Imię i nazwisko: Adam Bednorz

Numer indeksu: 248955

Dane prowadzącego: dr inż. Krzysztof Halawa

Termin zajęć: Czwartek 17:05

Projektowanie algorytmów i metody sztucznej inteligencji

Projekt 1: Algorytmy sortowania

1. Wprowadzenie:

1.1 Opis projektu:

Projekt polegał na przeanalizowaniu trzech algorytmów sortowania pod względem ich wydajności. Dla każdego algorytmu zostały zmierzone czasy, ile zajmuje im posortowanie poszczególnych porcji danych. W projekcie analizowane były następujące algorytm sortowania:

- **Quicksort**
- **Mergesort**
- **Introsort**

Testy zostały przeprowadzone dla 100 tablic (typu całkowitoliczbowego) o następujących rozmiarach:

- 10 000
- 50 000
- 100 000
- 500 000
- 1000 000

Sortowania wykonano dla następujących przypadków:

- Wszystkie elementy tablicy losowe
- 25%, 50%, 75%, 95%, 99%, 99,7% początkowych elementów tablicy jest już posortowanych oraz dodatkowo w drugiej nieposortowanej podtablicy nie znajduje się element mniejszy od tej posortowanej podtablicy
- Wszystkie elementy posortowane ale w odwrotnej kolejności

1.2 Opis algorytmów sortowania:

Quicksort

W tym algorytmie oryginalna tablica jest dzielona na dwie podtablice, z których pierwsza zawiera elementy mniejsze lub równe od pewnego wybranego klucza tzw. pivota. Druga tablica zawiera elementy większe lub równe od tego elementu osiowego. Obie tablice można posortować niezależnie, jednak zanim zostanie to wykonane, także i te podtablice można poddać podobnemu procesowi podziału.

Złożoność obliczeniowa Quicksort'a:

- Przypadek optymistyczny -> O(n log n)
- Przypadek typowy -> O(n log n)
- Przypadek pesymistyczny (pivot to najmniejsza lub największa wartość) -> O(n^2)

Złożoność pamięciowa:
Złożoność pamięciowa jest zależna od implementacji. QuickSort jest niestabilny.
Z uwagi na możliwość manipulacji wyboru pivota, algorytm ten może działać z bardzo rożną wydajnością.

Mergesort

Podstawowym procesem tego algorytmu jest scalanie dwóch połówek tablicy w jedną posortowaną tablicę. Jednak obie te połówki muszą być już posortowane, co można zapewnić przez scalanie dwóch posortowanych połówek tych połówek. Ten proces dzielenia tablic na połówki jest przerywany w momencie, gdy tablica zawiera mniej niż dwa elementy.

Złożoność obliczeniowa Mergesort'a:

- Optymistyczny przypadek -> O(n log n)
- Typowy przypadek -> O(n log n)
- Pesymistyczny przypadek -> O(n log n)

Złożoność pamięciowa:
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, wynika z potrzeby posiadania dodatkowej tymczasowej struktury danych, zatem złożoność pamięciowa wynosi O(n).

Introsort

Algorytm ten jest hybrydowy, tzn. że składa się on z kilku innych algorytmów sortowania. Na sortowanie introspektywne składa się: Quicksort, Heapsort i Insertionsort. Introsort partycjonuje dane w taki sam sposób jak Quicksort, czyli wykorzystuje funkcje wybierającą pivot i dzielenie tablicy na podtablice. Jeżeli głębokość rekurencji (dozwolona głębokość wywołań rekurencyjnych określana jako współczynnik 2*log n) jest równa 0 przełącz na Heapsort. Jeżeli pozostała ilość danych w określonym fragmencie jest mniejsza od 16 to należy przełączyć się na sortowanie przez wstawianie i posortować już fragment do końca.

Złożoność obliczeniowa Introsorta'a:

- Optymistyczny przypadek -> O(n log n)
- Typowy przypadek -> O(n log n)
- Pesymistyczny przypadek -> O(n log)

Złożoność pamięciowa:

Introsort nie potrzebuje dodatkowej struktury danych, żeby sortować oraz jest niestabilny. Algorytm ten został stworzony, żeby wyeliminować pesymistyczny przypadek Quicksort'a. Złożoność pamięciowa wynosi O (log n).

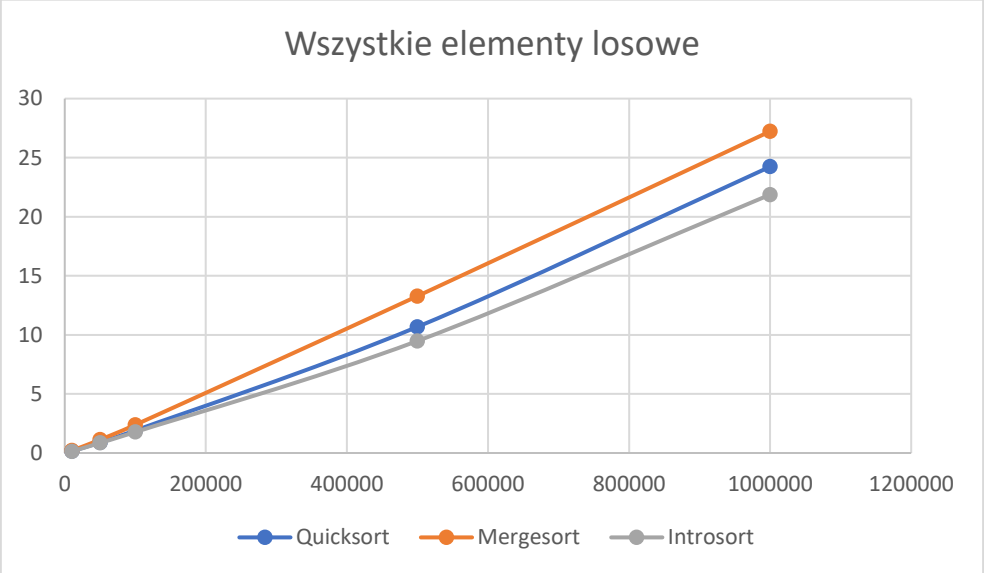
2. Testy:

2.1 Przewidywane wyniki:

Analizując wszystkie trzy algorytmy najszybciej dane sortować powinien Introsort, a najwolniej Mergesort. Szybkość Quicksort'a zależy od wyboru pivota. W projekcie wybieraliśmy środkowy element tablicy jako pivot zatem Quicksort również powinien szybciej sortować dane niż algorytm sortowania przez scalanie, ponieważ jest małe prawdopodobieństwo, że pivot będzie największym lub najmniejszym elementem. Introsort z uwagi na swoją wielozadaniowość oraz przystosowanie do różnych warunków pracy powinien być najlepszym ze wszystkich algorytmów.

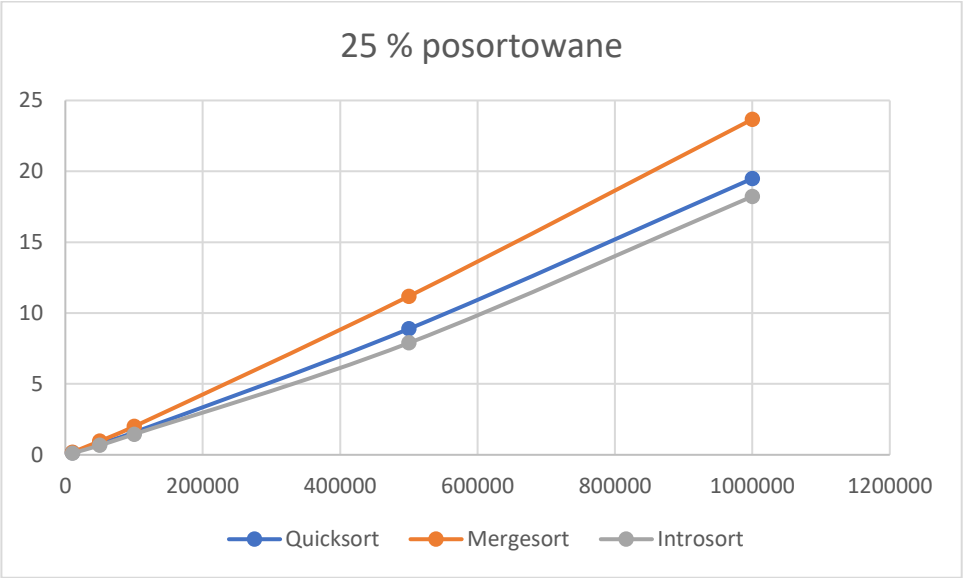
2.2 Przebieg testów:

1) Elementy posorotwane losowo:



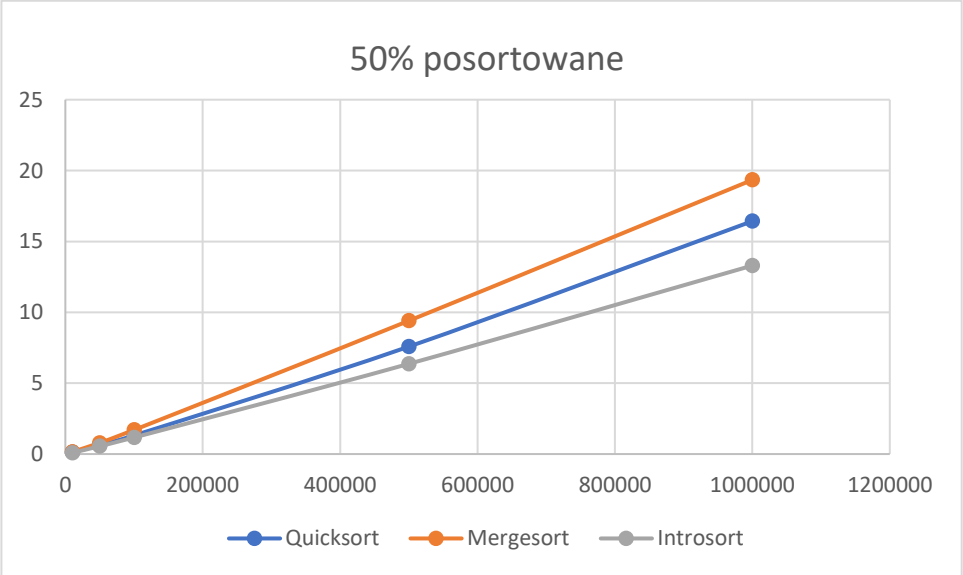
Długość wiersza	10000	50000	100000	500000	1000000
Quick czas[s]	0,152094	0,890385	1,90562	10,6819	24,2356
Merge czas[s]	0,197877	1,12307	2,38446	13,2816	27,2291
Intro czas[s]	0,132714	0,850025	1,77385	9,48551	21,8726

2) 25% pierwszych elementów posortowanych:



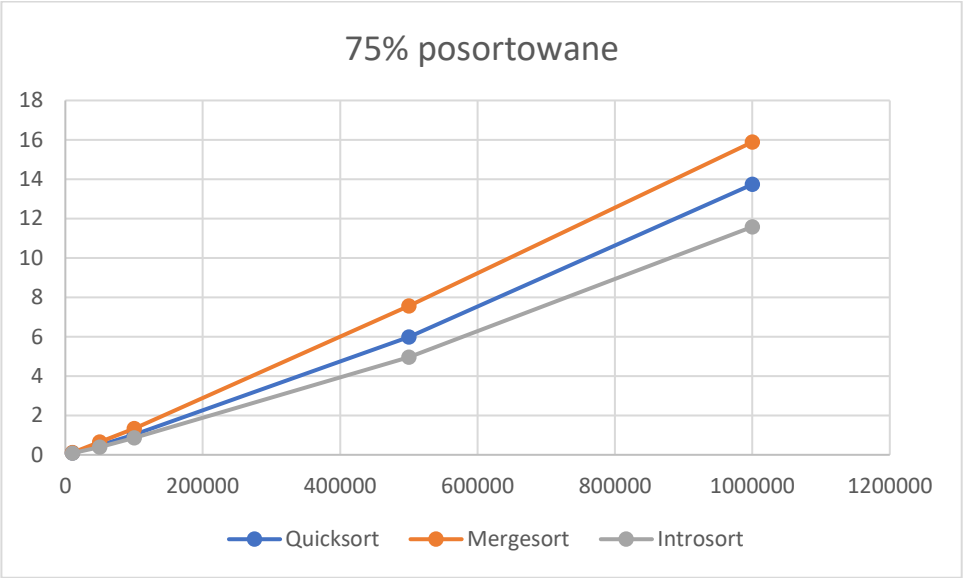
Długość wiersza	10000	50000	100000	500000	1000000
Quicksort czas[s]	0,146952	0,783847	1,58475	8,88699	19,4891
Mergersort czas[s]	0,175149	0,965384	2,00164	11,1873	23,6704
Introsort czas[s]	0,111627	0,66724	1,45212	7,89165	18,2315

3) 50% pierwszych elementów posortowanych:



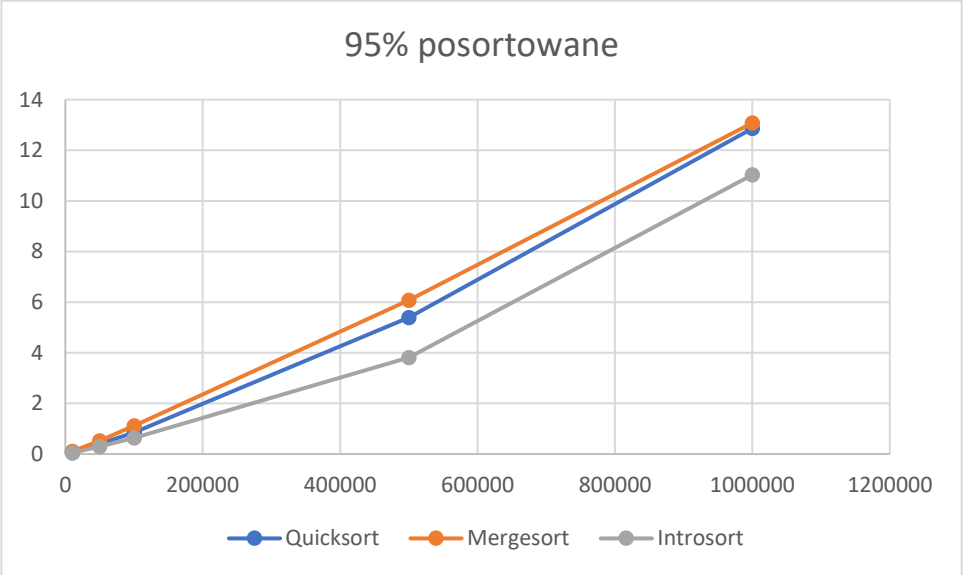
Długośc wiersza	10000	50000	100000	500000	1000000
Quicksort czas[s]	0,132847	0,62318	1,31424	7,58203	16,4305
Mergesort czas[s]	0,14904	0,79167	1,7013	9,40445	19,3412
Introsort czas[s]	0,084567	0,5558	1,16895	6,36613	13,2981

4) 75% pierwszych elementów posortowanych:



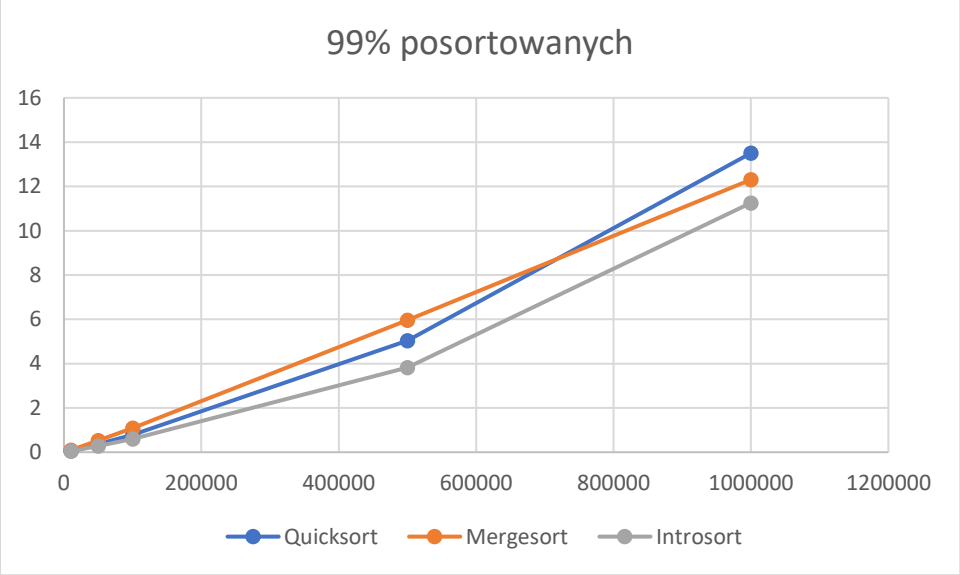
Długość wiersza	10000	50000	100000	500000	1000000
Quicksort czas[s]	0,090845	0,492219	1,02996	5,98649	13,7353
Mergersort czas[s]	0,106622	0,648884	1,33344	7,56467	15,8854
Introsort czas[s]	0,082647	0,393734	0,857557	4,95596	11,5823

5) 95 % pierwszych elementów posortowanych:



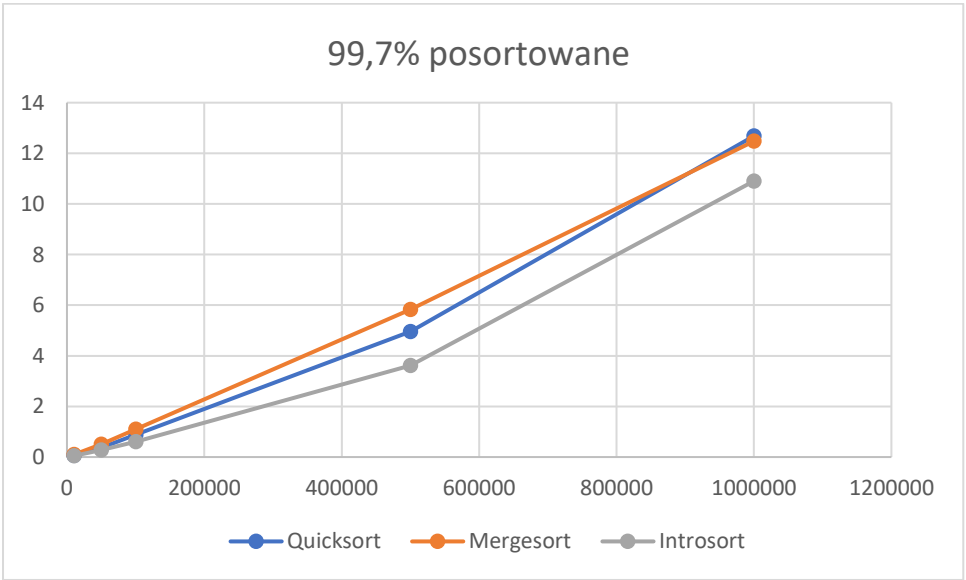
Długość wiersza	10000	50000	100000	500000	1000000
Quicksort czas[s]	0,060866	0,394633	0,844998	5,38836	12,8626
Mergesort czas[s]	0,103317	0,522446	1,11798	6,08029	13,0804
Introsort czas[s]	0,04597	0,29008	0,634727	3,81122	11,0311

6) 99% pierwszych elementów posortowanych:



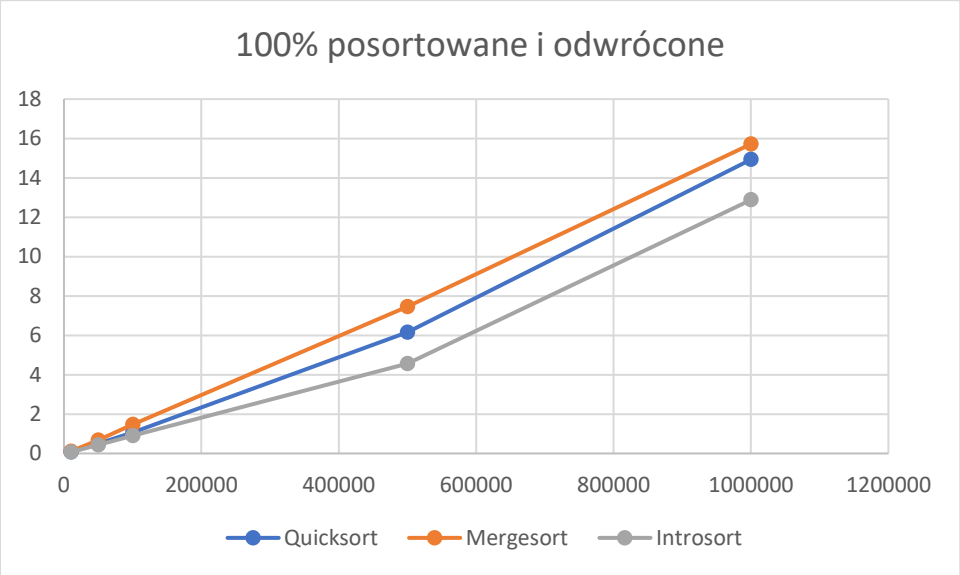
Długość wiersza	10000	50000	100000	500000	1000000
Quicksort czas[s]	0,062231	0,367487	0,792379	5,03611	13,5047
Mergesort czas[s]	0,094944	0,52179	1,09093	5,96812	12,307
Introsort czas[s]	0,042972	0,273834	0,604227	3,81728	11,2581

7) 99,7 % pierwszych elementów posortowanych:



Długość wiersza	10000	50000	100000	500000	1000000
Quicksort czas[s]	0,060548	0,362831	0,885038	4,95742	12,684
Mergesort czas[s]	0,093	0,505778	1,09571	5,83003	12,4792
Introsort czas[s]	0,045598	0,273815	0,606359	3,61589	10,9051

8) Dane posortowane, ale w odwrotnej kolejności:



Długość wiersza	10000	50000	100000	500000	1000000
Quicksort czas[s]	0,076766	0,496806	1,07046	6,15716	14,9299
Mergesort czas[s]	0,108535	0,683763	1,47256	7,46318	15,7145
Introsort czas[s]	0,071977	0,438592	0,897375	4,57088	12,8809

3. Wnioski:

3.1 Podsumowanie i obserwacje:

Ćwiczenie miało na celu przeprowadzenie badań algorytmów sortowań, porównując ich efektywność dla rożnych porcji danych.

Zgodnie z przewidywaniami, najlepszym algorytmem okazał się algorytm sortowania introspektywnego ,a najgorszym algorytm sortowania przez scalanie. Hybrydowa natura Introsort'a oraz jego wielozadaniowość dla różnych przypadków pokazuje ,że w każdym przypadku sortował on najszybciej. Quicksort również dla większości przypadków sortuje szybciej niż Mergesort. Co ciekawe dla większej ilości danych gdy elementy są już praktycznie posortowane np. dla 99% sortowanie przez scalanie jest szybsze od sortowania szybkiego. Wpływ na to może mieć wybór pivota. W implementacji tego Quicksorta pivot był wybierany jako element środkowy tablicy do posortowania.

Gdy wybieramy długość wiersza 1000000 różnice dla poszczególnych algorytmów są bardziej zauważalne, ponieważ są to różnice w sortowaniu kilku sekund.

Powyższe potwierdzają teoretyczne założenia i poprawność zaimplementowanych algorytmów.

4 Literatura:

1. Drozdek A., C++. Algorytmy i struktury danych, Helion
2. Forum Stackoverflow
3. Wikipedia
4. Forum GeeksForGeeks