Michał Odrobina

*Algorytmy i struktury danych*

*Projekt II, grupa 6*

# Spis treści

# 1. Wstęp wraz z opisem problemu

Celem projektu jest analiza dwóch wybranych algorytmów sortujących wraz z ich implementacją w formie programu w języku C++. Analiza obejmie sortowanie metodą „sortowanie przez wybieranie” oraz sortowanie metodą „quicksort”. Pod uwagę będą brane złożoność obliczeniowa, a także złożoność czasowa obu algorytmów. Testy tych algorytmów będą przeprowadzane na rosnących próbkach danych. Zostanie także uwzględniony rodzaj danych wejściowych, dla wylosowanego ciągu oczekiwanego, optymistycznego, a także pesymistycznego.

# 2. Opis podstaw teoretycznych zagadnienia

W następujących podrozdziałach zostanie omówiony sposób działania każdego z algorytmów.

## 2.1. Quicksort

Algorytm Quicksort opiera się o metodę „Dziel i zwyciężaj”, którą możemy podzielić w następujący sposób:

a) Dziel – większy problem zostaje podzielony na mniejsze podproblemy,
b) Zwyciężaj – odnajdujemy sposób w jaki może zostać rozwiązany problem,
c) Połącz – wszystkie rozwiązane podproblemy zostają połączone i ostatecznie rozwiązują główny problem.

W przypadku algorytmu Quicksort wygląda to następująco:

a) Dziel – najpierw dzielimy ciąg na dwie osobne części, lewa strona musi być mniejsza lub równa od prawej strony,
b) Zwyciężaj – każdą podzieloną część sortujemy algorytmem w sposób rekurencyjny,
c) Połącz – po połączeniu wszystkich części uzyskujemy posortowany ciąg.

W przypadku oczekiwanym jest najszybszym algorytmem sortującym o klasie złożoności $O(n \log n)$. Natomiast gdy mamy do czynienia z przypadkiem pesymistycznym może on spaść nawet do klasy o złożoności obliczeniowej równej $O(n^2)$.

### 2.1.1. Sposób działania

W celu podziału ciągu na części należy wyznaczyć element, który będzie, tzw. piwotem. Po lewej stronie zostaną ułożone elementy mniejsze od piwotu, natomiast po prawej większe. Elementy równe mogą występować w obu częściach, ponieważ nie mają wpływu na proces sortowania. W naszym algorytmie elementem, który zostanie wybrany jako piwot, będzie element środkowy, następnie zostaje on zamieniony z ostatnim elementem. Zostają także wybrane dwa wskaźniki, które umieszczane są na początku ciągu. Wskaźnik pierwszy przeszukuje wszystkie elementy w celu znalezienie tego, który będzie mniejszy od piwota. Drugi służy do zamiany miejscami z elementem pierwszym i przesuwa się o jedno miejsce dalej w ciągu. Proces ten odbywa się do momentu, w którym pierwszy wskaźnik nie będzie odnajdywał elementu mniejszego od piwotu. Na koniec piwot jest zamieniany miejscami z drugim elementem. W ten sposób piwot dzieli ciąg na dwie części, dla których procedura się powtarza (rekurencja), aż do momentu w którym element będzie równy piwotowi. W ostateczności uzyskujemy posortowany ciąg.

## 2.2. Sortowanie przez wybieranie

Sortowanie przez wybieranie jest w swoim działaniu bardzo proste. Chcąc posortować ciąg w kolejności rosnącej, należy najpierw znaleźć najmniejszy element, który zostanie umieszczony na początku ciągu. Identyczne działanie zostaje wykonane na coraz to większych elementach, które są umieszczane po kolei od lewej strony, aż do samego końca. Do momentu, w którym wszystkie elementy zostaną posortowane. Klasa złożoności obliczeniowej jest równa $O(n^2)$.

# 3. Opis szczegółów implementacji problemu

## 3.1. Quicksort

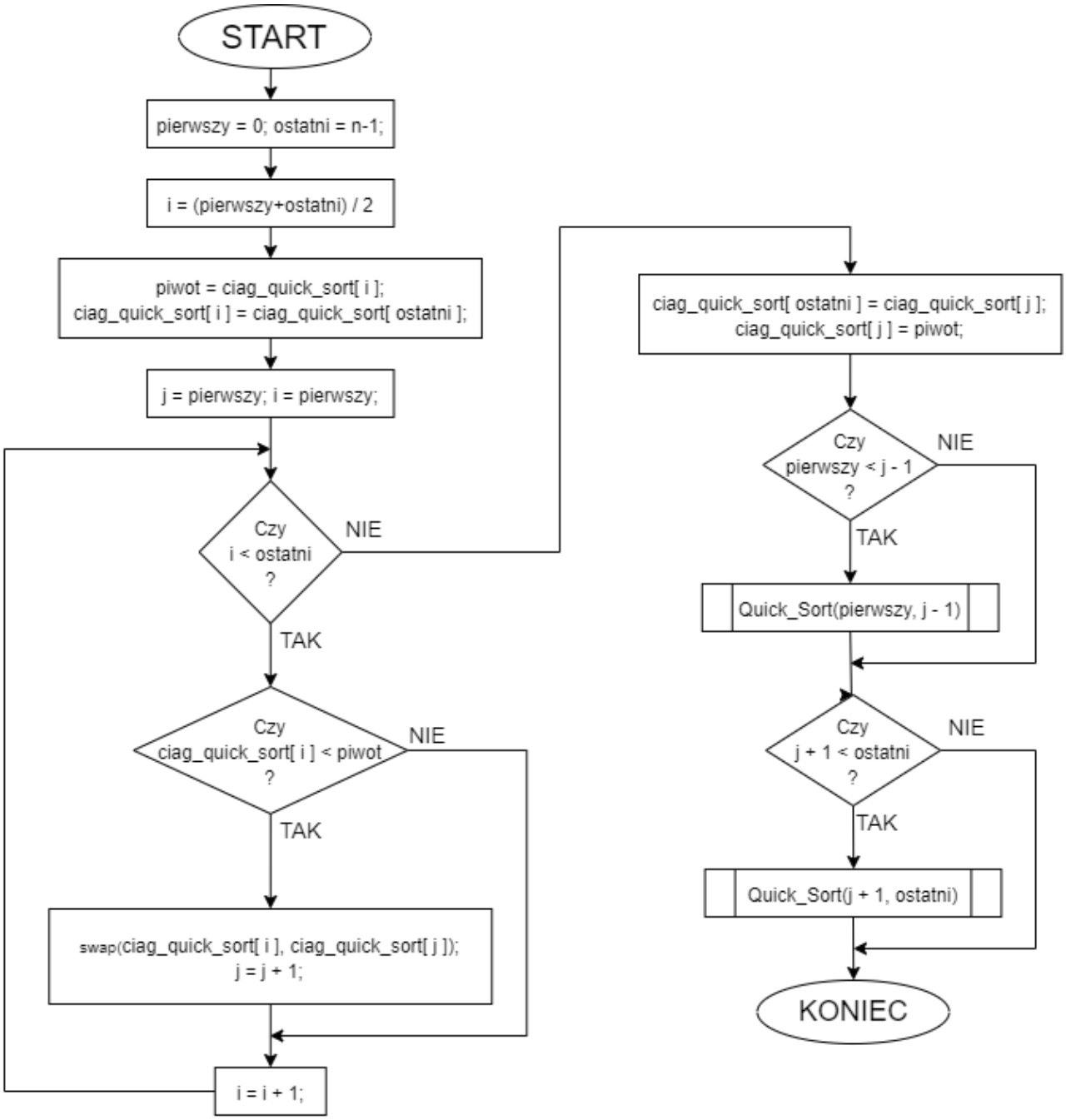
Podczas implementacji algorytmu należało wziąć pod uwagę zbiór jaki będzie miał sortować dany algorytm. W tym celu została stworzona funkcja generująca liczby „pseudolosowe”. Trzeba było wziąć także pod uwagę dobór pierwszego oraz ostatniego indeksu, gdzie ostatnim indeksem powinna być liczba *n*-1 ( *n* – liczba wyrazów ciągu ). W kolejnym kroku stworzono zmienne pomocnicze, w tym dwa wskaźniki *i, j* oraz *piwot*. Algorytm wykorzystując te dane zwraca posortowany ciąg, a także zapisuje go w pliku tekstowym.

## 3.2. Sortowanie przez wybieranie

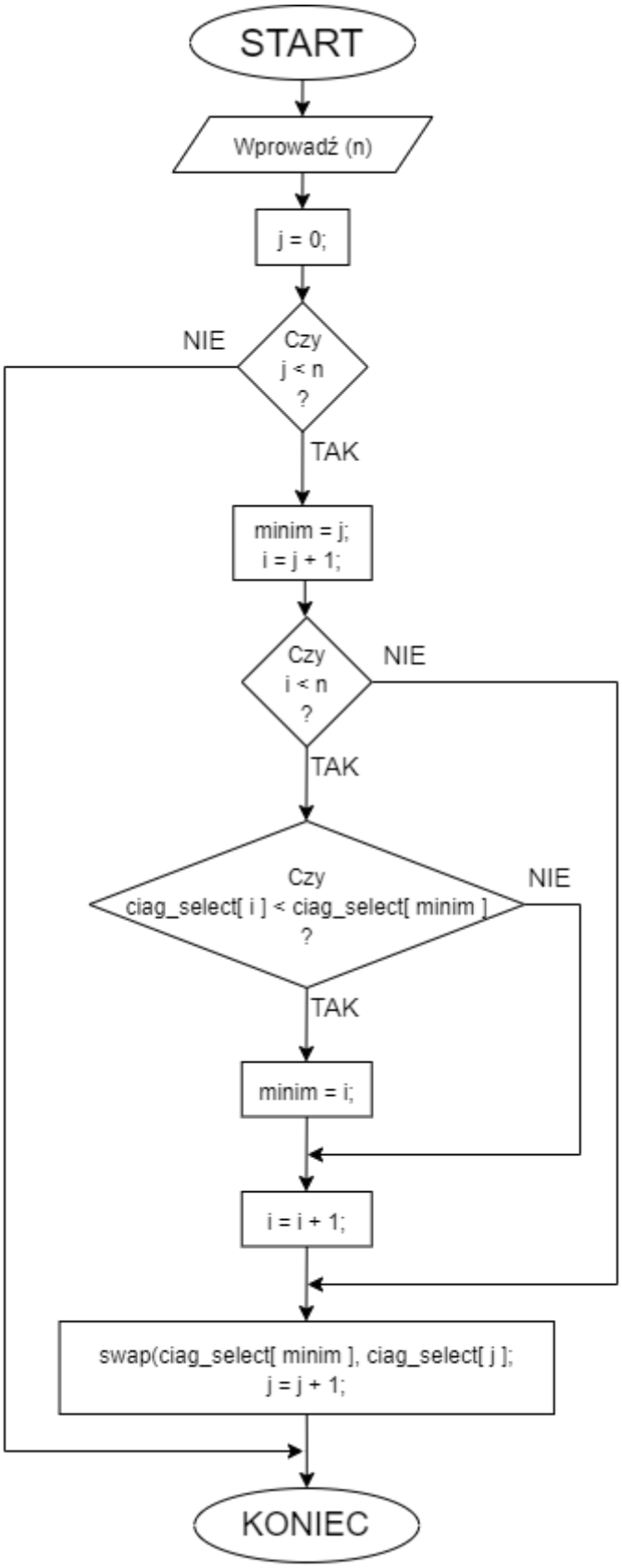
Również w przypadku tego algorytmu wykorzystano funkcje generującą liczby „pseudolosowe”. Do poprawnego działania algorytmu należało wziąć pod uwagę liczbę *n* – wyrazów ciągu. Należało stworzyć także zmienne pomocnicze w tym dwa wskaźniki *i, j* oraz zmienną *minim* „szufladkującą” liczbę jako minimalną. Posiadając te dane algorytm zwraca posortowany ciąg, a także zapisuje go w pliku tekstowym.

# 4. Schematy blokowe

## 4.1. Schemat funkcji: *Quick_Sort*( )



## 4.2. Schemat funkcji: *Sortowanie_Przez_Wybor*( )



# 5. Pseudokod

## 5.1. Pseudokod funkcji: *Quick_Sort*( )

1) Pierwszy = 0; ostatni = n – 1;
2) i = (pierwszy+ostatni) / 2
3) piwot = ciag_quick_sort[ i ]; ciag_quick_sort[ i ] = ciag_quick_sort[ ostatni ];
4) j = pierwszy; i = pierwszy;
5) Czy i < ostatni ?
   a) **TAK:** Przejdź do punktu: 6)
   b) **NIE:** Przejdź do punktu: 10)
6) Czy ciag_quick_sort[ i ] < piwot ?
   a) **TAK:** Przejdź do punktu: 7)
   b) **NIE:** Przejdź do punktu: 8)
7) swap(ciag_quick_sort[ i ], ciag_quick_sort[ j ]); j = j + 1;
8) i = i + 1;
9) **Przejdź do punktu:** 5)
10) ciag_quick_sort[ ostatni ] = ciag_quick_sort[ j ]; ciag_quick_sort[ j ] = piwot;
11) Czy pierwszy < j – 1 ?
    a) **TAK:** Przejdź do punktu: 12)
    b) **NIE:** Przejdź do punktu: 13)
12) Quick_Sort(pierwszy, j – 1)
13) Czy j + 1 < ostatni ?
    a) **TAK:** Przejdź do punktu: 14)
    b) **NIE:** Przejdź do punktu: 15)
14) Quick_Sort(j + 1, ostatni)
15) **KONIEC**

## 5.2. Pseudokod funkcji: *Sortowanie_Przez_Wybor*( )

1) Wprowadź (n)
2) j = 0;
3) Czy j < n ?
   a) **TAK:** Przejdź do punktu: 4)
   b) **NIE:** Przejdź do punktu: 10)
4) minim = j; i = j + 1;
5) Czy i < n ?
   a) **TAK:** Przejdź do punktu: 6)
   b) **NIE:** Przejdź do punktu: 9)
6) Czy ciag_select[ i ] < ciag_select[ minim ] ?
   a) **TAK:** Przejdź do punktu: 7)
   b) **NIE:** Przejdź do punktu: 8)
7) minim = i;
8) i = i + 1;
9) swap(ciag_select[ minim ], ciag_select[ j ]); j = j + 1;
10) **KONIEC**

# 6. Testy porównujące działanie obu algorytmów na różnych próbkach danych

W tym dziale zostaną porównane czasy działa obu algorytmów dla rosnących *n* – próbkach danych. Ilość danych zaczyna się od wartości *n* = 200 i co pętle zwiększa się dwukrotnie.

Tabela 1. Czasy działania algorytmów podane w **sekundach** dla:
*n* – ilość sortowanych danych, *CzasA* – czas algorytmu: Quicksort,
*CzasB* – czas algorytmu: Sortowanie przez wybieranie

| Lp. | n | CzasA [s] | CzasB [s] |
|---|---|---|---|
| 1 | 200 | 0 | 0 |
| 2 | 400 | 0 | 0 |
| 3 | 800 | 0 | 0 |
| 4 | 1600 | 0 | 0.00009 |
| 5 | 3200 | 0.00099 | 0.00299 |
| 6 | 6400 | 0.00710 | 0.01199 |
| 7 | 12800 | 0.05799 | 0.04698 |
| 8 | 25600 | 0.11046 | 0.18795 |
| 9 | 51200 | 0.64681 | 0.73978 |
| 10 | 102400 | 2.58018 | 2.97320 |

Z wyników czasowych jakich otrzymaliśmy z tabeli możemy wywnioskować, że algorytm „Quicksort” jest zdecydowanie szybszym algorytmem sortującym od algorytmu „Sortowanie przez wybieranie”. Wraz ze zwiększającą się ilością danych, różnica pomiędzy czasem algorytmu A, a czasem algorytmu B znacznie zwiększa się.

# 7. Złożoność obliczeniowa algorytmów

Omówiona zostanie złożoność obliczeniowa wraz z uwzględnieniem czasu wykonywania każdego z algorytmów dla rosnącej ilości próbek danych *n*. Wzięta zostanie także pod uwagę klasa złożoności obliczeniowej dla różnych przypadków, które możemy wyróżnić w następujący sposób:

a) Przypadek oczekiwany – dla losowo dobranego zbioru danych,
b) Przypadek optymistyczny – dla najlepiej dopasowanego zbioru danych do zadanego algorytmu, dzięki czemu uzyskamy najszybszy czas posortowania,
c) Przypadek pesymistyczny – dla najgorzej dopasowanego zbioru danych do zadanego algorytmu, przez co uzyskamy najwolniejszy czas sortowania.

Każdy przypadek zostanie także przedstawiony w formie wykresu t(n).

## 7.1. Quicksort

Czas sortowania dla algorytmu „Quicksort" jest zależny od wybranego elementu jako piwot. Im bardziej wybrany element dzieli ciąg na zrównoważone części tym szybszy będzie czas sortowania. W przeciwnym przypadku algorytm może osiągnąć nawet złożoność obliczeniową przybliżoną do złożoności obliczeniowej algorytmu „Sortowanie przez wybieranie" równej ($n^2$). Możemy wyróżnić następujące przypadki ciągów do posortowania:

### 7.1.1. Przypadek oczekiwany

Jest on bliski przypadkowi optymistycznemu. W większości przypadków elementem wybranym jako piwot, zostaje właśnie element dzielący ciąg w sposób najbardziej zrównoważony. Czas działania dla takiego doboru piwota jest określony następującym wzorem:

$$T(n) = O(2n\ ln\ n)$$

W związku, iż 2*n* ln *n* w przybliżeniu jest równe 1,39*n* ln *n* można uznać, że średnia ilość porównań jest większa o około 39% względem przypadku optymistycznego (*n* ln *n*).

### 7.1.2. Przypadek optymistyczny

Zachodzi on w momencie, w którym jako piwot zostaje wybrana z każdym podziałem mediana w każdej części. Dzięki czemu uzyskujemy za każdym razem równomierny podział ciągu. Złożoność taka jest opisana wzorem:

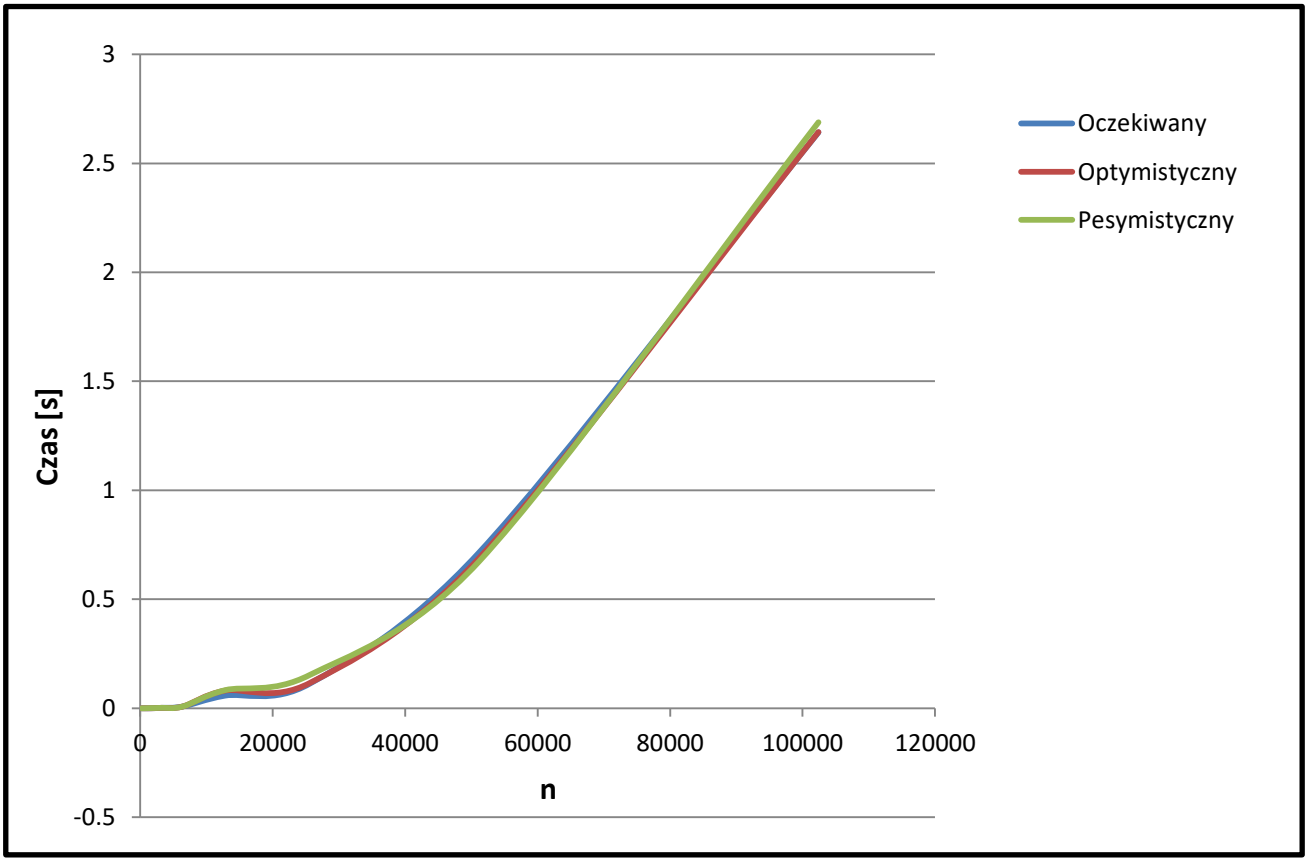
$$T(n) = O(n \log n)$$

### 7.1.3. Przypadek pesymistyczny

Dochodzi do niego w momencie, w którym wylosowanym ciągiem zostaje: ciąg posortowany (malejąco lub rosnąco). Jest to najgorszy przypadek, ponieważ element najmniejszy zostaje wybrany za każdym razem. Dodatkowo cały ciąg musi zostać przeszukany przez wskaźnik co nową pętle, w rezultacie czas całego sortowania przebiega bardzo długo. W ostateczni degraduje się on do klasy złożoności równej:

$$T(n) = O(n^2)$$

Rys. 1. Wykres *t*(*n*) złożoności czasowej algorytmu Quicksort dla przypadków: oczekiwanego, optymistycznego i pesymistycznego.



Z wykresu możemy wywnioskować, że przypadek oczekiwany jest rzeczywiście bardzo zbliżony (wręcz identyczny) do przypadku optymistycznego, natomiast przypadek pesymistyczny odbiega od dwóch poprzednich.

## 7.2. Sortowanie przez wybieranie

W przypadku „Sortowania przez wybieranie" złożoność osiąga taką samą klasę złożoności dla wszystkich przypadków. Jest to złożoność kwadratowa:

$$O(n^2)$$

### 7.2.1. Przypadek oczekiwany

Są w nim uwzględnione losowo dobrane wartości ciągu

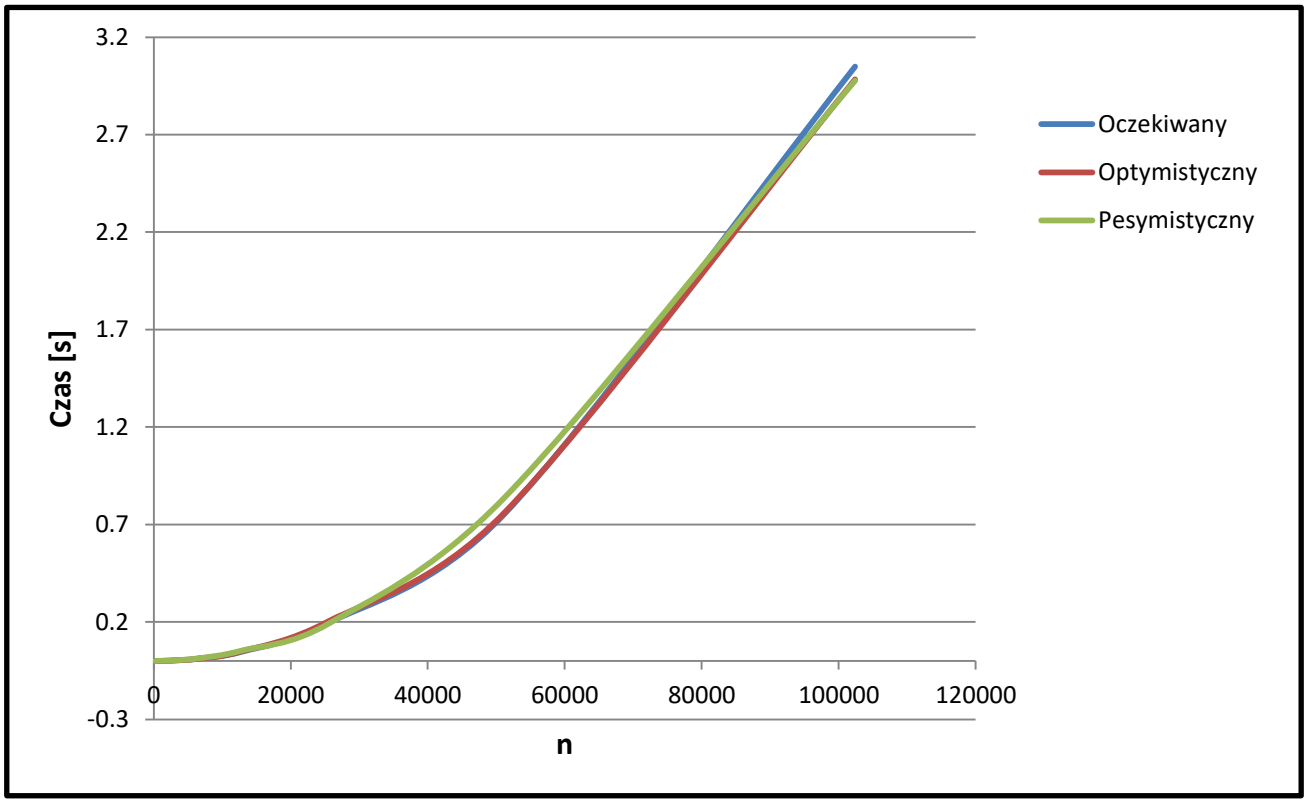
### 7.2.2. Przypadek optymistyczny

Odnosi się do ciągu liczb, który jest już posortowany.

### 7.2.3. Przypadek pesymistyczny

W tym przypadku mamy do czynienia z ciągiem posortowanym w sposób malejący.

Rys. 2. Wykres *t*(*n*) złożoności czasowej algorytmu Sortowanie przez wybieranie dla przypadków: oczekiwanego, optymistycznego i pesymistycznego.



# 8. Wnioski i podsumowanie

W projekcie zostały wzięte pod uwagę dwa algorytmy sortujące: „Quicksort” oraz „Sortowanie przez wybieranie”. Dzięki możliwości doboru dwóch różnych od siebie algorytmów można zauważyć, że pomimo iż wykonują tą samą funkcje różnią się od siebie czasem w jakim posortują dostarczone dane. Czasy te, a właściwie, tzw. złożoność obliczeniową można przedstawić w postaci klas złożoności obliczeniowej, w notacji dużego *O*, która jest określona przez funkcję $T(n)$. Jak się okazuje znaczenie ma także sposób spreparowania dostarczonych danych, dzięki czemu wyróżniamy trzy przypadki klas złożoności obliczeniowej: oczekiwaną, optymistyczną oraz pesymistyczną. Podsumowując całość projektu można uświadomić sobie jak ważny jest dobór określonych algorytmów do określonego problemu, w tym przypadku posortowania ciągu danych.

# 9. Appendix

## 9.1. Kod źródłowy programu

```
/* Zaimplementuj sortowanie przez wybieranie oraz sortowanie metoda quicksort. */

#include <iostream>
#include <fstream>
#include <chrono>
#include <vector>
#include <time.h>
#include <cstdlib>
#include <iomanip>
#include <cstdio>
#include <algorithm>

using namespace std;
using namespace std::chrono;


// Funkcja losujaca ciag liczb "pseudolosowych"
vector<int> Losuj_Ciag(int min, int max, int n);

// Funkcja wypisujaca ciag na ekranie
void Wypisz_Ciag(vector<int> ciag);

// Funkcja zapisujaca ciag do pliku tekstowego
void Zapisz_Ciag_Do_Pliku(string nazwa_pliku, vector<int> ciag);

// Funkcja odczytujaca ciag z pliku tekstowego
vector<int> Odczytaj_Ciag_Z_Pliku(string nazwa_pliku);

// Funkcja wypisujaca ciag z pliku tekstowego na ekranie
void Wypisz_Ciag_Z_Pliku(string nazwa_pliku);

// Funkcja wypisujaca posortwany ciag na ekranie
void Wypisz_Wynik(vector<int> wynik, int n);

// Algorytm sortujacy Quicksort
vector<int> Quick_Sort(int pierwszy, int ostatni);

// Algorytm sortujacy Sortowanie przez wybieranie
vector<int> Sortowanie_Przez_Wybor(int n);

// Funkcja zapisujaca posortowany ciag do pliku
void Zapisz_Wynik_Do_Pliku(const string nazwa_pliku, vector<int> wynik, int n);
```

```
// Funkcja zapisujaca ciag do pliku tekstowego z funkcji: testy()
void Zapisz_Ciag_Do_Pliku_Testy(string nazwa_pliku_ciag, vector<int> ciag_testowy);

// Funkcja zapisujaca czasy wykonywania kazdego z algorytmow do pliku
void Zapisz_Czasy_Do_Pliku(string nazwa_pliku_czas, vector<double> czasy, string litera);

// Funkcja generujaca ciag optymistyczny dla algorytmu Quicksort
vector<int> Ciag_Optymistyczny(vector<int> mediana, int poczatek, int n);

// Funkcja bedaca ciagiem pesymistycznym dla algorytmu Quicksort
// Funkcja bedaca ciagiem optymistycznym dla algorytmu Sortowanie przez wybieranie
vector<int> Posortowany_Ciag(int n);

// Funkcja szukajaca median dla funkcji: Posortowany_Ciag(int n);
vector<int> Szukaj_Median(int poczatek, int n);

// Funkcja bedaca ciagiem pesymistycznym dla algorytmu Sortowanie przez wybieranie
vector<int> Posortowany_Ciag_Malejacy(int n);

// Funkcja wykonujaca testy dla zwiekszajacych sie probek danych 'n'
void testy();


// Licznik czasu wykonywanych funkcji
high_resolution_clock::time_point start;
high_resolution_clock::time_point stop;
duration<double> czas;


vector<int> ciag_quick_sort; // Ciag ktory ma zostac posortowany przez: Quick_Sort
vector<int> ciag_select;     // Ciag ktory ma zostac posortowany przez:
Sortowanie_Przez_Wybor
vector<int> ciag_sort;       // Ciag w ktorym maja zostac odnaleziony mediany
vector<int> ciag_opt;        // Ciag optymistyczny ktory ma zostac posortowany przez:
Quick_Sort
vector<int> ciag_pes;        // Ciag pesymistyczny ktory ma zostac posortowany przez:
Quick_Sort
vector<int> ciag_opt_select; // Ciag optymistyczny ktory ma zostac posortowany przez:
Sortowanie_Przez_Wybor
vector<int> ciag_pes_select; // Ciag pesymistyczny ktory ma zostac posortowany przez:
Sortowanie_Przez_Wybor
```

```
int main()
{
    int n = 20;     // ciag wylosuje 20 liczb
    vector<int> wynik;
    vector<int> ciag;

    //////////////////////////
    /* Odczyt ciagu z pliku */
    //////////////////////////

//    string nazwa_pliku =  "Plik.txt" ;
//    ciag = Odczytaj_Ciag_Z_Pliku(nazwa_pliku);
//    Wypisz_Ciag_Z_Pliku(nazwa_pliku);
//    n = ciag.size();


    ciag = Losuj_Ciag(0, 100, n);
    ciag_quick_sort = ciag;
    ciag_select = ciag;
    Wypisz_Ciag(ciag);
    cout << "------------------------------------------------------" << endl << endl;
    Zapisz_Ciag_Do_Pliku("Wylosowany ciag.txt", ciag);

    ////////////////////////////////////////

    // Quicksort //

    // Sortowanie oraz czas wykonania algorytmu dla wybranej liczby 'n' wartości ciagu
losowego

    start = high_resolution_clock::now();
    wynik = Quick_Sort(0, n-1);
    stop = high_resolution_clock::now();
    czas = stop-start;



    cout << "Posortowany ciag (Quick_Sort): " << endl << endl;
    Wypisz_Wynik(wynik, n);
    cout << endl << "Czas wykonania algorytmu: " << czas.count() << " s." << endl;
    cout << "------------------------------------------------------------" << endl;
    Zapisz_Wynik_Do_Pliku("Posortowany ciag (Quick_Sort).txt", wynik, n);

    ////////////////////////////////////////
```

```
 // Sortowanie Przez Wybor //

    // Sortowanie oraz czas wykonania algorytmu dla wybranej liczby 'n' wartości ciagu losowego

    start = high_resolution_clock::now();
    wynik = Sortowanie_Przez_Wybor(n);
    stop = high_resolution_clock::now();
    czas = stop-start;



    cout << "Posortowany ciag (Sortowanie_Przez_Wybor): " << endl << endl;
    Wypisz_Wynik(wynik, n);
    cout << endl << "Czas wykonania algorytmu: " << czas.count() << " s." << endl;
    Zapisz_Wynik_Do_Pliku("Posortowany ciag (Sortowanie_Przez_Wybor).txt", wynik, n);


    testy();



    return 0;

}



////////////////////////////////////////////////////////////////////////////////////////
/*-------------------------------------------------------------*/
////////////////////////////////////////////////////////////////////////////////////////
```

```
void testy()
{
   int n = 200;
   int ilosc_testow = 10;
   vector<int> wynik_test;
   vector<int> ciag_testowy;
   vector<int> mediana;
   vector<double> czas_quick_sort(ilosc_testow);
   vector<double> czas_select(ilosc_testow);

   cout << "_____" << endl;
   cout << "Testy:" << endl << endl;


   for(int i=0; i<ilosc_testow; i++)
   {

      string nazwa_pliku_ciag_opt = ("Wylosowany ciag OPTYMISTYCZNY TESTY
(Quick_Sort) " + to_string(i) +".txt");
      string nazwa_pliku_ciag_pes = ("Wylosowany ciag PESYMISTYCZNY TESTY
(Quick_Sort) " + to_string(i) +".txt");
      string nazwa_pliku_ciag = ("Wylosowany ciag LOSOWY TESTY " + to_string(i) +".txt");
      string nazwa_pliku_ciag_opt_select = ("Wylosowany ciag OPTYMISTYCZNY TESTY
(Sortowanie_Przez_Wybor) " + to_string(i) +".txt");
      string nazwa_pliku_ciag_pes_select = ("Wylosowany ciag PESYMISTYCZNY TESTY
(Sortowanie_Przez_Wybor) " + to_string(i) +".txt");
      string nazwa_pliku_wynik1 = ("Posortowany ciag TESTY (Quick_Sort) " + to_string(i)
+".txt");
      string nazwa_pliku_wynik2 = ("Posortowany ciag TESTY (Sortowanie_Przez_Wybor) "
+ to_string(i) +".txt");

      cout << "n: " << n << endl;

      // Ciag optymistyczny //

      ciag_sort = Posortowany_Ciag(n);
      mediana = Szukaj_Median(0, n-1);


      // Losowanie ciagu //

      ciag_testowy = Losuj_Ciag(0, 100, n);

      ciag_opt = Ciag_Optymistyczny(mediana, 0, n-1);
      ciag_pes = Posortowany_Ciag(n);

      ciag_opt_select = Posortowany_Ciag(n);
      ciag_pes_select = Posortowany_Ciag_Malejacy(n);
```

```
        ////////////////////////////////
        /* WYBOR CIAGU DO POSORTOWANIA */
        /* Moze byc zaznaczona tylko jedna opcja, dla jednego algorytmu */
        ////////////////////////////////



        // Quicksort //

        /* LOSOWY */

//      ciag_quick_sort = ciag_testowy;
//      Zapisz_Ciag_Do_Pliku_Testy(nazwa_pliku_ciag, ciag_testowy);

        /* OPTYMISTYCZNY */
        ciag_quick_sort = ciag_opt;
        Zapisz_Ciag_Do_Pliku_Testy(nazwa_pliku_ciag_opt, ciag_opt);

        /* PESYMISTYCZNY */
//      ciag_quick_sort = ciag_pes;
//      Zapisz_Ciag_Do_Pliku_Testy(nazwa_pliku_ciag_pes, ciag_pes);



        // Sortowanie Przez Wybor //

        /* LOSOWY */

//      ciag_select = ciag_testowy;
//      Zapisz_Ciag_Do_Pliku_Testy(nazwa_pliku_ciag, ciag_testowy);

        /* OPTYMISTYCZNY */
//      ciag_select = ciag_opt_select;
//      Zapisz_Ciag_Do_Pliku_Testy(nazwa_pliku_ciag_opt_select, ciag_opt_select);

        /* PESYMISTYCZNY */
        ciag_select = ciag_pes_select;
        Zapisz_Ciag_Do_Pliku_Testy(nazwa_pliku_ciag_pes_select, ciag_pes_select);

        ////////////////////////////////
        ////////////////////////////////
```

```
    // Licznik czasu dla: Quick Sort //

      start = high_resolution_clock::now();
      wynik_test = Quick_Sort(0, n-1);
      stop = high_resolution_clock::now();
      czas = stop-start;
      czas_quick_sort[i] = czas.count();
      cout << "CzasA (Quick_Sort) " << i+1 << ": ";
      cout << czas_quick_sort[i] << " s." << endl;



      Zapisz_Wynik_Do_Pliku(nazwa_pliku_wynik1, wynik_test, n);


      // Licznik czasu dla: Sortowanie Przez Wybor //

      start = high_resolution_clock::now();
      wynik_test = Sortowanie_Przez_Wybor(n);
      stop = high_resolution_clock::now();
      czas = stop-start;
      czas_select[i] = czas.count();
      cout << "CzasB (Sortowanie_Przez_Wybor) " << i+1 << ": ";
      cout << czas_select[i] << " s." << endl << endl;
      cout << "-------------------------------------" << endl;


      Zapisz_Wynik_Do_Pliku(nazwa_pliku_wynik2, wynik_test, n);




      n*=2;
   }


   Zapisz_Czasy_Do_Pliku("Czasy Quick Sort.txt", czas_quick_sort, "A");
   Zapisz_Czasy_Do_Pliku("Czasy Sortowanie Przez Wybor.txt", czas_select, "B");


}


//-----------------------------------------------------------------------------
```

```
vector<int> Losuj_Ciag(int min, int max, int n)
{
   srand(time(NULL));

   vector<int> tab;

   for(int i=0; i<n; i++)
   {
      tab.push_back( (rand()%max) + min + 1 );
   }


   return tab;


}



//------------------------------------------------------------------------------



void Wypisz_Ciag(vector<int> ciag)
{

   cout << "Wylosowany ciag: " << endl;

   for(int i=0; i<ciag.size(); i++)
   {
      cout << setw(5) << ciag[i];
   }

   cout << endl << endl;

}



//------------------------------------------------------------------------------
```

```
void Zapisz_Ciag_Do_Pliku(string nazwa_pliku, vector<int> ciag)

{

   ofstream out(nazwa_pliku);

   for(int i=0; i<ciag.size(); i++)
   {
      out << setw(5) << ciag[i];
   }

}

//---------------------------------------------------------------------------

vector<int> Odczytaj_Ciag_Z_Pliku(string nazwa_pliku)
{
   ifstream in(nazwa_pliku);
   vector<int> tab;
   int liczba_ciagu;

   while(!in.eof())
   {
      in >> liczba_ciagu;
      tab.push_back(liczba_ciagu);

   }

   return tab;

}

//---------------------------------------------------------------------------


void Wypisz_Ciag_Z_Pliku(string nazwa_pliku)
{
   ifstream in(nazwa_pliku);
   int liczba_ciagu;

   while(!in.eof())
   {
      in >> liczba_ciagu;
      cout << setw(6) << liczba_ciagu;

   }

}

//---------------------------------------------------------------------------
```

```
vector<int> Quick_Sort(int pierwszy, int ostatni)
{

   int piwot, i, j;


   i = (pierwszy + ostatni) / 2;
   piwot = ciag_quick_sort[i];

   ciag_quick_sort[i] = ciag_quick_sort[ostatni];

   j = pierwszy;
   i = pierwszy;
   while(i < ostatni)
   {
      if(ciag_quick_sort[i] < piwot)
      {
         swap(ciag_quick_sort[i], ciag_quick_sort[j]);
         j++;

      }

      i++;
   }

   ciag_quick_sort[ostatni] = ciag_quick_sort[j];
   ciag_quick_sort[j] = piwot;


   if(pierwszy < j-1)
   {
      Quick_Sort(pierwszy, j-1);
   }

   if(j+1 < ostatni)
   {
      Quick_Sort(j+1, ostatni);

   }


   return ciag_quick_sort;

}



//--------------------------------------------------------------------------------
```

```
vector<int> Sortowanie_Przez_Wybor(int n)
{
   int i, j, minim;

   j = 0;

   while(j < n)
   {
      minim = j;
      i = j + 1;

      while(i < n)
      {
         if(ciag_select[i] < ciag_select[minim])
         {
            minim = i;
         }

         i++;

      }

      swap(ciag_select[minim], ciag_select[j]);

      j++;

   }


   return ciag_select;

}



//-------------------------------------------------------------------------------
```

```
void Wypisz_Wynik(vector<int> wynik, int n)
{


   for(int i=0; i<n; i++)
   {
      cout << setw(5) << wynik[i];
   }

   cout << endl;

}


//-------------------------------------------------------------------------------


void Zapisz_Wynik_Do_Pliku(string nazwa_pliku, vector<int> wynik, int n)
{
   ofstream out(nazwa_pliku);


   for(int i=0; i<n; i++)
   {
      out << setw(5) << wynik[i];
   }

   cout << endl;


}


//-------------------------------------------------------------------------------


void Zapisz_Ciag_Do_Pliku_Testy(string nazwa_pliku_ciag, vector<int> ciag_testowy)
{
   ofstream out(nazwa_pliku_ciag);

   for(int i=0; i<ciag_testowy.size(); i++)
   {
      out << setw(5) << ciag_testowy[i];
   }


}

//-------------------------------------------------------------------------------
```

```
void Zapisz_Czasy_Do_Pliku(string nazwa_pliku_czas, vector<double> czasy, string litera)
{
   ofstream out(nazwa_pliku_czas);

   for(int i=0; i<czasy.size(); i++)
   {
      out << "Czas" << litera << " " << i << ": ";
      out << czasy[i] << " s." << endl;

   }



}



//-------------------------------------------------------------------------------



vector<int> Ciag_Optymistyczny(vector<int> mediana, int poczatek, int n)
{
   int srodek;
   int p, i;


   for(p=0;p<ciag_opt.size(); p++)
   {

      if(ciag_opt[p]==mediana[0])
      {


         srodek =  (poczatek + n ) / 2;


         swap(ciag_opt[p], ciag_opt[srodek]);

      }

   }

   i=1;
   p=0;
```

```
 while(poczatek < srodek-5)
   {

      if(ciag_opt[p]==mediana[i])
      {
         srodek =  (poczatek + srodek-1) / 2;

         swap( ciag_opt[srodek], ciag_opt[p] );

         i++;
         p=0;
      }
      else
      {

         p++;
      }

   }

   srodek =  (poczatek + n ) / 2;

   while(srodek+5 < n)
   {

      if(ciag_opt[p]==mediana[i])
      {
         srodek =  (srodek+1 + n ) / 2;

         swap( ciag_opt[srodek], ciag_opt[p] );

         i++;
         p=0;
      }
      else
      {

         p++;
      }

   }

   return ciag_opt;

}


//------------------------------------------------------------------------------
```

```
vector<int> Posortowany_Ciag(int n)
{
   vector<int> ciag;

   ciag = Losuj_Ciag(0, 100, n);
   ciag_opt = ciag;


   sort( ciag.begin(), ciag.end() );

   return ciag;

}

//---------------------------------------------------------------------------


vector<int> Szukaj_Median(int poczatek, int n)
{

   vector<int> mediana;

   int ind_mediana =  (poczatek + n ) / 2;

   mediana.push_back( ciag_sort[ind_mediana] );


   while(poczatek < ind_mediana-5)
   {
      ind_mediana =  (poczatek + ind_mediana-1) / 2;

      mediana.push_back( ciag_sort[ind_mediana] );
   }

   ind_mediana =  (poczatek + n ) / 2;


   while(ind_mediana+5 < n)
   {
      ind_mediana =  (ind_mediana+1 + n ) / 2;
      mediana.push_back( ciag_sort[ind_mediana] );
   }


   return mediana;

}

//---------------------------------------------------------------------------
```

```
vector<int> Posortowany_Ciag_Malejacy(int n)
{
   vector<int> ciag;


   ciag = Losuj_Ciag(0, 100, n);


   sort( ciag.begin(), ciag.end(), greater<int>() );


   return ciag;

}
```