Projektowanie algorytmów i metody sztucznej inteligencji

Projekt 1

Termin zajęć: Wykonał:

Czwartek 9:15 Eryk Matecki 249484

Prowadzący: dr inż. Łukasz Jeleń

1. Wstęp

Projekt polegał na przeanalizowaniu trzech algorytmów sortowania pod względem ich wydajności, czyli czasu w jakim są one w stanie posortować określone ilości danych. W wykonywanym przeze mnie projekcie testowałem następujące rodzaje sortowań:

- sortowanie szybkie (Quicksort),
- sortowanie przez scalanie (Merge sort),
- sortowanie introspektywne (Introsort).

Podawane dane w poszczególnych przypadkach różnią się od siebie ilością oraz ułożeniem względem siebie, czyli procentem wstępnego posortowania. Rozważane były następujące ilości danych (100 tablic o rozmiarach):

- 10 000,
- 50 000,
- 100 000,
- 500 000,
- 1 000 000,

oraz następujących stopniach wstępnego posortowania:

- wszystkie elementy losowe,
- 25 %,
- 50 %,
- 75 %,
- 95 %,
- 99 %,
- 99,7 %,
- wszystkie elementy posortowane, ale w odwrotnej kolejności.

2. Badane algorytmy

a) sortowanie szybkie (Quicksort)

Algorytm stosuje metodę dziel i rządź. Na początku wybierany jest element rozdzielający, tzw. pivot, który rozdziela zestaw danych na dwie części. Pierwsza z nich zawiera elementy mniejsze od elementu osiowego, druga elementy większe lub równe, element osiowy znajdzie się między nimi. Wykonywana jest rekurencyjnie powyższa czynność dla kolejnych fragmentów zestawu danych aż do osiągnięcia zbioru, który jest uważany za posortowany. Wybór elementu osiowego wpływa na równomierność podziału na podtablice (najprostszy wariant – wybór pierwszego elementu tablicy – nie sprawdza się w przypadku, gdy tablica jest już prawie uporządkowana).

Złożoność obliczeniowa Quicksort'a wygląda następująco: •

- Najlepszy przypadek → $\mathcal{O}(n\log n)$
- Średni przypadek → $\mathcal{O}(n\log n)$
- Najgorszy przypadek → $\mathcal{O}(n^2)$

Złożoność pamięciowa:

- $\mathcal{O}(\log n)$ | $\mathcal{O}(n)$

Quicksort nie potrzebuje dodatkowej struktury danych żeby sortować, tzn. sortuje on „w miejscu". Złożoność pamięciowa $\mathcal{O}(n)$ wynika z występowania pesymistycznego przypadku. QuickSort jest niestabilny. Z uwagi na możliwość manipulacji wyboru pivota, algorytm ten może działać z bardzo różną wydajnością.

b) sortowanie przez scalanie (Merge sort)

Algorytm również stosuje metodę dziel i rządź. Sortowana tablica dzielona jest rekurencyjnie na dwie podtablice aż do uzyskania tablic jednoelementowych. Następnie podtablice te są scalane w odpowiedni sposób, dający w rezultacie tablicę posortowaną.

Złożoność obliczeniowa Mergesort'a wygląda następująco:

- Najlepszy przypadek → $\mathcal{O}(n\log n)$
- Średni przypadek → $\mathcal{O}(n\log n)$
- Najgorszy przypadek → $\mathcal{O}(n\log n)$

Złożoność pamięciowa: • $\mathcal{O}(n)$

Mergesort jest algorytmem stabilnym, co oznacza, że jeśli przed sortowaniem mamy elementy o takiej samej wartości, które są względem siebie ułożone w konkretnej kolejności to po sortowaniu te elementy będą nadal w takiej samej kolejności. Złożoność pamięciowa Mergesort'a wynika z potrzeby posiadania dodatkowej tymczasowej struktury danych.

c) sortowanie introspektywne (Introsort)

Jest to metoda hybrydowa, będąca połączeniem sortowania szybkiego i sortowania przez kopcowanie. Sortowanie introspektywne pozwala uniknąć najgorszego przypadku dla sortowania szybkiego (nierównomierny podział tablicy w przypadku, gdy jako element osiowy zostanie wybrany element najmniejszy lub największy). Działanie IntroSort'a można opisać w kilku prostych krokach:

- Partycjonuj dane w taki sam sposób jak QuickSort, czyli wykorzystując element rozdzielający,
- Jeżeli głębokość rekurencji (dozwolona głębokość wywołań jest równa 0 przełącz się na Heapsort'a,
- Jeżeli pozostała ilość danych w określonym fragmencie jest mniejsza od 16 to należy przełączyć się na InsertionSort'a i posortować fragment do końca.

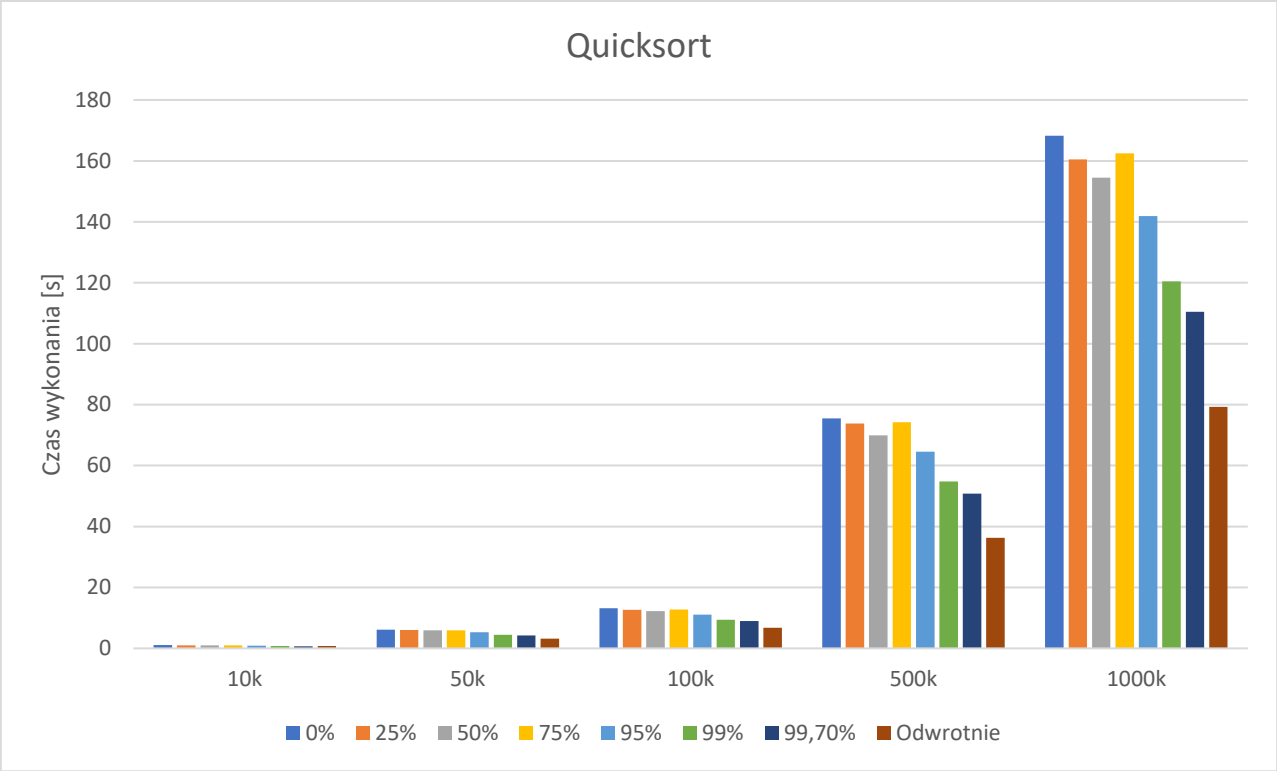
Złożoność obliczeniowa IntroSort'a wygląda następująco:

- Najlepszy przypadek → $\mathcal{O}(n\log n)$
- Średni przypadek → $\mathcal{O}(n\log n)$
- Najgorszy przypadek → $\mathcal{O}(n\log n)$

Złożoność pamięciowa: • $\mathcal{O}(\log n)$

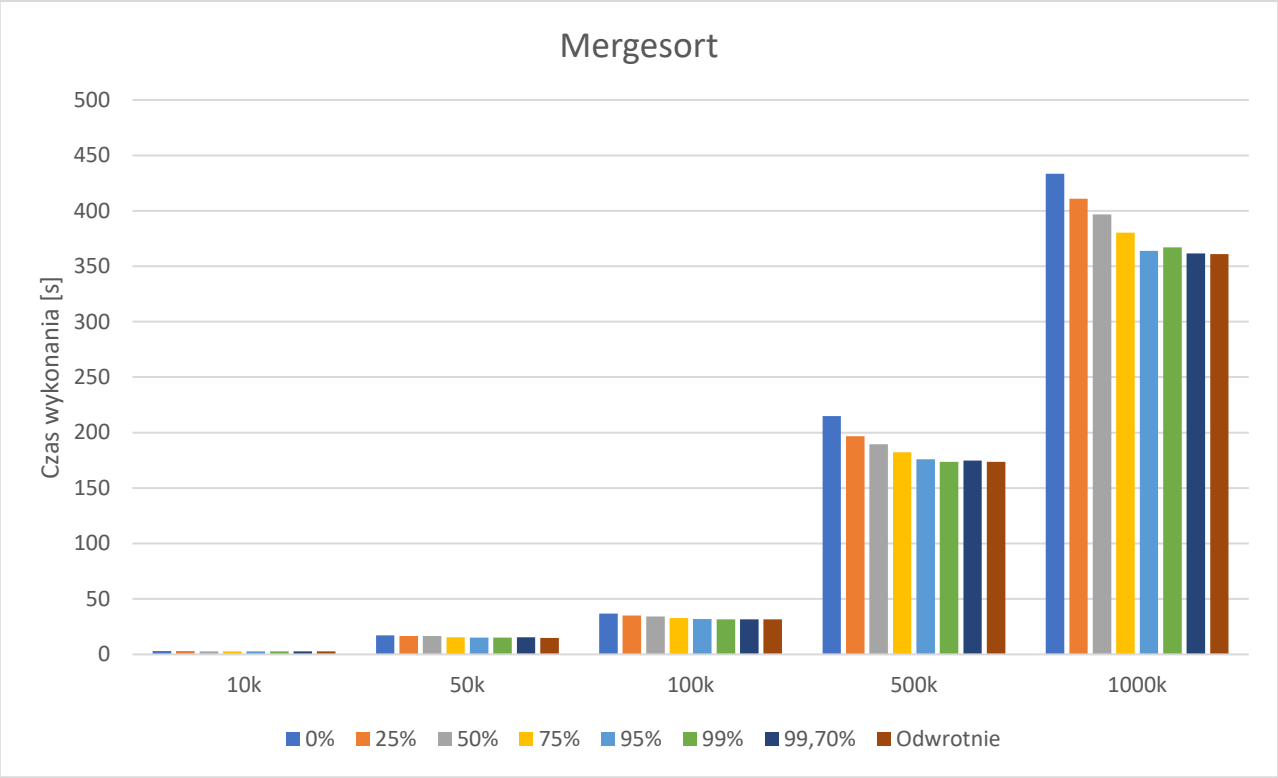
Intro również nie potrzebuje dodatkowej struktury danych żeby sortować oraz jest niestabilny. Algorytm ten został stworzony aby wyeliminować najgorszy przypadek złożoności obliczeniowej Quicksorta. Warto wspomnieć że jest to algorytm którego używa język programowania c++ pod instrukcją std::sort() w nagłówku <algorithm>.

3. Wyniki dla różnego stopnia posortowania



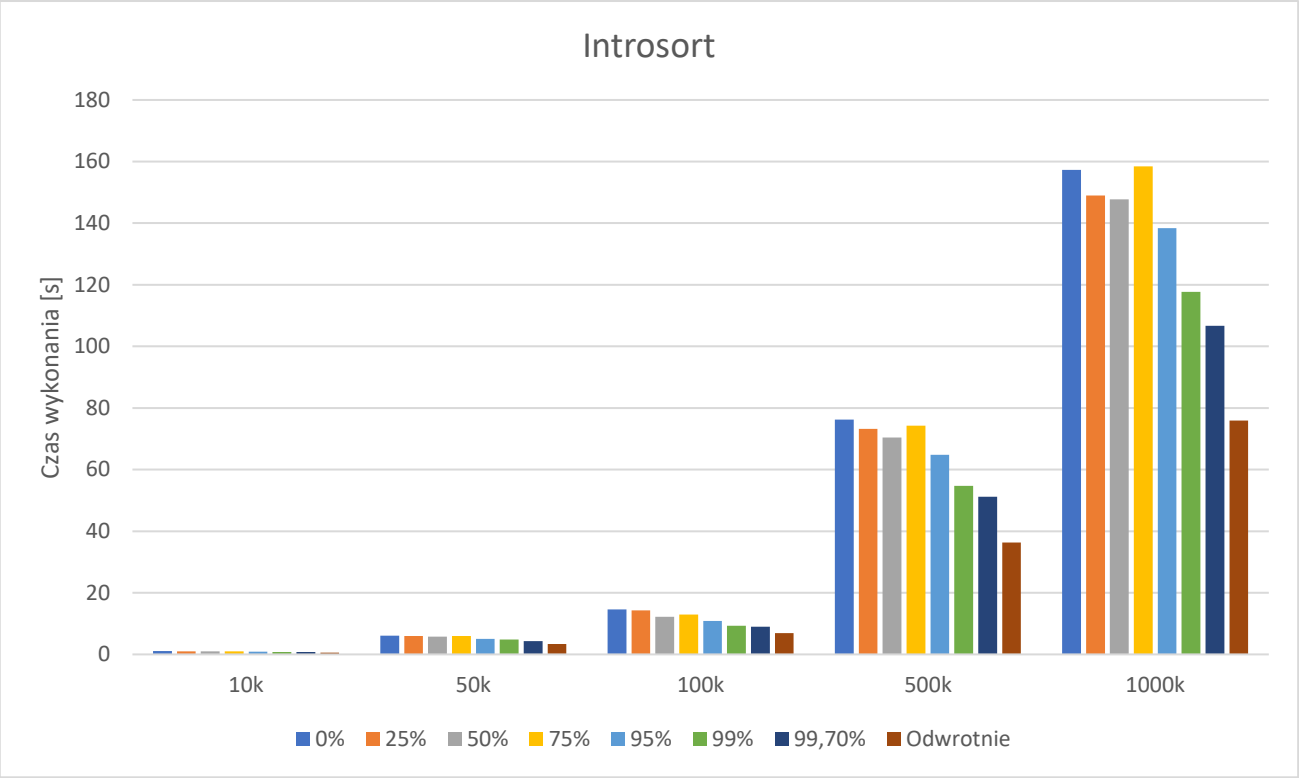
	0%	25%	50%	75%	95%	99%	99,7%	Odwrotnie
10k elementów	1,045	1,01311	0,981901	1,00023	0,83785	0,747263	0,726891	0,763594
50k elementów	6,16541	6,01145	5,9368	5,91946	5,25948	4,40929	4,21228	3,21435
100k elementów	13,1421	12,6815	12,1856	12,7854	11,0919	9,38322	8,95625	6,74387
500k elementów	75,498	73,753	69,925	74,2062	64,5112	54,8299	50,8064	36,3111
1000k elementów	168,294	160,455	154,501	162,517	141,887	120,409	110,511	79,2936

Wykres oraz tabela przedstawiają czasy poszczególnych sortowań podane w sekundach dla sortowania szybkiego.



	0%	25%	50%	75%	95%	99%	99,7%	Odwrotnie
10k elementów	3,02958	2,95066	2,85805	2,7331	2,71411	2,66964	2,71194	2,67267
50k elementów	17,2417	16,7312	16,5633	15,5574	15,2002	15,0679	15,4694	14,8298
100k elementów	36,7027	35,2117	34,2818	32,894	31,7971	31,6795	31,7016	31,5322
500k elementów	214,778	196,654	189,468	182,212	176,057	173,729	174,93	173,603
1000k elementów	433,403	410,853	396,886	380,355	363,918	367,095	361,521	361,111

Wykres oraz tabela przedstawiają czasy poszczególnych sortowań podane w sekundach dla sortowania przez scalanie.



	0%	25%	50%	75%	95%	99%	99,7%	Odwrotnie
10k elementów	1,0598	1,02769	0,985746	1,01671	0,834078	0,74411	0,738627	0,543369
50k elementów	6,11887	5,97193	5,73074	5,95222	5,03444	4,87387	4,28791	3,37862
100k elementów	14,6104	14,3356	12,228	12,9319	10,8472	9,34736	8,96814	6,92584
500k elementów	76,2745	73,1665	70,4209	74,2569	64,8241	54,7611	51,1872	36,3106
1000k elementów	157,26	148,984	147,713	158,42	138,335	117,658	106,644	75,9334

Wykres oraz tabela przedstawiają czasy poszczególnych sortowań podane w sekundach dla sortowania introspektywnego.

4. Podsumowanie

Analizując algorytmy można było stwierdzić, że najszybszy powinien być Introsort a najwolniejszy Mergesort. Przyjmuje się, że z uwagi na dobrą kompatybilność z pamięcią podręczną Quicksort jest szybszy niż Mergesort. Kompatybilność ta wynika z faktu, że sortowanie przez scalanie korzysta z dodatkowej struktury danych. Introsort jest szybszy od Quicksorta, ponieważ eliminuje jego pesymistyczny przypadek. Warto przyjrzeć się podpunktowi 75%. Widać na nim odstępstwo od reguły. W przypadku sortowania szybkiego oraz introspektywnego pomimo większego początkowego stopnia posortowania, zajmuje to dłużej niż dla np. 25 i 50 procent. Posortowanie części tablicy skraca czas sortowania w każdym z algorytmów, poza drobnymi odstępstwami jak w przypadku 75% początkowego posortowania w Quicksorcie i Introsorcie. Może to być spowodowane chwilowymi spadkami

wydajności, czy też niedociągnięciami w użytym algorytmie. Ćwiczenie miało na celu przeprowadzenie badań na różnych algorytmach sortujących, porównanie ich efektywności i złożoności obliczeniowej. Zgodnie z przewidywaniami, najmniej efektywnym algorytmem okazało się sortowanie przez scalanie, a najbardziej efektywnym sortowanie introspektywne. To właśnie jego hybrydowa natura pozwala na brak prawdopodobieństwa wybrania złego punktu odniesienia, który może sprawić, że sortowanie szybkie nie będzie wcale lepsze niż sortowanie bąbelkowe. Powyższe wyniki potwierdzają teoretyczne założenia i poprawność zaimplementowanych algorytmów. Należy także pamiętać, że wyniki mogą się różnić w zależności od użytego sprzętu i chwilowego zapotrzebowania na moc obliczeniową podczas pracy użytkownika.