Projektowanie Algorytmów i Metody Sztucznej Inteligencji

Sprawozdanie z zadania na ocenę bdb (5.0)

Projekt 2

Wtorek 13:15-15:00, Y03-51f

Bartosz Korchut 259292

Data wykonania sprawozdania 17.05.2022

Link do repozytorium: https://github.com/bkorchut/PAMSI

1. Cel projektu

Zadaniem, które należało wykonać, było utworzenie trzech wybranych algorytmów sortowania. Poza tym, trzeba było przefiltrować dane w pliku .csv z informacjami o filmach. Po wykonaniu filtracji i sortowania trzeba było podać informację o czasu sortowania, i średniej wartości i medianie rankingu.

2. Wybrane algorytmy sortowania

W projekcie wykorzystałem następujące algorytmy sortowania:

- Quick Sort
- Merge Sort
- Bucket/Bin Sort

3. Proces filtracji danych

W celu przefiltrowania danych zostały odrzucone wszystkie pozycje, przy których nie było informacji o rankingu filmu. Okazało się, że w zbiorze danych występują powtórzenia nazw filmów wraz z rankingami, jednak nie zostały one usunięte, ze względu na brak takiego polecenia. Po filtracji ilość danych zmniejszyła się do …, co spowodowało brak możliwości wykonania algorytmów sortowania dla 1000000 danych.

4. Sortowanie

4.1 Quick Sort

Sortowanie szybkie jest (losowym) algorytmem sortującym bazującym na technice dziel i zwyciężaj. Wybierany jest losowy element x (nazywany piwotem). Następnie sprawdzane są kolejne elementy, które są dzielone na mniejsze od x, równe x i większe od x. Potem sortowane są elementy mniejsze od x i większe od x. Oczekiwana złożoność obliczeniowa sortowania szybkiego wynosi $O(nlog\ n)$. Jednakże złożoność obliczeniowa jest proporcjonalna do sumy:

$$n + (n-1) + \ldots + 2 + 1,$$

Zatem najgorszy czas tego sortowania to $O(n^2)$.

4.2 Merge Sort

Sortowanie przez scalanie jest algorytmem sortującym bazującym na technice dziel i zwyciężaj. Sortowanie przez scalanie listy zawierającej n elementów składa się z trzech

kroków. Najpierw lista dzielona jest na dwie sekwencje. Następnie jest rekurencyjnie sortowana. Na koniec obie sekwencje są łączone w jedną posortowaną listę. Sortowanie przez scalanie wykorzystuje komparator, posiada złożoność $O(nlog\ n)$. Nie wykorzystuje zewnętrznej kolejki priorytetowej. Pobiera dane w sposób sekwencyjny (odpowiedni do sortowania danych na dysku zewnętrznym).

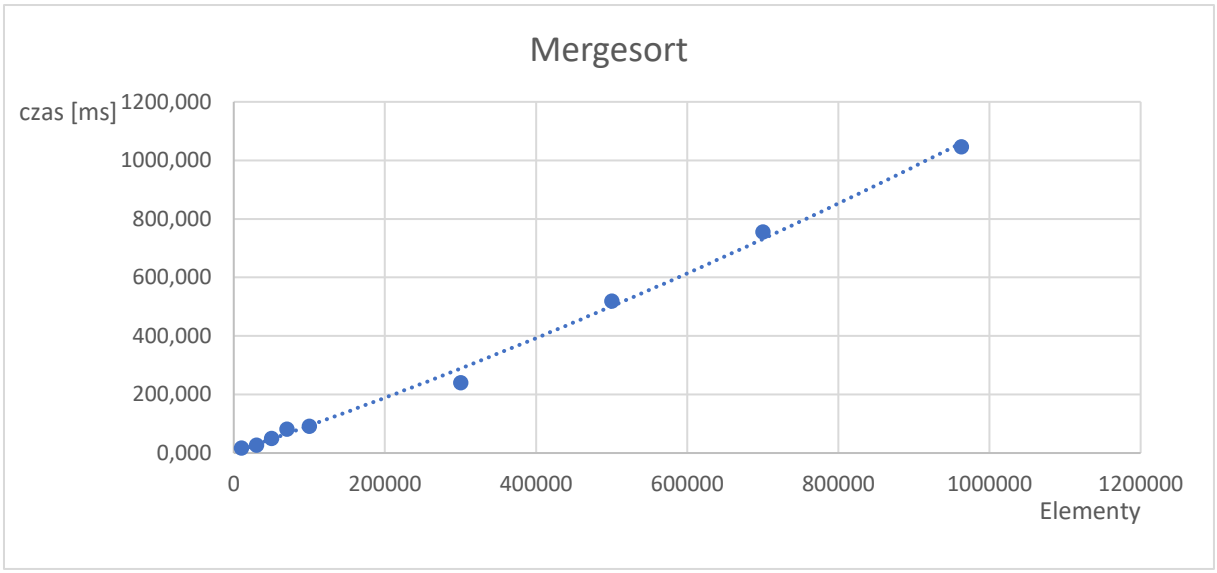
4.3 Bucket/Bin Sort

Dla listy elementów tworzymy sekwencję n wpisów (klucz, wartość) z kluczami w przedziale $(0, N - 1)$. Sortowanie kubełkowe (Bucket/Bin Sort) wykorzystuje klucze jako indeksy w zewnętrznej tablicy sekwencji (kubełki). Najpierw przenosimy wszystkie wpisy z listy do kubłów. Potem Dla $i = 0, \dots, N - 1$, przenosimy wpisy kubła $B(i)$ na koniec sekwencji. Sortowanie Bucket/Bin Sort ma złożoność O(n).

5. Testy algorytmów

Tabela 1 Czas sortowania- merge sort

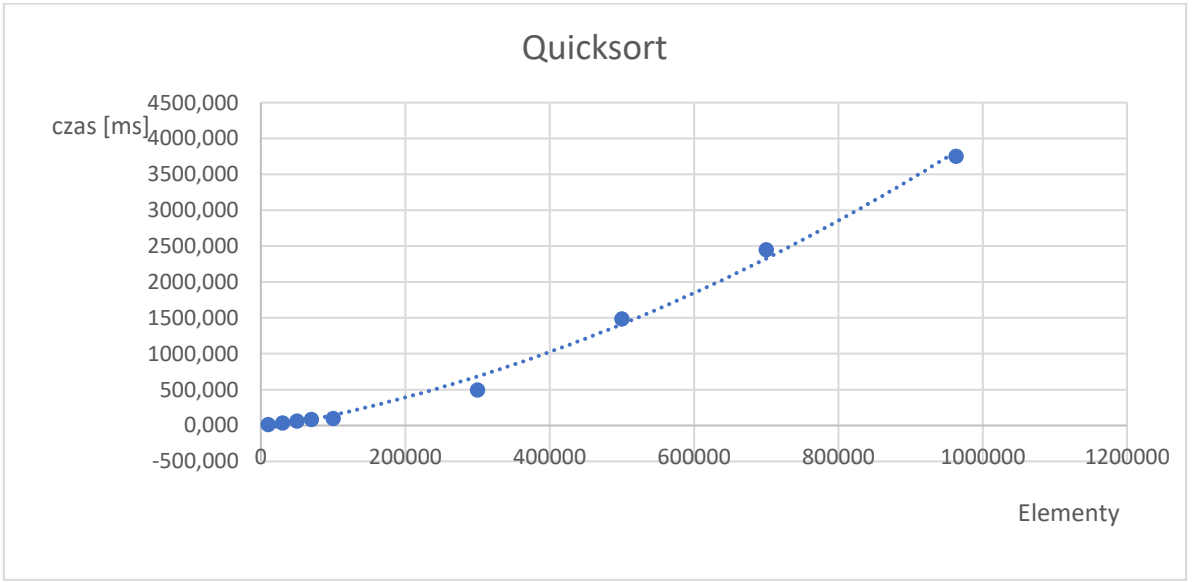
Ilość elementów	czas [ms]			śr. czas [ms]
	Mergesort			
10000	18,606	15,398	16,985	16,996
30000	28,766	24,825	26,706	26,766
50000	51,058	51,566	47,421	50,015
70000	84,812	79,513	80,234	81,520
100000	90,696	89,27	96,07	92,012
300000	237,51	247,592	236,884	240,662
500000	539,764	536,223	481,241	519,076
700000	784,074	767,025	715,404	755,501
962834	1016,182	1111,72	1012,25	1046,717



Wykres 1 Czas sortowania zależny od ilości danych – merge sort

Tabela 2 Czas sortowania- quick sort

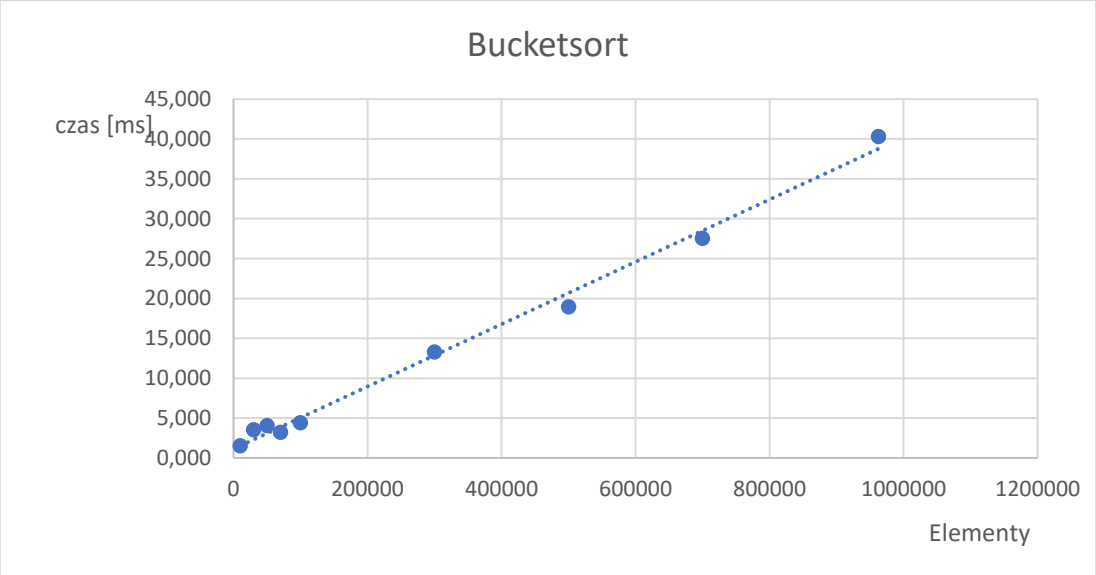
Ilość elementów	czas [ms]			śr. czas [ms]
	Quicksort			
10000	10,721	11,11	12,466	11,432
30000	33,148	31,082	40,809	35,013
50000	59,484	58,326	57,98	58,597
70000	81,175	86,905	81,337	83,139
100000	95,46	99,85	98,124	97,811
300000	496,923	486,736	496,511	493,390
500000	1464,195	1519,395	1470,578	1484,723
700000	2501,34	2431,581	2411,039	2447,987
962834	3858,298	3704,569	3685,778	3749,548



Wykres 2 Czas sortowania zależny od ilości danych – quick sort

Tabela 3 Czas sortowania- bucket sort

Ilość elementów	czas [ms]			śr. czas [ms]
	Bucketsort			
10000	1,474	1,51	1,549	1,511
30000	3,126	4,285	3,092	3,501
50000	3,749	4,124	4,22	4,031
70000	3,802	2,723	3,042	3,189
100000	5,294	3,886	4,042	4,407
300000	14,343	12,613	12,816	13,257
500000	20,148	18,557	18,058	18,921
700000	30,521	25,96	26,08	27,520
962834	41,525	40,261	39,132	40,306



Wykres 3 Czas sortowania zależny od ilości danych – bucket sort

Tabela 4 Średnia wartość i mediana rankingu filmów

Ilość elementów	Średnia	Mediana
10000	5,4603	5
30000	5,4714	5
50000	5,4909	5,5
70000	5,8225	6
100000	6,0902	7
300000	6,5352	7
500000	6,6657	7
700000	6,6655	7
962834	6,6366	7

6. Wnioski

- Ze względu na duże różnice w czasie wykonywania sortowania spowodowane przez szybkość działania programu dla każdego sortowania zostały wykonane 3 pomiary.
- Quick sort nie nadaje się do sortowania już posortowanych danych, ponieważ dla takich jego złożoność wzrasta do $O(n^2)$.
- Merge sort ma stałą wartość obliczeniową przez co zachowuje się stabilnie dla każdej wielkości zbioru $O(n\ logn)$.
- Bucket/Bin sort cechuje bardzo krótki czas sortowania ze względu na liniową złożoność obliczeniową $O(n)$.
- Dla różnej ilości danych możemy zauważyć różnice dla średniej wartości i mediany rankingu filmów.
- Na liście wielokrotnie powtarzają się nazwy filmów i ich oceny co powoduje nieadekwatne rankingi ich wartości średnie i mediany.

7. Literatura

https://www.baeldung.com/java-invoke-static-method-reflection

https://www.baeldung.com/java-merge-sort

https://www.baeldung.com/java-quicksort

https://en.wikipedia.org/wiki/Bucket_sort

https://stackoverflow.com/questions/27254306/algorithms-for-bucket-sort