wynika to z tego, że za każdym razem zbiór jest dzielony przez dwa i tworzy się drzewo binarne, a jego wysokość zależy logarytmicznie od liczby elementów.

2.3 Sortowanie introspektywne

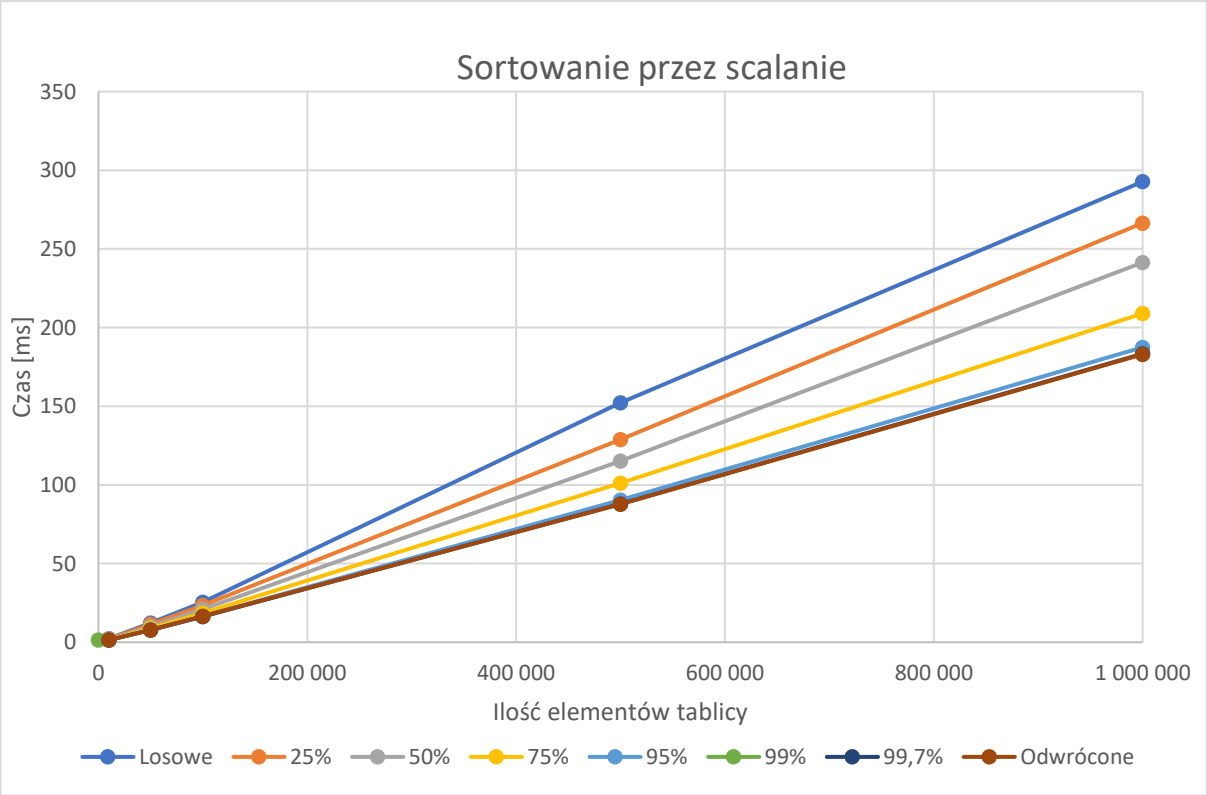
Jest to tzw. sortowanie hybrydowe, czyli takie, które wykorzystuje elementy różnych sortowań. W tym przypadku występuje hybryda sortowania przez kopcowanie oraz sortowania szybkiego. Ma ona na celu wykluczenie przypadku pesymistycznego sortowania szybkiego. W tym celu bada się głębokość rekurencji ze wzoru $M = 2 \times log_2 n$, gdzie n oznacza długość tablicy. W przypadku, gdy $M > 0$, algorytm wykonuje sortowanie szybkie oraz wartość $M = M - 1$ przy kolejnej rekurencji, a gdy $M = 0$, wywoływane jest sortowanie przez kopcowanie, którego złożoność obliczeniowa jest równa zawsze $O(nlogn)$.

3. Prezentacja wyników

Przedstawione wykresy oraz tabele dotyczą uśrednionego czasu wykonania dla 1 tablicy danego rozmiaru. Jednostką czasu w każdym przypadku jest ms.

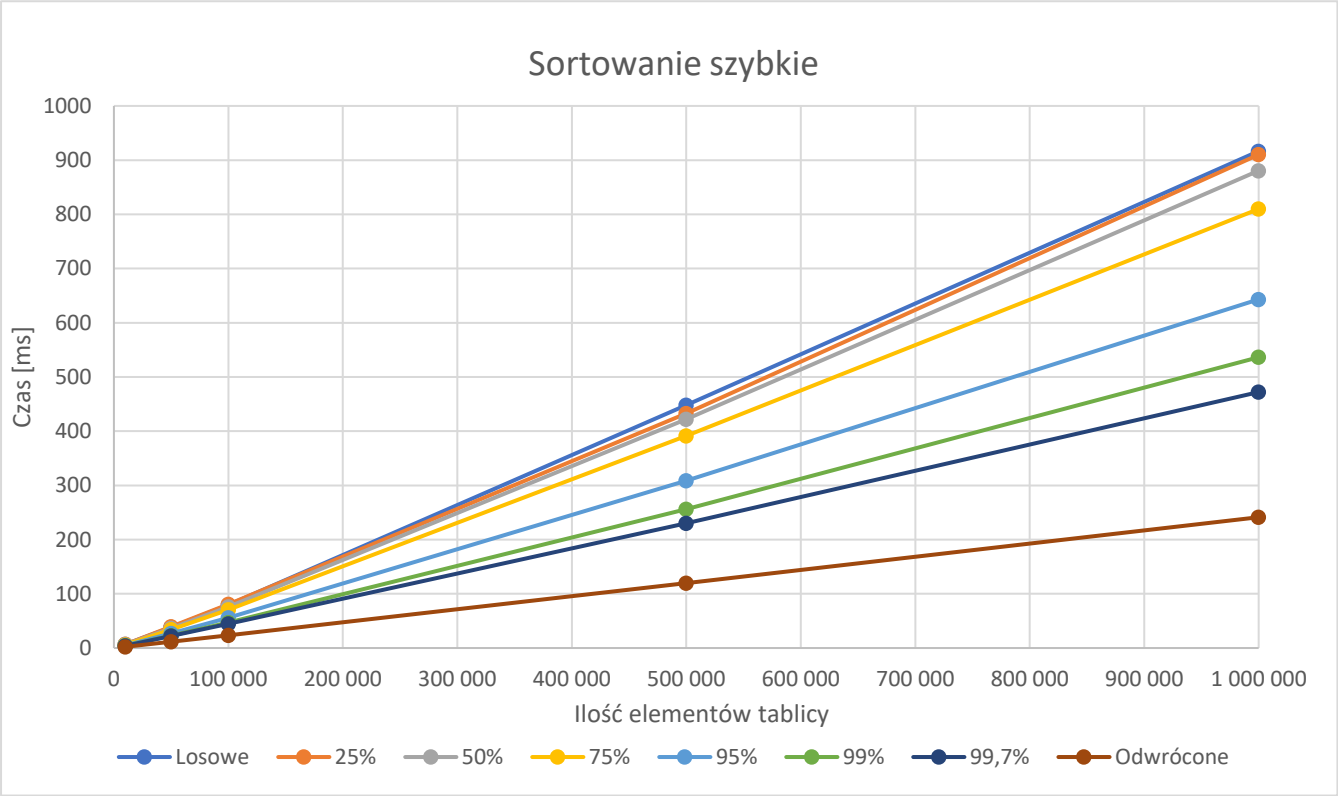
- **Sortowanie przez scalanie**

Liczba elementów	**Poziom posortowania początkowego**							
	0%	**25%**	**50%**	**75%**	**95%**	**99%**	**99,70%**	**Odwrócone**
10 000	2,30894	2,0858	1,83897	1,64045	1,46705	1,48653	1,42757	1,46787
50 000	12,24830	11,5079	9,97385	8,98384	7,90959	7,75630	7,76178	7,84976
100 000	25,43850	23,5132	20,98150	18,51660	16,45390	16,21120	16,33600	16,60340
500 000	152,16100	128,7590	115,13400	101,08400	90,45330	88,12530	87,98530	87,78500
1 000 000	292,76700	266,4420	241,40400	208,97500	187,48800	181,75700	183,37000	183,02300



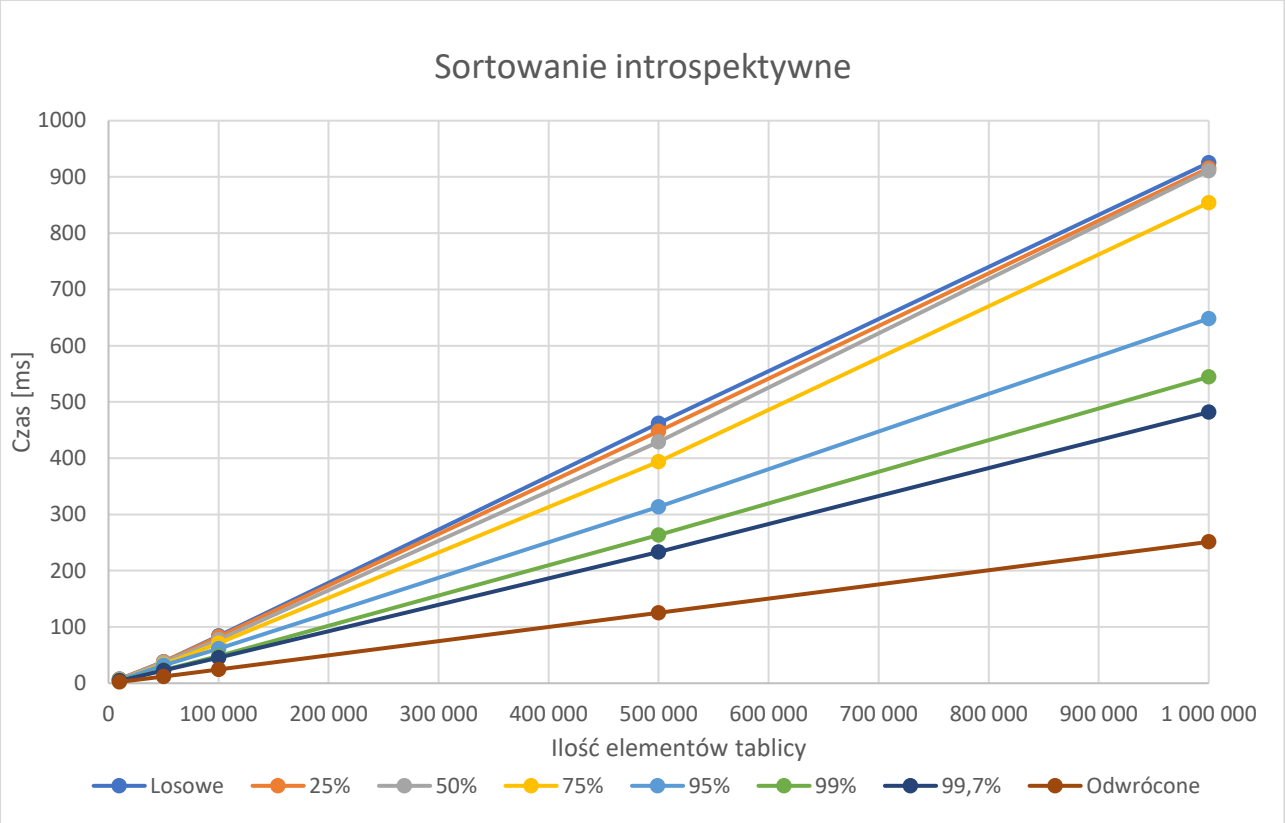
- **Sortowanie szybkie**

Liczba elementów	Poziom posortowania początkowego							
	0%	25%	50%	75%	95%	99%	99,70%	Odwrócone
10 000	6,80091	6,75437	6,40009	5,9889	4,82628	4,34886	4,11069	2,26021
50 000	38,1783	39,07950	36,2019	33,9113	26,90970	23,08300	22,17340	11,43570
100 000	79,39540	80,44460	75,95470	70,8281	56,01510	47,07330	44,79973	23,36180
500 000	448,26500	432,63400	422,04300	391,0540	308,48000	256,00000	230,23800	119,83800
1 000 000	916,56600	910,21900	880,56200	809,8790	643,17000	536,40200	472,14800	241,30400



- **Sortowanie introspektywne**

Liczba elementów	Poziom posortowania początkowego							
	0%	25%	50%	75%	95%	99%	99,70%	Odwrócone
10 000	7,21078	6,79838	6,58589	6,01436	5,75825	4,47423	4,25501	2,35304
50 000	38,37190	38,13870	37,00710	33,80140	31,99630	23,41540	22,50040	12,11420
100 000	84,02730	82,41900	77,14800	71,05370	61,69580	48,18300	45,75990	24,36240
500 000	461,99800	447,54400	428,97700	393,67800	313,66100	263,51700	233,22600	125,43300
1 000 000	925,15300	915,73900	911,15100	854,24600	648,18900	544,48100	481,81400	251,23600



Wnioski z zaprezentowanych wyników:

- Algorytmy zachowują swoją złożoność obliczeniową
- W każdym z przypadków sortowanie przez scalanie działa najszybciej, a sortowanie szybkie oraz introspektywne mają podobny czas działania
- Sortowanie wykonuje się najszybciej w każdym z przypadków dla danych posortowanych w odwrotnej kolejności, a najwolniej gdy dane są zupełnie losowe

4. Podsumowanie

Pomimo swojej nazwy algorytm sortowania szybkiego, nie jest sortowaniem najszybszym. W przypadku ogólnym lepiej sprawdza się sortowanie przez scalanie, które posiada zawsze taką samą złożoność obliczeniową. W przypadku moich testów sortowanie introspektywne wykonuje się delikatnie dłużej niż sortowanie szybkie, ze względu na wpływ sortowania przez kopcowanie oraz zastosowania algorytmów pomocniczych, takich jak „Median of three” w funkcji obsługującej dzielenie zbioru i wyznaczania pivota, które miały na celu zminimalizowanie prawdopodobieństwa wystąpienia przypadku najgorszego.

Warto również zaznaczyć, że szybkość wykonywania wszystkich algorytmów sortowania zależy od wielkości zbiorów oraz ich pierwotnego uporządkowania.

5. Bibliografia

1) https://en.wikipedia.org/wiki/Quicksort
2) https://en.wikipedia.org/wiki/Introsort
3) http://informatyka.2ap.pl/ftp/3d/algorytmy/podr%C4%99cznik_algorytmy.pdf?fbclid=IwAR1FtqA2Aeq6B5uqZjyVnvV3YnmfF0SZTSqiiNq7P05BNwCj31esVcqsViA
4) https://stackoverflow.com/questions/13445688/how-to-generate-a-random-number-in-c