Politechnika Wrocławska

Mikroprojekt 2

Projektowanie i analiza algorytmów

Wydział Informatyki i Telekomunikacji

Informatyczne Systemy Automatyki

Aleksander Żołnowski, 272536

Spis treści

1 Wstęp

Zadanie polegało na zaimplementowania trzech algorytmów sortowania i przebadaniu ich efektywności. W projekcie zostały wybrane następujące algorytmy:

- Sortowanie przez scalenie
- Sortowanie szybkie
- Sortowanie kubełkowe

Efektywność algorytmów należało przebadać na zbiorze danych zawartym w pliku „projekt2_dane.csv". Plik ten to okrojona baza filmów z „IMDb Largest Review Dataset" z kaggle.com, zawierająca jedynie ranking i tytuł. Sortowanie wykonano według rankingu.

2 Struktura danych

W celu przechowania zbioru danych z pliku, zdecydowano się na wykorzystanie struktury danych wektora ze względu na jego elastyczność i efektywność. Wektor służył jako dynamiczna tablica na struktury przechowujące informacje o poszczególnych filmach, takie jak: indeks, tytuł i ranking.

```
struct Movie
{
    int index;
    string title;
    float rank;

};

class Vector
{
private:
    vector<Movie> MovieVector;
```

Rysunek 1 Struktura danych na zbiór danych z pliku

W celu umożliwienia badań wydajności algorytmów zdecydowano się na metodę, która uzupełnia nasz wektor danymi z pliku jednocześnie odrzucając ewentualne błędne lub puste wpisy. Przypuszczana złożoność obliczeniowa O(n).

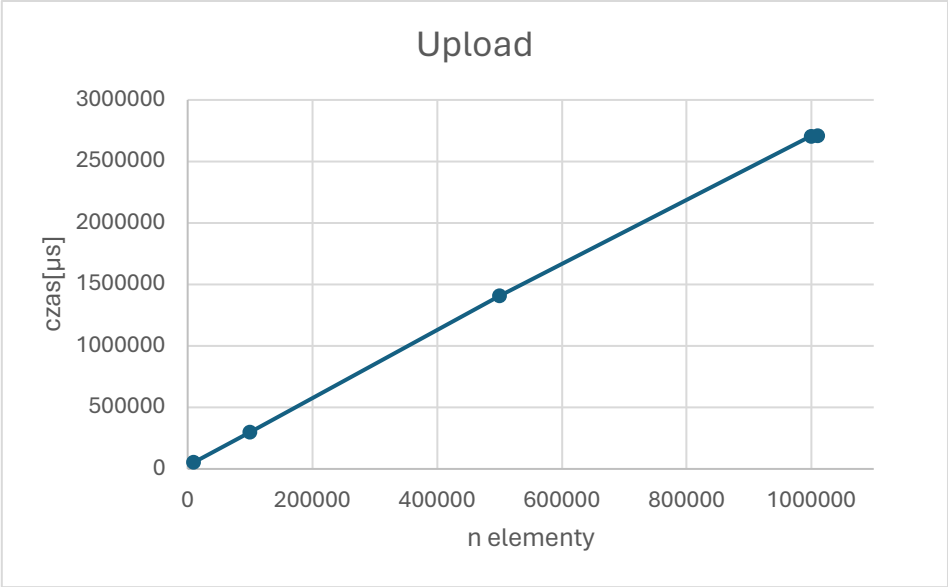
```
while (getline(File, line) && MovieVector.size() < size)
{
    if (firstLine == true)
    {
        firstLine = false;
        continue;
    }

    stringstream ss(line);
    string index_str, movie, rank_str;

    getline(ss, index_str, ',');
    getline(ss, movie, ',');
    while (movie.front() == '"')
    {
        string temp;
        if (movie.back() == '"')
        {
            break;
        }
        getline(ss, temp, ',');
        movie = movie + "," + temp;
    }

    getline(ss, rank_str, ',');

    if (rank_str.empty())
    {
        continue;
    }
```

Rysunek 2 Fragment metody odpowiadającej za wgranie danych

Zbadano czas i złożoność obliczeniowa tej metody

Upload	
n	t[µs]
10000	53569
100000	299413
500000	1407759
1000000	2704774
1010292	2708982

Tabela 1 Wyniki operacji upload



Rysunek 3 Charakterystyka złożoności obliczeniowej operacji upload

Badanie potwierdziło przypuszczaną złożoność obliczeniowa operacji Upload, która wynosi O(n).

3 Algorytmy

3.1 Sortowanie przez scalenie (ang. merge sort)

Algorytm merge sort polega na rekurencyjnym dzieleniu tablicy na dwie części, aż do uzyskania tablic jednoelementowych. Następnie poszczególne podtablice są scalane w sposób, który zapewnia posortowanie. Algorytm ten działa według metody "dziel i zwyciężaj", która polega na podzieleniu problemu na mniejsze, łatwiejsze do rozwiązania.

Złożoność obliczeniowa dla przypadku średniego i pesymistycznego jest taka sama, wynosi $O(nlogn)$.

3.2 Sortowanie szybkie (ang. quicksort)

Algorytm quicksort polega na wyborze tzw. elementu osiowego (pivot), a następnie podziale tablicy na dwie części: jedną zawierającą elementy mniejsze od pivota, a drugą zawierającą elementy większe od pivota. Proces ten jest rekurencyjnie powtarzany dla obu części tablicy, aż do uzyskania tablic jednoelementowych. Właściwe sortowanie jest tutaj jakby ukryte w procesie przygotowania do sortowania.

Złożoność obliczeniowa:

- Średnia: $O(nlogn)$ – odpowiedni wybór pivota, równomierny rozkład danych.
- Pesymistyczna $O(n^2)$ - gdy pivot jest zawsze wybierany jako największy lub najmniejszy element, co prowadzi do nierównomiernych podziałów.

3.3 Sortowanie kubełkowe (ang. bucket sort)

Bucket Sort to algorytm sortowania, który dzieli zbiór elementów na kubełki, sortuje każdy kubełek, a następnie łączy posortowane kubełki w celu uzyskania pełnej, posortowanej tablicy.

Złożoność obliczeniowa:

- Średnia $O(n+k)$ – gdy dane są rozłożone równomiernie (n-liczba elementów, k – liczba kubełków).
- Pesymistyczna $O(n^2)$ – gdy dane zostaną rozłożone nierównomiernie np. wszystkie dane wylądują w jednym kubełku.

4 Badania

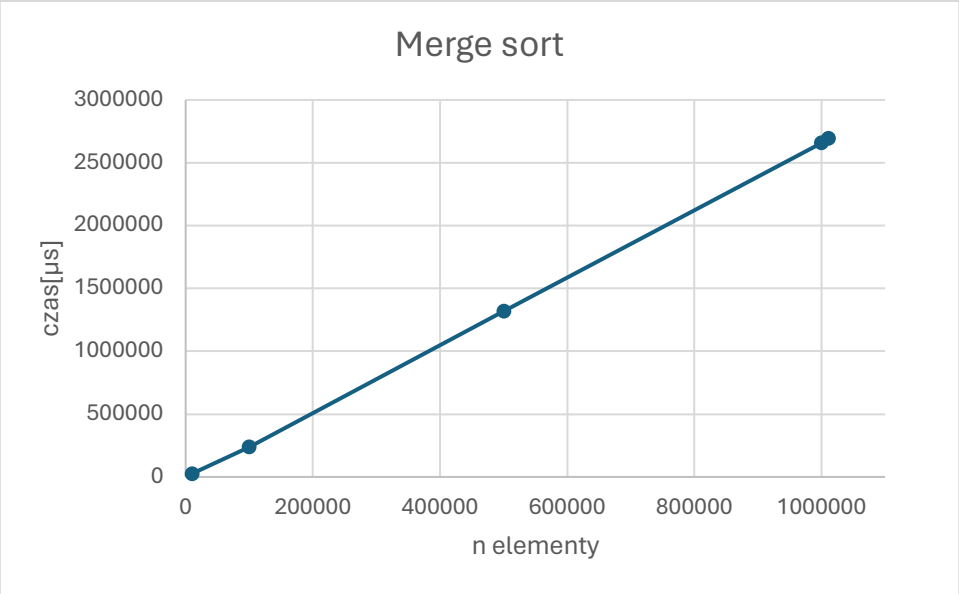
W celu wyznaczenia wydajności badanych algorytmów, przeprowadzano dla każdego z nich serie testów operacji sortowania. Badania zostały przeprowadzone na następujących ilościach n elementów: 10000, 100000, 500000, 1000000 oraz maksymalnej ilości, czyli 1010292. Wydajność została zmierzona w mikrosekundach(µs).

4.1 Sortowanie przez scalenie (ang. merge sort)

Badanie wydajności dla sortowania przez scalenie.

Merge sort	
n	t[µs]
10000	25944
100000	240015
500000	1319604
1000000	2658386
1010292	2694675

Tabela 2 Wyniki sortownia merge sort



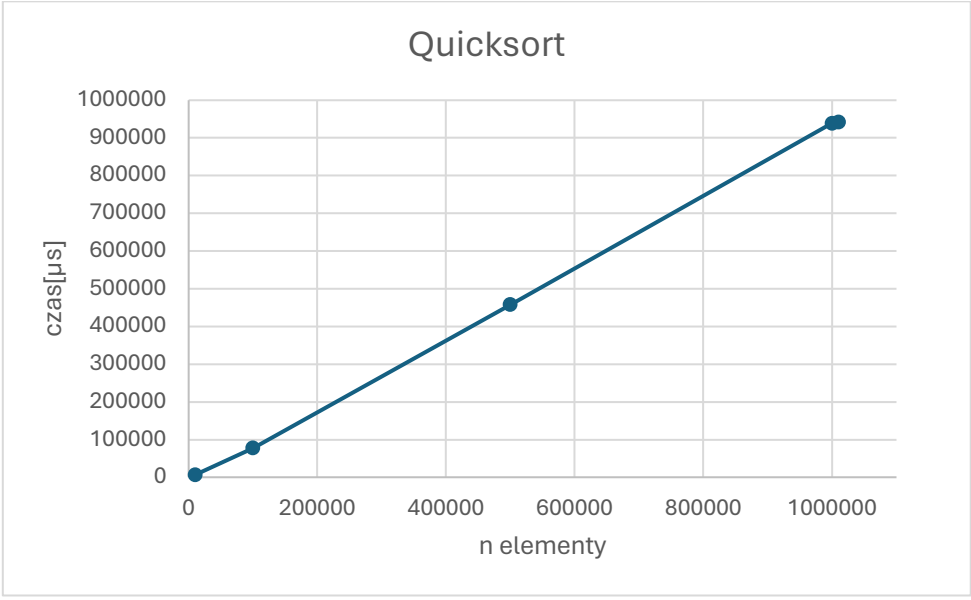
Rysunek 4 Charakterystyka złożoności obliczeniowej sortowania merge sort

4.2 Sortowanie szybkie (ang. quicksort)

Badanie wydajności sortowania szybkiego, quicksort.

Quicksort	
n	t[µs]
10000	6431
100000	77663
500000	457564
1000000	939650
1010292	942621

Tabela 3 Wyniki sortownia quicksort



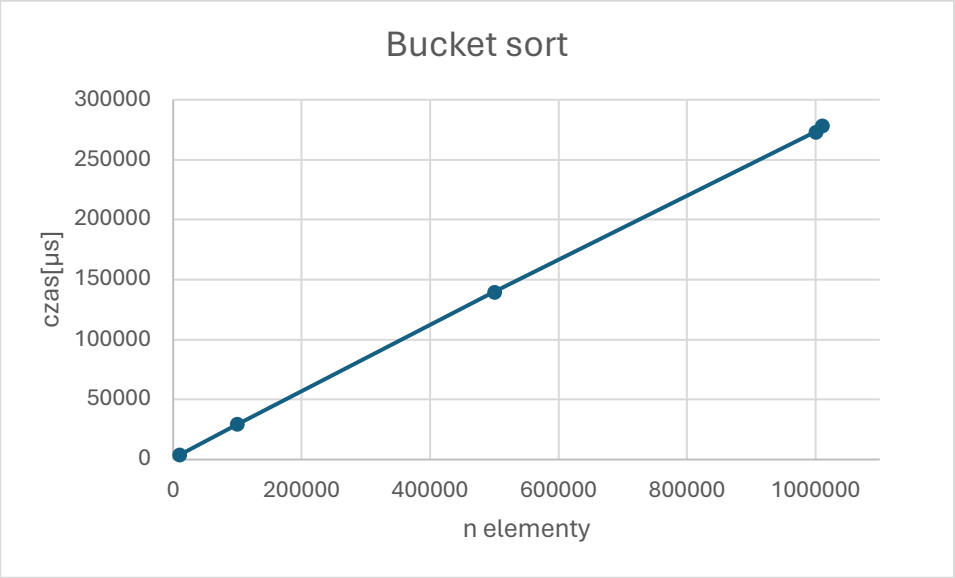
Rysunek 5 Charakterystyka złożoności obliczeniowej sortowania quicksort

4.3 Sortowanie kubełkowe (ang. bucket sort)

Badanie wydajności dla sortowania kubełkowego.

Bucket sort	
n	t[µs]
10000	3849
100000	29567
500000	139696
1000000	273248
1010292	278122

Tabela 4 Wyniki sortowania bucket sort



Rysunek 6 Charakterystyka złożoności obliczeniowej sortowania bucket sort

5 Analiza

5.1 Złożoność obliczeniowa

Następnie przeanalizowano złożoność obliczeniowa otrzymanych wyników zgodnie z notacja dużego O.

Sortowanie przez scalenie, merge sort, ma złożoność obliczeniowa O(n log(n)), ponieważ każde scalenie posortowanej podtablicy, uzyskanej w rekurencyjnym dzieleniu jej na mniejsze, to koszt O(n). W związku z tym, że strukturę wyjściową dzieliliśmy za każdym razem na poł, takich scaleń mamy log(n)

Sortowanie szybkie, quicksort, uzyskało złożoność O(nlog(n)), dzięki odpowiedniemu wyborowi pivota. Wybierany był on za każdym razem na środku sortowanej podtablicy.

Sortowanie kubełkowe, bucket sort, uzyskało złożoność O(n+k), dzięki możliwości równomiernego rozkładu danych na odpowiadające im kubełki.

Merge sort	O(n log(n))
Quicksort	O(n log(n))
Bucket sort	O(n + k)

Tabela 5 Złożoności obliczeniowe algorytmów sortowania

5.2 Wyniki

Kolejnym krokiem było podanie czasów sortowań, mediany i wartości średniej dla każdego z przebadanych zestawów danych.

n	quicksort t[µs]	mergesort t[µs]	bucketsort t[µs]	mediana	średnia
10000	6431	25944	3849	5	5,4603
100000	77663	240015	29567	7	6,08993
500000	457564	1319604	139696	7	6,66572
1000000	939650	2658386	273248	7	6,63661
1010292	942621	2694675	278122	7	6,63661

Tabela 6 Wyniki badań

Za wyznaczenie mediany i wartości średniej odpowiadały odpowiednie metody.

6 Wnioski

- Wszystkie algorytmy osiągnęły średnią złożoność obliczeniowa zgodnie z założeniem.
- Merge sort i quicksort mają złożoność O(n log(n)), co sprawia, że są efektywne dla większych zbiorów danych. W kontekście stabilności, merge sort jest algorytmem stabilnym, natomiast quicksort nie.
- Bucket sort ma złożoność O(n + k) co sprawia, że jest wydajny szczególnie wtedy, gdy dane można równomiernie rozłożyć na kubełki, tak jak w przypadku naszego zbioru danych.
- Na przebadanym zbiorze danych, jeśli chodzi o wyniki czasowe, najlepiej wypadł alogrytm sortowania kubełkowego, drugi był quicksort, natomiast najgorzej sprawdził się algorytm sortowania przez scalenie.
- Wybór algorytmu sortowania powinien być dostosowany do specyfiki danych, ich rozmiaru oraz oczekiwanej wydajności. W przypadku danych o możliwie równomiernym rozkładzie, algorytmy takie jak bucket sort mogą być korzystne, podczas gdy dla ogólnych zbiorów danych,

quicksort może być dobrym wyborem. W przypadku bardzo dużych zbiorów danych, o których mamy mało informacji najlepiej zdecydować się na stabilny algorytm taki jak merge sort.

7 Bibliografia

- Data Structures and Algorithms in C++, 2nd Edition Michael T. Goodrich, Roberto Tamassia, David M. Mount
- Cormen T., Leiserson C.E., Rivest R.L., Stein C., Wprowadzenie do algorytmów, WNT
- https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie
- https://www.geeksforgeeks.org/bucket-sort-2/
- https://pl.wikipedia.org/wiki/Sortowanie_szybkie