Politechnika Wrocławska

PAMSI - Projektowanie Algorytmow i Metody Sztucznej inteligencji

Projekt 2

Zadanie na ocene 5.0 - Link do repozytorium

Skład grupy

Marcin Cichocki 259322

Wydział i kierunek studiów

W12N, Automatyka i Robotyka

Kod grupy zajęciowej, termin zajęć

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

Prowadzący

Dr hab. inż. Andrzej Rusiecki

Data wykonania ćwiczenia, termin oddania sprawozdania

16.05.2022, 17.05.2022

autor szablonu:

Jakub Marczak

Spis treści

1 Wprowadzenie

Naszym zadaniem bylo zaimplementowanie trzech algorytmow sortowania i przeprowadzić analizę ich efektywności na podanym zbiorze danych.Ja posrod czterech algorytmow na ocene 5.0 wybralem sortowanie przez scalanie, quick sort oraz sortowanie kubelkowe. W naszym zbiorze danych znajdywaly sie filmy wraz z ocena. Duzo filmow bylo bez oceny dlatego trzeba bylo przeprowadzic filtrowanie zeby dane filmy odrzucic.

1.1 Proba implementacji listy jednokierunkowej - przejscie na tablice dynamiczna

W poprzednim projekcie struktura jaka wykorzystalem byla wlasnie lista jednokierunkowa. Majac juz troche gotowego kodu pomyslalem ze wlasnie i w tym projekcie z tego skorzystam lecz przy probie implementacji sortowania przez scalania dla listy jednokierunkowej uznalem ze bedzie to dla mnie za ciezkie przez trudny dostep do danego elementu listu. Postanowilem wiec ze latwiej bedzie stworzyc jakas klase (w moim przypadku jest to klasa o nazwie: Ranking) w ktorej beda sie znajdowaly zmienne w ktorych bede przechowywal nazwe filmu oraz ocene filmu a potem stworzenie tablicy dynamicznej o typie mojej klasy. Dzieki temu mamy bardzo latwy dostep do kazdego filmu i jego oceny.

```
//Klasa "Ranking" w ktorej znajduje sie
class Ranking
{
public:
    string film_name;
    float film_rate = 0;
```

Rysunek 1: Klasa w ktorej $film\_name$ to nazwa filmu a $film\_rate$ to ocena filmu

```
tab[turn].film_name= name;
tab[turn].film_rate = rate;
```

Rysunek 2: Tak uzyskujemy dostep do danego filmu i jego oceny(turn to miejsce w kolejce, dla turn=0 bedzie to pierwszy film w rankingu)

1.2 Implementacja sortowan

Przy implementacji sortowan nie mialem az tak wielu problemow poniewaz bylo duzo pomocy w internecie jesli chodzilo o sam kod. Strony ktorymi sie posilkowalem znajduja sie oczywiscie w bibliografii. Sama zasada dziala kazdego z sortowan nie byla trudna, moge nawet stwierdzcic ze nawet byly latwe w zrozumieniu zasady dzialania kazdego z sortowan ale sama implementacja przysporzyla mi troche problemow dlatego duzo czytalem jak zaimplementowac dany algorytm sortowania.

1.3 Proba zaimplementowania wiekszej losowosci

Naszym zadaniem bylo posortowanie danych filmow w ilosci 10k,100k,500k oraz maksymalna liczbe filmow (w poleceniu tez bylo 1m filmow ale po przefiltrowaniu bylo ich troche mniej niz 1m). Moja funkcja do wczytywania filmow z pliku.txt wczytywala filmy po kolei, linia po linii. Za kazdym razem dla sortowania dla danej ilosci filmow nasz program sortowal te same filmy. Chcialem zrobic tak zeby za kazdym razem nasz program sortowal dziesiec tysiecy filmow ale zeby to byly rozne dziesiec tysiecy filmow. Postanowilem wiec stworzyc sobie wektor o zmiennych typu int od 1 do maksymalnej liczby filmow. Nastepnie przemieszac ten wektor zeby liczby znajdywaly sie w nim w losowej kolejnosci a nastepnie jak bede dodawal filmy do naszej tablicy to beda sie dodawac na podstawie wartosci naszego wektora(np. pierwszy film ktory sie doda to bedzie film z linii pliku 401202 a nie z linii pierwszej jak w przypadku wczytywania po kolei).

```
vector<int> arr;

//Tworze vector liczb od 1 do 962903 (od 1 do maksymalnej liczby filmow poniewaz filmow w pliku jest 962903)
for (int j = 1; j <= 962903; ++j)
{
    arr.push_back(j);
}
//Mieszam vector zeby liczby w nim byly w losowej kolejnosci
random_shuffle(arr.begin(), arr.end());



if (!MyReadFile.is_open())
{
    cout << "Nie otwarto pliku";
}
else
{
    for (int i = 0; i < sizee; i++)
    {
        //Wczytuje film z lini o numerze arr[1], czyli np. moj pierwszy film w rankingu bedzie 423901 filmem z pliku tekstowego
        for (int which_line=1; which_line <= arr[i]; which_line++)
        {
            getline(MyReadFile, tmp);
        }
        MyReadFile.close(); //Zamykam plik po to zeby przy nastepnej petli znowu czytalo plik od poczatku.
        MyReadFile.open("projekt_dane.txt"); //Otwieram plik i przy nastepnej petli znowu bedziemy czytac plik od 1 lini.
```

Rysunek 3: vector arr o wartosciach typu int, dodajemy do niego wartosci od 1-962903 a nastepnie mieszamy. Nastepnie przy wczytywaniu filmow z pliku tekstowego nasz program omija tyle linijek ile wynosi wartosc arr[0] i jesli jest rowny to zapisuje ta linie do zmiennej "tmp".

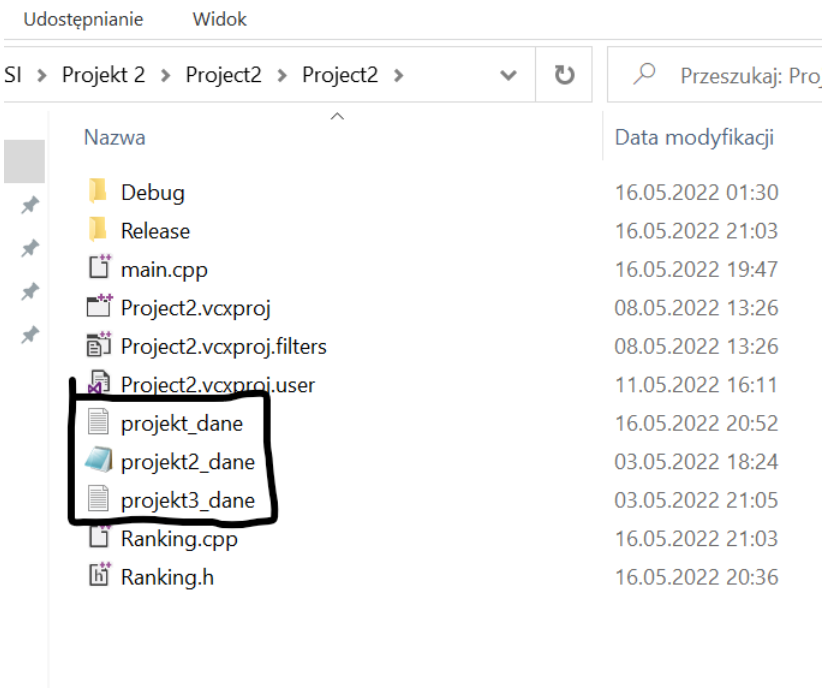
Tutaj pojawil sie problem poniewaz w moim rozwiazaniu program dzialal ale nie bylo to optymalne rozwiazanie poniewaz przy probie wczytania dziesieciu tysiecy filmow moj program wpisywal je do tablicy okolo 10 minut. Stalo sie tak dlatego poniewaz wartosci w wektorze byly ustawione losowo. Na przyklad pierwsze trzy wartosci wektora byly bliskie 900k. Wtedy nasz program zeby wpisac 3 filmy do tablicy musial przeczytac nasz caly plik trzy razy. Nie moglem znalezc rozwiazania w ktorym program wczytywalby podana linijke, jedyne co wymyslilem to omijanie tylu linii zeby nasza dana linijke wpisac do tablicy. Przy probie wpisania dziesieciu tysiecy filmow filmow moj program wpisywal filmy 15 minut wiec jakbym sprobowal wpisac pol miliona filmow to prawdopodobnie moglbym wziac urlop dziekanski a program nadal wpisywalby filmy do tablicy. Uznalem wiec ze aktualnie jest to zbedne i usunalem caly kod i wrocilem do poprzedniego wczytywania po kolei.

2 Wizualizacja dzialania programu - Link do filmu na YT

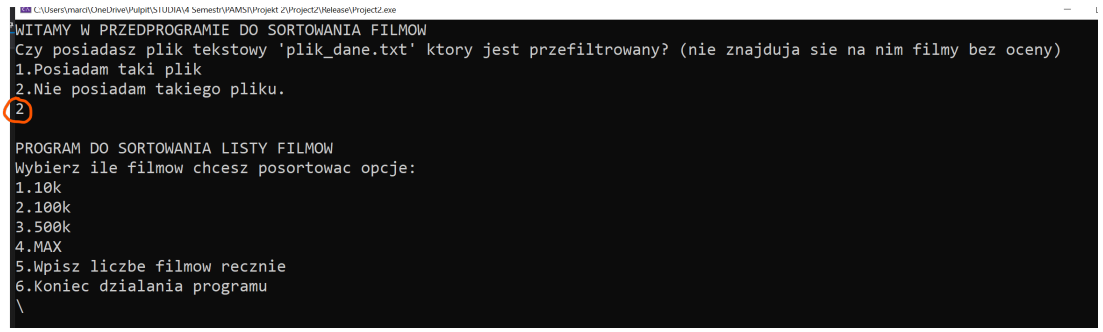
Powyzej odnosnik "Link do filmy na YT" ktory przekieruje na platforme youtube.com do filmiku na ktorym pokazana jest zasada dzialania programu.

2.1 Warunek zadziala programu

Nasz program czyta filmy z pliku *projekt_dane.txt*. Plik tworzy sie jesli w danym folderze z plikami zrodlowymi znajduje sie plik ktory byl wstawiony na eportal. Podczas uruchomienia programu pierwsza opcja bedzie do wybrania wlasnie opcja czy mamy plik *projekt_dane.txt*. Jesli nie to program stworzy nam plik w ktorym beda przefiltrowane filmy. Na githuba nie moglem wstawic danych plikow tekstowych poniewaz wazyly one za duzo.



Rysunek 4: Miejsce w ktorym musza znajdowac sie dane pliki tekstowe



Rysunek 5: Wybranie opcji zeby stworzyc plik *projekt_dane.txt* z przefiltrowanymi filmami

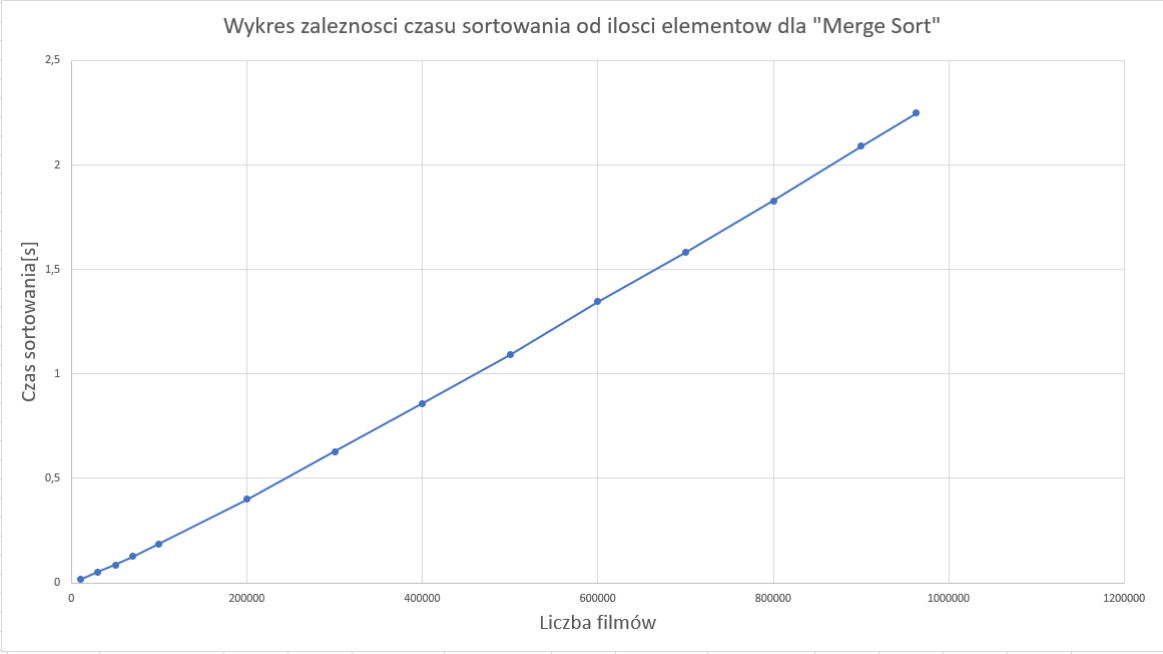
3 Testy efektywnosci dla algorytmu ”Merge sort”

3.1 Tabela z wynikami czasow sortowan

	Merge sort
Liczba filmow	Czas sortowania[s]
10000	0,016
30000	0,052
50000	0,086
70000	0,125
100000	0,186
200000	0,401
300000	0,628
400000	0,859
500000	1,092
600000	1,347
700000	1,582
800000	1,828
900000	2,09
962903	2,249

Rysunek 6: Tabela z czasami sortowan dla algorytmu ”Merg Sort”

3.2 Wykres przedstawiajacy czasy sortowan



Rysunek 7: Wykres przedstawiajacy czasy sortowan dla algorytmu ”Merg Sort”

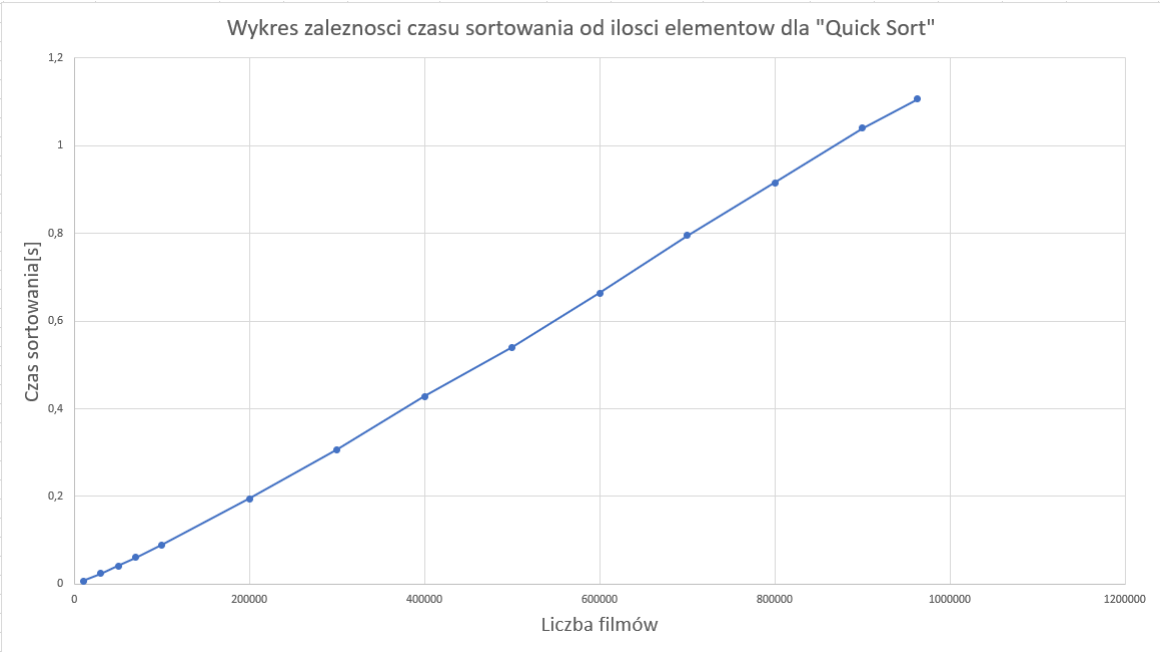
4 Testy efektywnosci dla algorytmu "Quick sort"

4.1 Tabelka z wynikami czasow sortowan

	Quick sort
Liczba filmow	Czas sortowania
10000	0,007
30000	0,024
50000	0,041
70000	0,061
100000	0,09
200000	0,195
300000	0,307
400000	0,428
500000	0,54
600000	0,664
700000	0,795
800000	0,915
900000	1,04
962903	1,106

Rysunek 8: Tabela z czasami sortowan dla algorytmu "Quick Sort"

4.2 Wykres przedstawiajacy czasy sortowan



Rysunek 9: Wykres przedstawiajacy czasy sortowan dla algorytmu "Quick Sort"

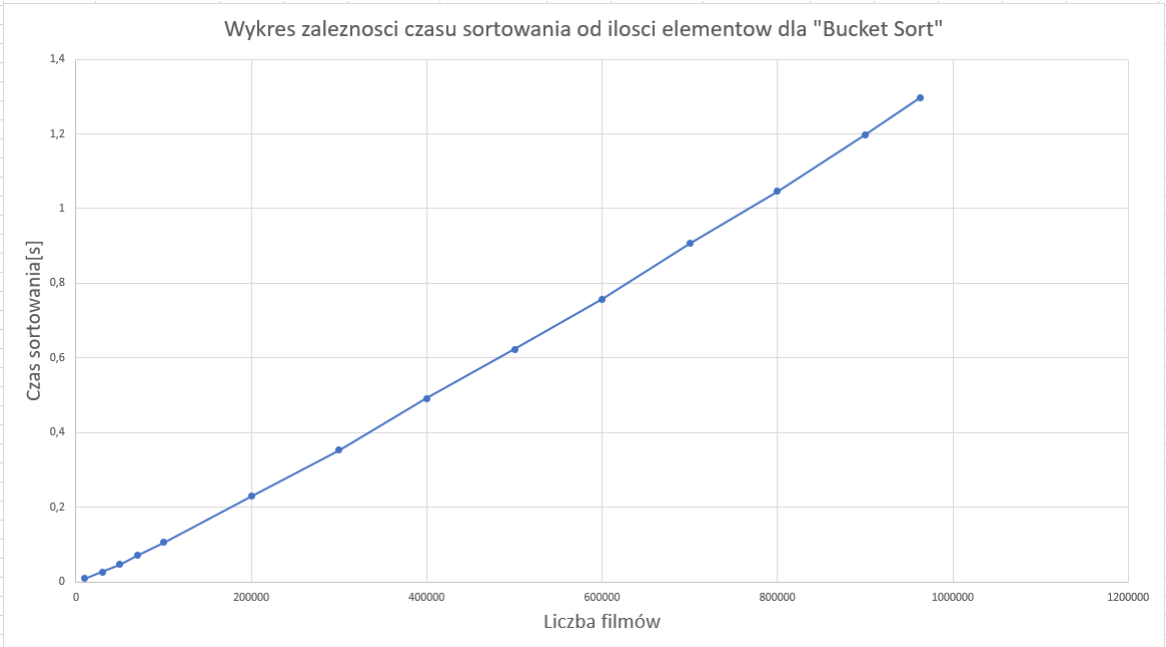
5 Testy efektywnosci dla algorytmu "Bucket sort"

5.1 Tabela z wynikami czasow sortowan

	Bucket sort
Liczba filmow	Czas sortowania
10000	0,009
30000	0,026
50000	0,046
70000	0,07
100000	0,106
200000	0,229
300000	0,353
400000	0,491
500000	0,623
600000	0,758
700000	0,906
800000	1,046
900000	1,198
962903	1,298

Rysunek 10: Tabela z czasami sortowan dla algorytmu "Bucket Sort"

5.2 Wykres przedstawiajacy czasy sortowan



Rysunek 11: Wykres przedstawiajacy czasy sortowan dla algorytmu "Bucket Sort"

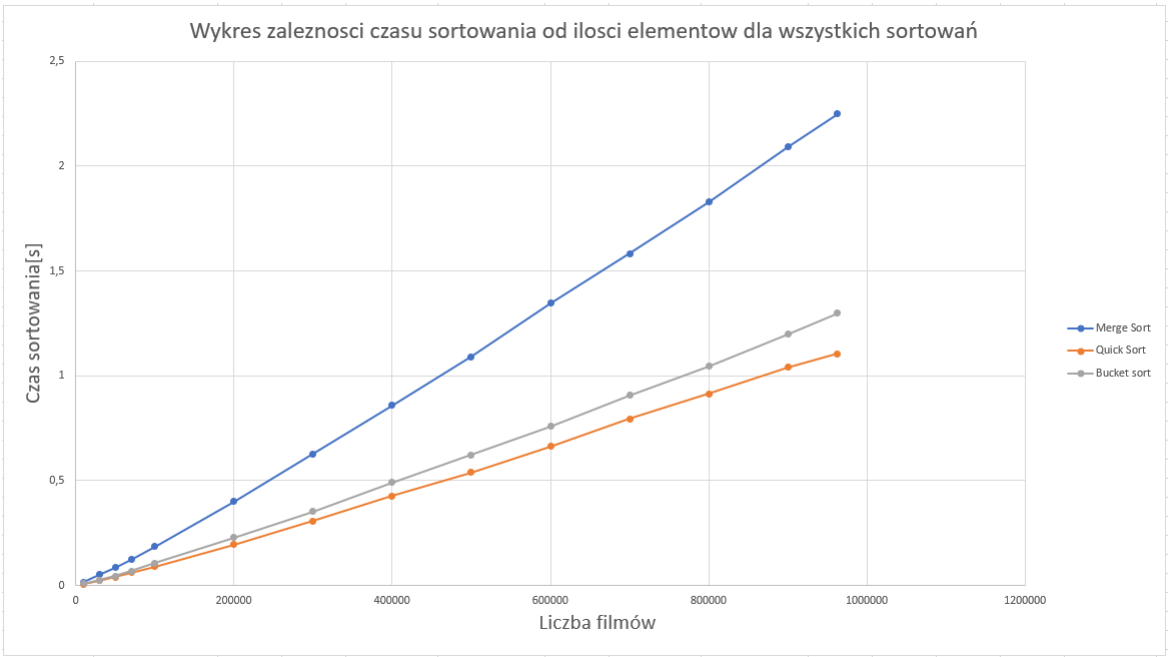
Testy efektywnosci dla wszystkich sortowan

5.3 Zbiorcza Tabelaa z wynikami czasow sortowan

	Merge sort	Quick sort	Bucket sort
Liczba filmow	Czas sortowania[s]	Czas sortowania	Czas sortowania
10000	0,016	0,007	0,009
30000	0,052	0,024	0,026
50000	0,086	0,041	0,046
70000	0,125	0,061	0,07
100000	0,186	0,09	0,106
200000	0,401	0,195	0,229
300000	0,628	0,307	0,353
400000	0,859	0,428	0,491
500000	1,092	0,54	0,623
600000	1,347	0,664	0,758
700000	1,582	0,795	0,906
800000	1,828	0,915	1,046
900000	2,09	1,04	1,198
962903	2,249	1,106	1,298

Rysunek 12: Tabela z czasami sortowan dla wszystkich algorytmow

5.4 Wykres przedstawiajacy czasy sortowan dla wszystkich algorytmow



Rysunek 13: Wykres przedstawiajacy czasy sortowan dla wszystkich algorytmow

6 Wnioski

- Kazde z sortowan dziala w sposob prawidlowy. Po kazdym posortowaniu wywolywana jest funkcja "check" ktora sprawdza czy sortowanie sie udalo. Brak informacji w terminalu oznacza ze tablica zostala posortowana prawidlowo. W kazdym przypadku jest brak informacji zwrotnej. Oznacza to ze sortowania dzialaja.
- Najszybszym sortowaniem okazalo sie sortowanie algorytmem "Quick Sort" a najwolniejszym "Merge Sort". Z ogolno dostepnej wiedzy wiemy ze "Quick sort" dziala szybciej niz Merge sort. Moze byc to spowodowane tym ze "Merge sort" wykorzystuje dodatkowa pamiec do tworzenia podtablic.
- Wartosci median i srednich wartosci dla kazdego algorytmu sortowania sa takie same. Przy zle posortowanych srednie wartosci tak czy siak wynosilyby tyle samo ale wartosci median sie zgadzaja.

	Merg sort	
Liczba filmow	Srednia wartosc	Mediana
10000	5,4603	5
100000	6,08993	6
500000	6,66572	7
max	6,63661	7

	Quick sort	
Liczba filmow	Srednia wartosc	Mediana
10000	5,4603	5
100000	6,08993	6
500000	6,66572	7
max	6,63661	7

	Bucket sort	
Liczba filmow	Srednia wartosc	Mediana
10000	5,4603	5
100000	6,08993	6
500000	6,66572	7
max	6,63661	7

Rysunek 14: Tabela z wartosciami median i srednich wartosci dla kazdego algorytmu sortowania

Literatura

[1] Wikipedia Merg sort
https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie

[2] Wikipedia Quick sort
https://en.wikipedia.org/wiki/Quicksort

[3] Wikipedia Bucket sort
https://en.wikipedia.org/wiki/Bucket_sort

[4] Merge sort by GeeksforGeeks
https://www.geeksforgeeks.org/merge-sort/

[5] Quick sort by GeeksforGeeks
https://www.geeksforgeeks.org/cpp-program-for-quicksort/

[6] Bucket sort by GeeksforGeeks
https://www.geeksforgeeks.org/bucket-sort-2/

[7] Quick sort - sortowanie szybkkie
https://www.algorytm.edu.pl/algorytmy-maturalne/quick-sort.html