

# Zadanie Projektowe nr 2

## Algorytmy i Struktury Danych

Inżynieria i Analiza Danych

Kinga Trojanowska  
Nr Indeksu 166710  
Grupa nr 8

# Spis treści

Wstęp .....	2
Treść zadania .....	2
Opis podstaw teoretycznych zagadnienia.....	3
Sortowanie przez wstawianie.....	3
Sortowanie grzebieniowe.....	4
Opis szczegółów implementacji problemu.....	5
Schemat blokowy algorytmu.....	7
Sortowanie przez wstawianie.....	7
Sortowanie grzebieniowe.....	8
Pseudokody .....	9
Sortowanie przez wstawianie.....	9
Sortowanie grzebieniowe.....	10
Rezultaty testów.....	11
Złożoność czasowa .....	11
Sortowanie przez wstawianie.....	11
Sortowanie grzebieniowe.....	11
Porównanie złożoności w tabeli .....	13
Losowy ciąg liczb .....	13
Wnioski .....	15
Kod programu.....	16

# Wstęp

Niniejsze sprawozdanie przedstawia wykonanie projektu z przedmiotu „Algorytmy i struktury danych”. Zawiera opis i treść zadania, szczegóły w jaki sposób zostało ono przeze mnie wykonane. Zamieszczone zostały również schematy blokowe oraz pseudokod algorytmów. Przedstawione oraz opisane są rezultaty testów wykonane przy pomocy napisanego programu. Omówione zostały także pojęcia takie jak złożoność czasowa, obliczeniowa. Ukazane są moje wnioski i przemyślenia odnośnie zadanego projektu. Jego celem była implementacja kodu czyli proces napisania i utworzenia kodu programu w środowisku C++ tj. Codeblocks.

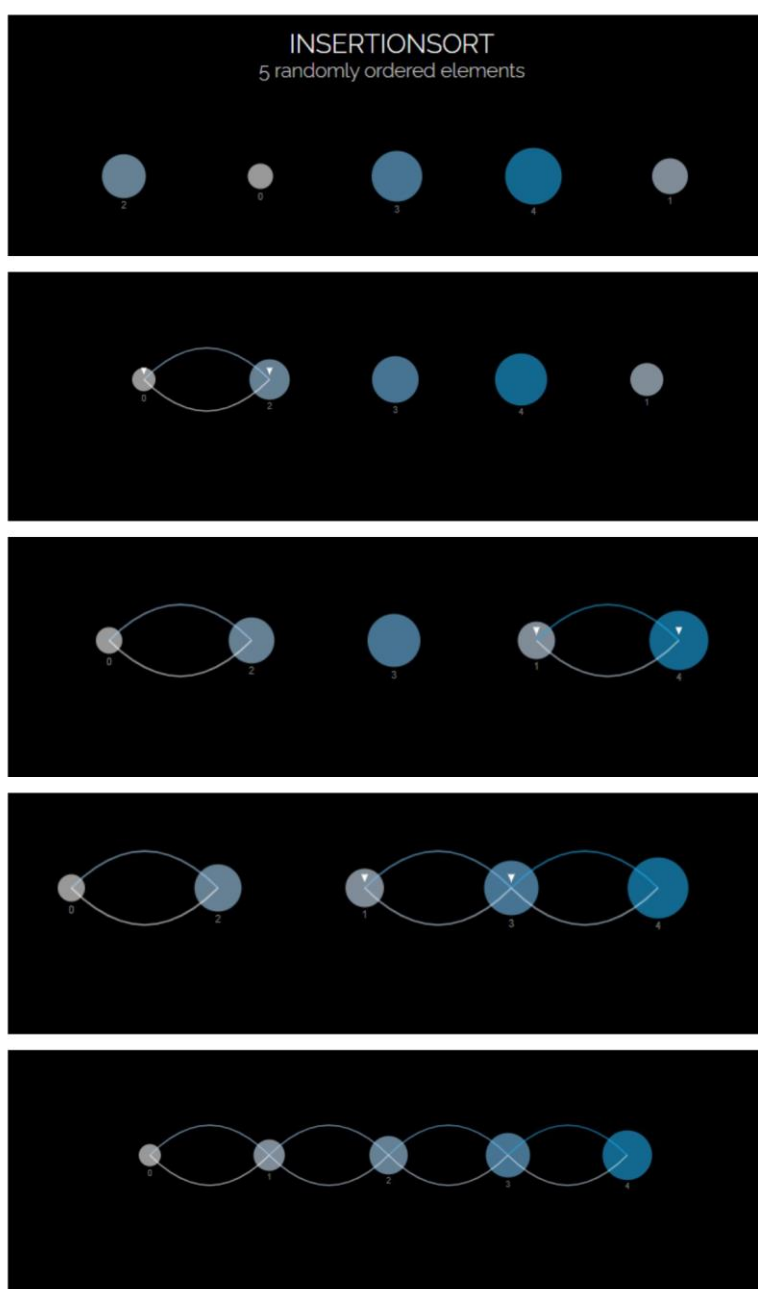
## Treść zadania

Zaimplementuj sortowanie przez wstawianie oraz sortowanie grzebieniowe. Wykonaj testy porównujące działanie obu metod. Przedstaw ich teoretyczne podstawy.

# Opis podstaw teoretycznych zagadnienia

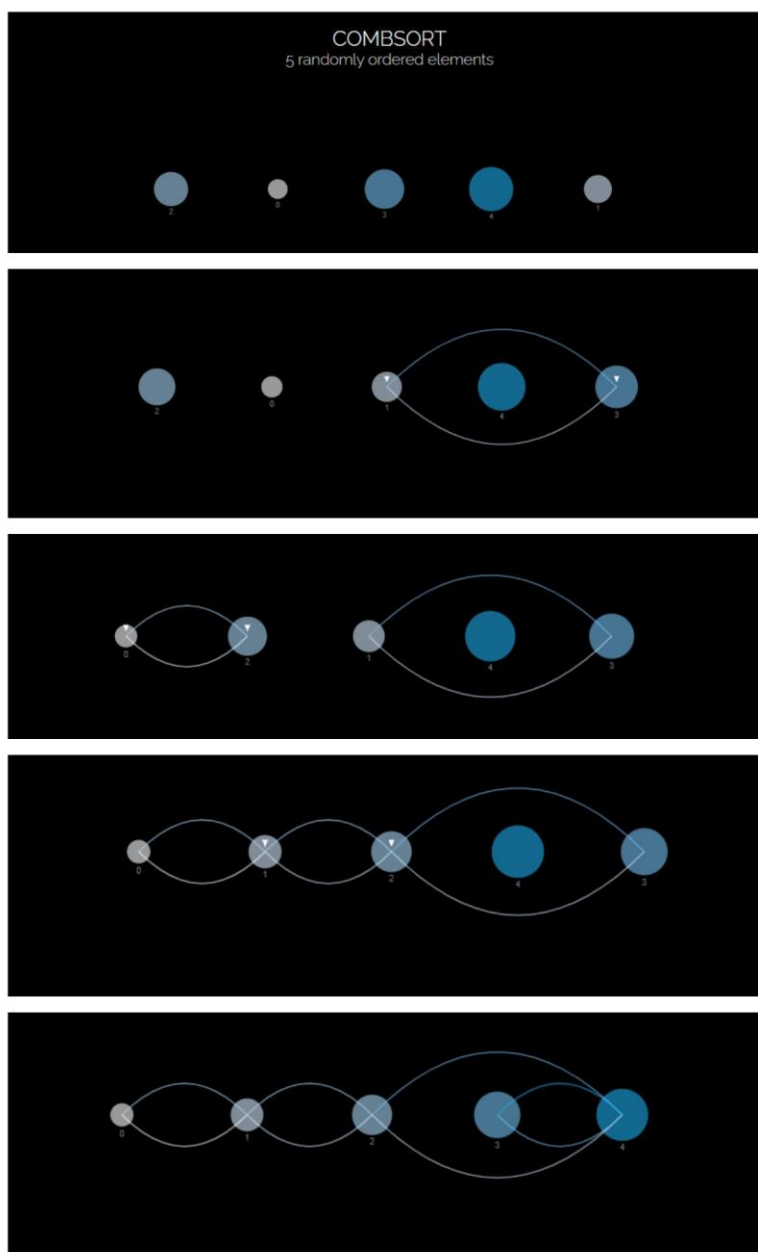
## Sortowanie przez wstawianie

Jeden z najprostszych algorytmów sortowania, którego zasada działania odzwierciedla sposób układania kart przez pokerzystę - kolejne elementy wejściowe są ustawiane na odpowiednie miejsca docelowe. Algorytm przedstawia się następująco. Każdą iterację zaczynamy od pierwszego miejsca w tablicy, na którym znajdują się element, który będziemy przyrównywać, do elementów znajdujących się na pozycjach poprzedzających. Jeżeli element znajdujący się przed wybranym przez nas elementem okaże się większy, przesuwamy się o jedno miejsce wstecz. Dzieje się tak do momentu, kiedy znajdziemy się na pozycji, w której element poprzedzający jest mniejszy, lub dojdziemy do początku tablicy. Wybrany element zostaje wstawiony na ową pozycję. Wszystkie elementy, znajdujące się przed jego wcześniejszą pozycją, przesunięte zostają o jedną pozycję w prawo. Algorytm jest powtarzany, do momentu, aż zanalizowany zostanie ostatni element tablicy.



## Sortowanie grzebieniowe

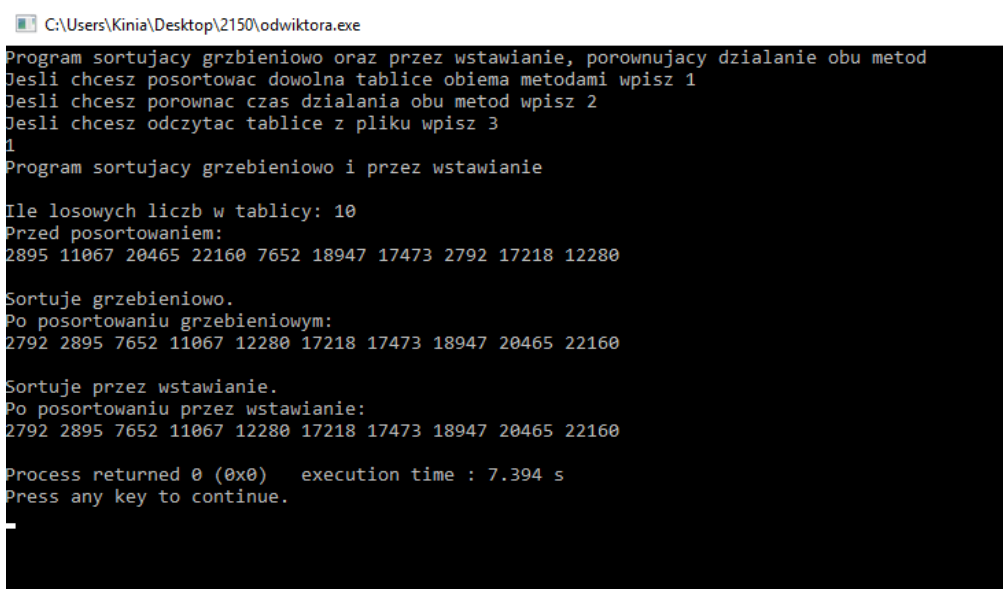
Rodzaj sortowania oparty na metodzie bubblesort. Zasada działania algorytmu przypomina czesanie włosów przy użyciu grzebienia. Początkowo rozczesujemy włosy stroną grzebienia z większymi odstępami pomiędzy ząbkami, dzięki temu łatwiej rozczeszą się pojedyncze pasma. Algorytm sortowania grzebieniowego cechują się empirią czyli wyliczonym doświadczalnie współczynnikiem 1.3. Ma on wpływ na szybkość wykonywania algorytmu. W ten sposób owe sortowanie jest szybsze od sortowania bąbelkowego, jednakże statystycznie wolniejsze od sortowania szybkiego. Podczas etapu sortowania istotną kwestią jest wyliczana w każdej iteracji rozpiętość. Na początku jest nią długość listy. W każdej iteracji dzielimy aktualną wartość rozpiętości przez współczynnik. Ważną kwestią jest fakt, że część ułamkowa po wyliczeniach zostaje odrzucona. Porównywane są tylko pary elementów, które są odległe o rozpiętość. W przypadku spełnienia wymogów zostają zamienione miejscami podobnie jak w metodzie sortowania bąbelkowego. Ze względu na rozpiętość algorytm nie wie kiedy skończyć, dlatego potrzebna jest dodatkowa zmienna, która sprawdza czy wykonaliśmy jakąkolwiek zamianę, kiedy rozpiętość wynosi 1.



## Opis szczegółów implementacji problemu

Wykonanie programu wymagało wglębnienia się w zagadnienia sortowania, implementacja algorytmów została utworzona w osobnych funkcjach. W funkcji głównej zawarte są podstawowe pętle i instrukcje dzięki którym po uruchomieniu konsoli użytkownik musi dokonać wyboru.

Do rozpatrzenia umieszczone zostały trzy opcje. Jeśli odbiorca chce posortować dowolną tablicę obiema metodami, chce wykonać testy porównujące czasy działania obu metod bądź chce posortować tablicę zawartą w pliku wpisuje kolejno liczby 1,2 lub 3. W opcji 1 użytkownik ma za zadanie podać rozmiar tworzonego ciągu liczb do posortowania. Następuje dynamiczna alokacja tablic oraz inicjowanie generatora, który losuje liczby z zakresie od 1 do 100000. Elementy tablicy1 zostają skopiowane do „tablicy2”. Przechodzimy do głównego celu czyli sortowania grzebieniowego oraz przez wstawianie. Wyświetlona jest tablica przed i po sortowaniu dla obu przypadków.



```

C:\Users\Kinia\Desktop\2150\odwiktora.exe
Program sortujacy grzebieniowo oraz przez wstawianie, porownujacy dzialanie obu metod
Desli chcesz posortowac dowolna tablice obiema metodami wpisz 1
Desli chcesz porownac czas dzialania obu metod wpisz 2
Desli chcesz odczytac tablice z pliku wpisz 3
1
Program sortujacy grzebieniowo i przez wstawianie

Ile losowych liczb w tablicy: 10
Przed posortowaniem:
2895 11067 20465 22160 7652 18947 17473 2792 17218 12280

Sortuje grzebieniowo.
Po posortowaniu grzebieniowym:
2792 2895 7652 11067 12280 17218 17473 18947 20465 22160

Sortuje przez wstawianie.
Po posortowaniu przez wstawianie:
2792 2895 7652 11067 12280 17218 17473 18947 20465 22160

Process returned 0 (0x0)   execution time : 7.394 s
Press any key to continue.

```

Druga opcja różni się wyłącznie faktem, że posiada serie 20 testów dzięki nim oraz zaimplementowanemu odmierzaniu czasu jesteśmy w stanie zauważyć, która metoda sortowania działa sprawniej i lepiej na określonej liczbie elementów ciągu.

C:\Users\Kinia\Desktop\2150\odwiktora.exe

```
Program sortujacy grzebieniowo oraz przez wstawianie, porownujacy dzialanie obu metod
Desli chcesz posortowac dowolna tablice obima metodami wpisz 1
Desli chcesz porownac czas dzialania obu metod wpisz 2
Desli chcesz odczytac tablice z pliku wpisz 3
2
```

Ile losowych liczb w tablicy: 1000

Sortuje grzebieniowo.

Ilosc danych w tablicy: 1000

Czas sortowania grzebieniowego: 0 s

Sortuje przez wstawianie.

Ilosc danych w tablicy: 1000

Czas sortowania przez wstawianie: 0.003 s

Ile losowych liczb w tablicy: 2000

Sortuje grzebieniowo.

Ilosc danych w tablicy: 2000

Czas sortowania grzebieniowego: 0 s

Sortuje przez wstawianie.

Ilosc danych w tablicy: 2000

Czas sortowania przez wstawianie: 0.011 s

Opcja trzecia pozwala odbiorcy posortować ciąg liczb zapisany w wybranym pliku. Stworzona została specjalna pętla for, która wczytuje liczby z pliku do „tablica[i]”. W każdej z wybranych opcji zawarte są instrukcje dzięki którym najistotniejsze kwestie zapisywane są do pliku „wyniki.txt”.

C:\Users\Kinia\Desktop\2150\odwiktora.exe

```
Program sortujacy grzebieniowo oraz przez wstawianie, porownujacy dzialanie obu metod
Desli chcesz posortowac dowolna tablice obima metodami wpisz 1
Desli chcesz porownac czas dzialania obu metod wpisz 2
Desli chcesz odczytac tablice z pliku wpisz 3
3
```

Sortuje grzebieniowo.

Ilosc danych w tablicy: 100

Czas sortowania grzebieniowego: 0 s

Sortuje przez wstawianie.

Ilosc danych w tablicy: 100

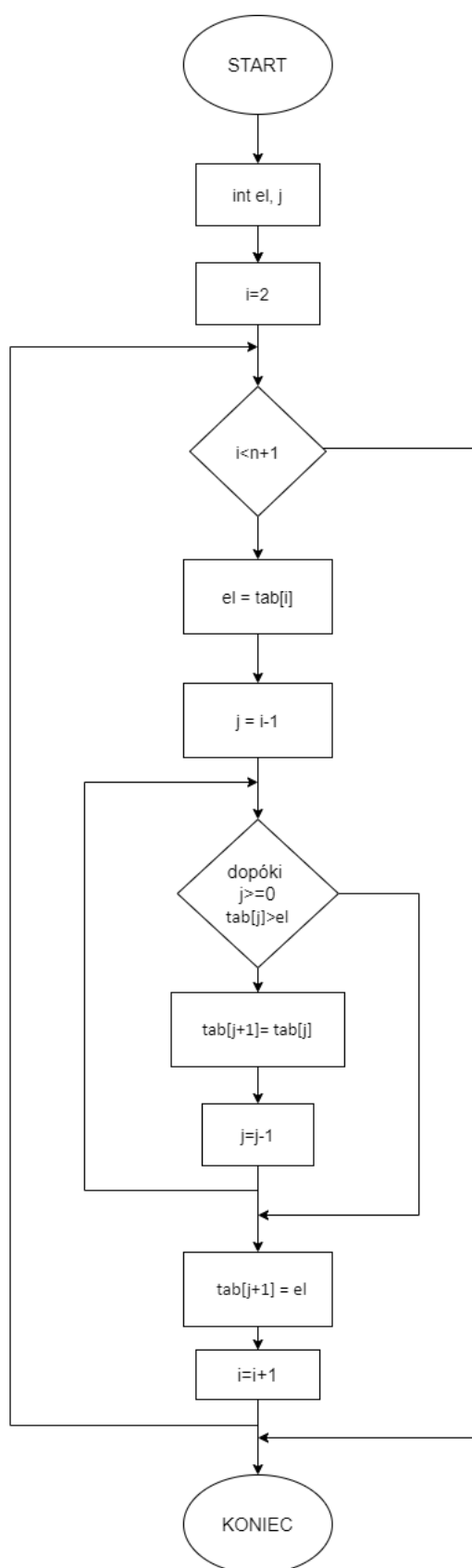
Czas sortowania przez wstawianie: 0 s

Process returned 0 (0x0) execution time : 1.782 s

Press any key to continue.

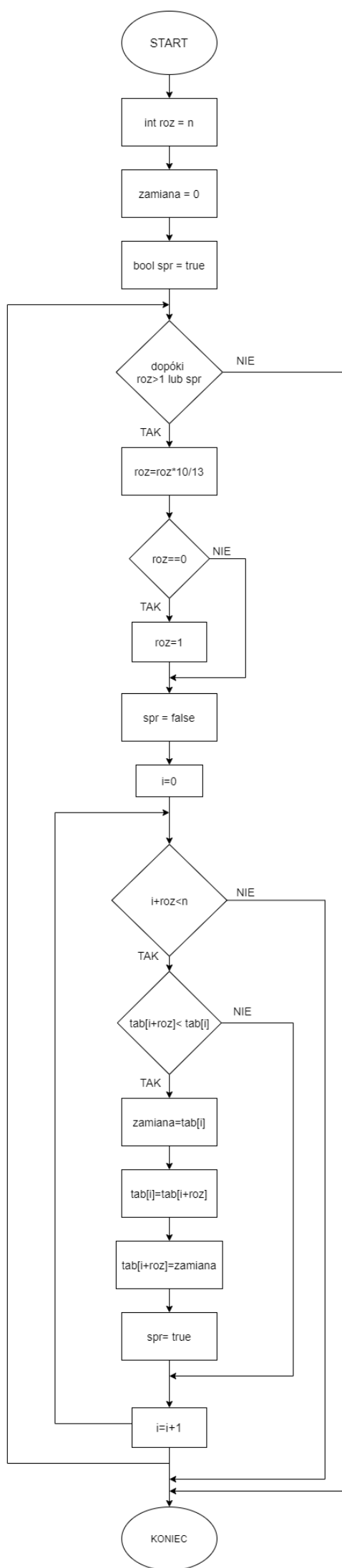
# Schemat blokowy algorytmu

Sortowanie przez wstawianie





# Sortowanie grzebieniowe



# Pseudokody

Pseudokod jest to połączenie języka naturalnego z elementami języka programowania.

## Sortowanie przez wstawianie

funkcja sortowanie przez wstawianie(tab, n)

el , j;

dla  $i \leftarrow 2$  mniejszego od n wykonuj

el  $\leftarrow$  tab[i];

j  $\leftarrow$  i-1;

dopóki  $j > 0$  i tab[j]>el wykonuj

tab[j+1] = tab[j];

zmniejsz j o 1;

tab[j+1] = el;

## Sortowanie grzebieniowe

Funkcja sortowanie grzebieniowe (tab, n)

roz  $\leftarrow$  n,

zamiana  $\leftarrow$  0;

bool spr = true;

dopóki (roz > 1 lub spr) wykonuj

roz = roz\*10/13;

jeśli roz jest równe 0

roz  $\leftarrow$  1;

spr = false;

dla i  $\leftarrow$  0 po dodaniu roz mniejsze od n wykonuj

Jeśli tab[i + roz] < tab[i]

zamiana = tab[i];

tab[i] = tab[i + roz];

tab[i + roz] = zamiana ;

spr = true;

# Rezultaty testów

## Złożoność czasowa

### Sortowanie przez wstawianie

#### *Przypadek optymistyczny*

Przypadek optymistyczny występuje dla ciągu danych uporządkowanych rosnąco. W każdym przebiegu pętli for sprawdzany jest warunek pętli while zachodzi wówczas  $el > tab[j]$ . Pętla while nie zostaje wykonana ani razu, a więc liczba sprawdzeń warunku pętli ( $\delta(i)$ ) wynosi 1 dla wszystkich  $j$  od 1 do  $n$ . Całkowity czas wykonania algorytmu jest więc wyrażony funkcją liniową. Zatem algorytm działa w czasie  $O(n)$ .

#### *Przypadek pesymistyczny*

Przypadek pesymistyczny jest zauważalny gdy ciąg liczb do posortowania jest uporządkowany malejąco, w każdym przebiegu pętli for przy sprawdzaniu warunków pętli while zachodzi  $el < tab[j]$ . Pętla while jest więc w każdym przebiegu pętli for wywoływana  $i-1$  razy. Otrzymuje się zatem złożoność czasową kwadratową  $O(n^2)$ .

#### *Przypadek oczekiwany*

Przypadek oczekiwany można dostrzec jeśli każda z liczb jest tak samo prawdopodobna. W każdym wykonaniu pętli głównej wystąpi od 1 do  $j$  porównań. Zakładając, że zachodzi powyższa sytuacja średnia liczba porównań wynosi  $(j+1)/2$ . Nadal jest to zatem złożoność kwadratowa  $O(n^2)$ .

### Sortowanie grzebieniowe

#### *Przypadek optymistyczny*

Najlepsza konfiguracja występuje, gdy wszystkie elementy są już posortowane lub prawie posortowane. W takim przypadku pętla zostają uruchomione tylko raz. A złożoność wygląda następująco  $O(n \log n)$ . Jest to złożoność liniowo-logarytmiczna.

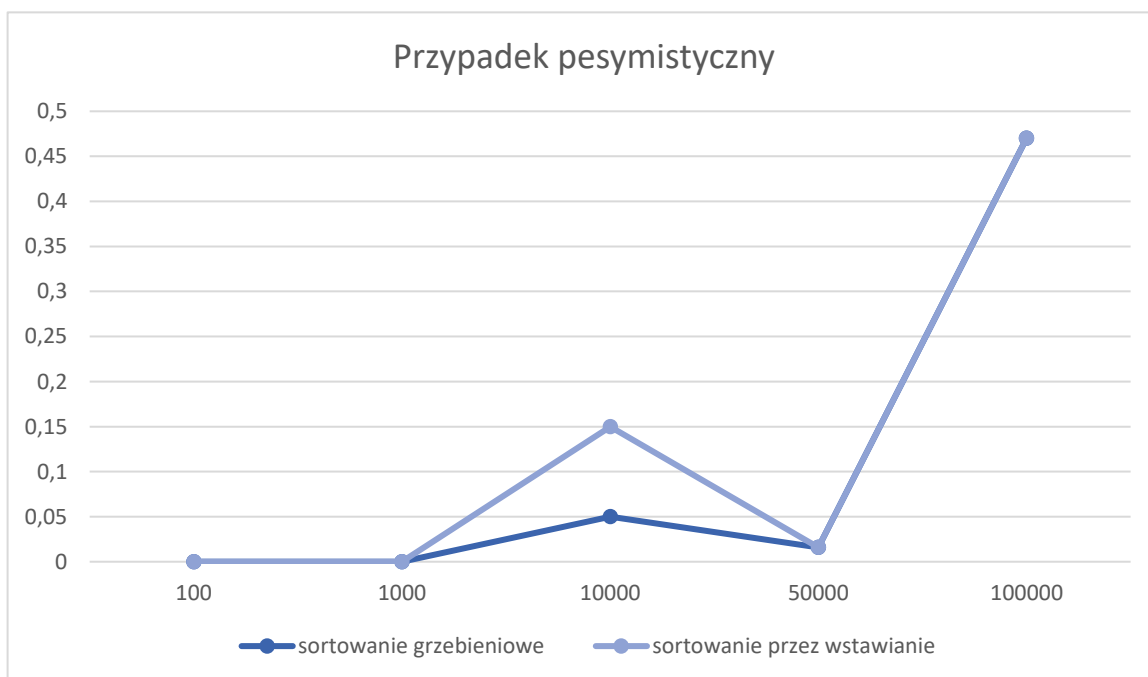
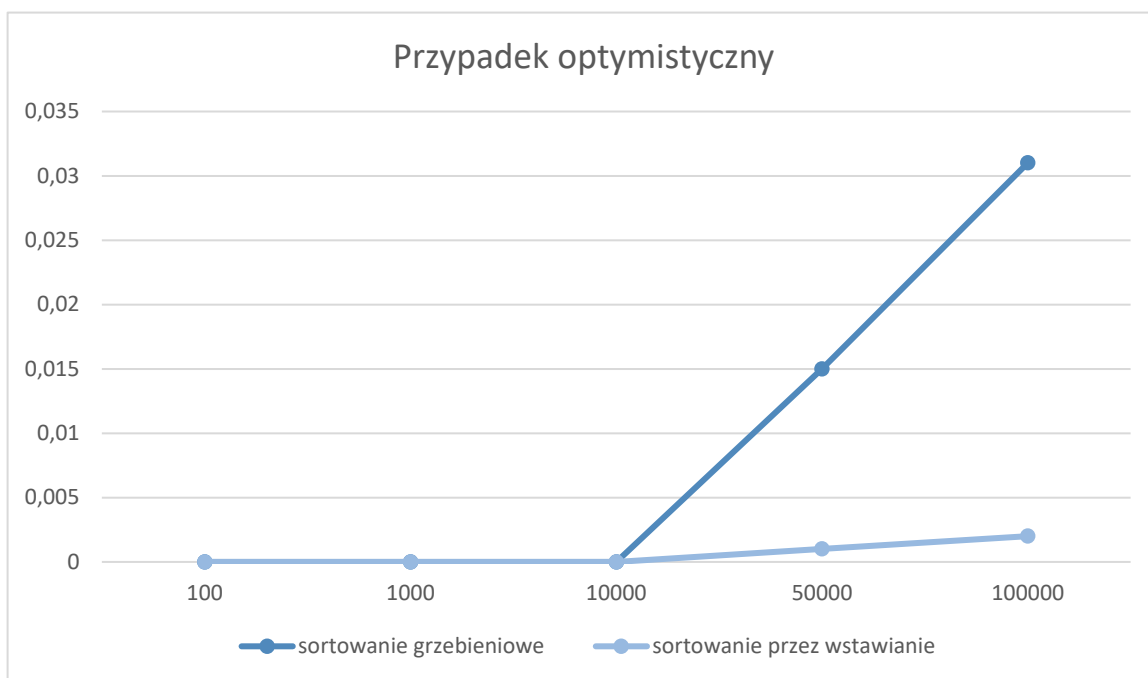
#### *Przypadek pesymistyczny*

Przypadek pesymistyczny jest zauważalny gdy ciąg liczb do posortowania jest uporządkowany malejąco. Jest to złożoność czasowa kwadratowa  $O(n^2)$ .

#### *Przypadek oczekiwany*

Złożoność obliczeniowa przypadku oczekiwanego określa się następująco  $O(n^2/2^p)$ . Zmienna  $p$  określa liczbę przyrostów.

Wykresy porównujące czas działania obu algorytmów dla przypadków pesymistycznych i optymistycznych

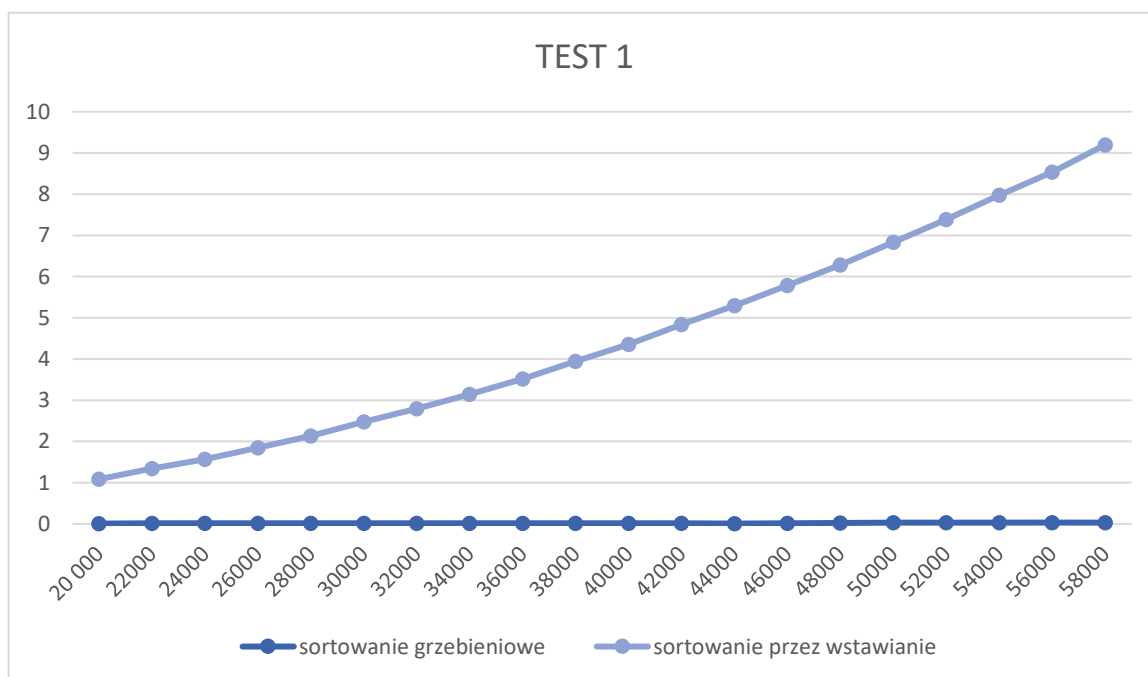


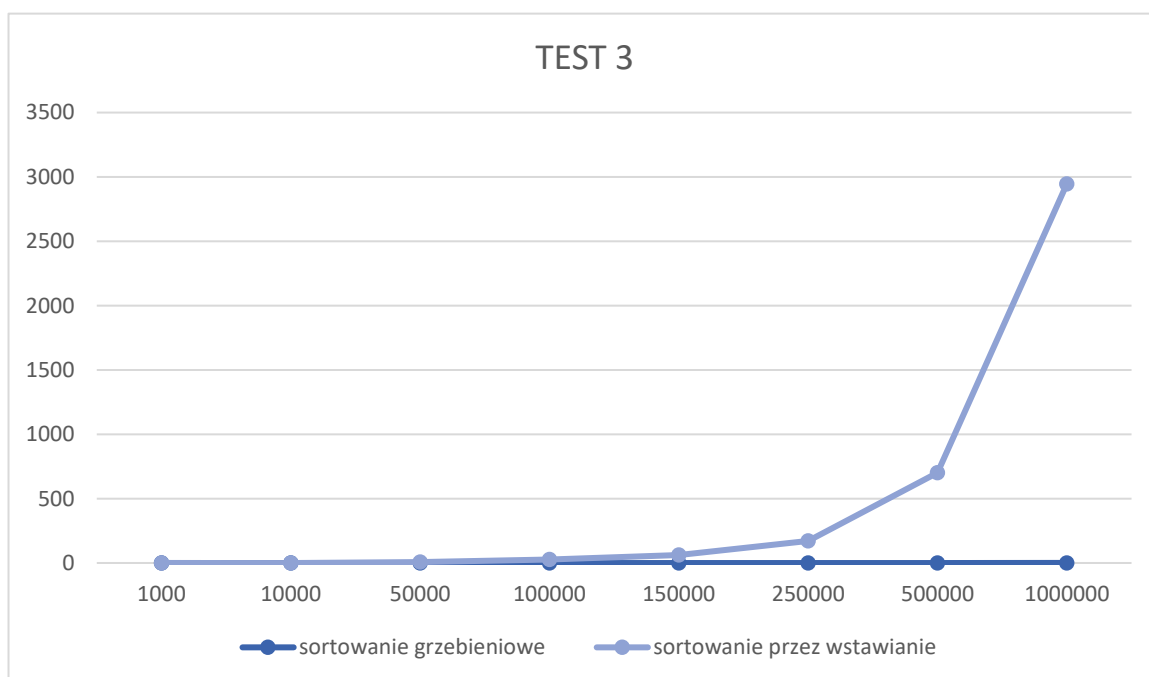
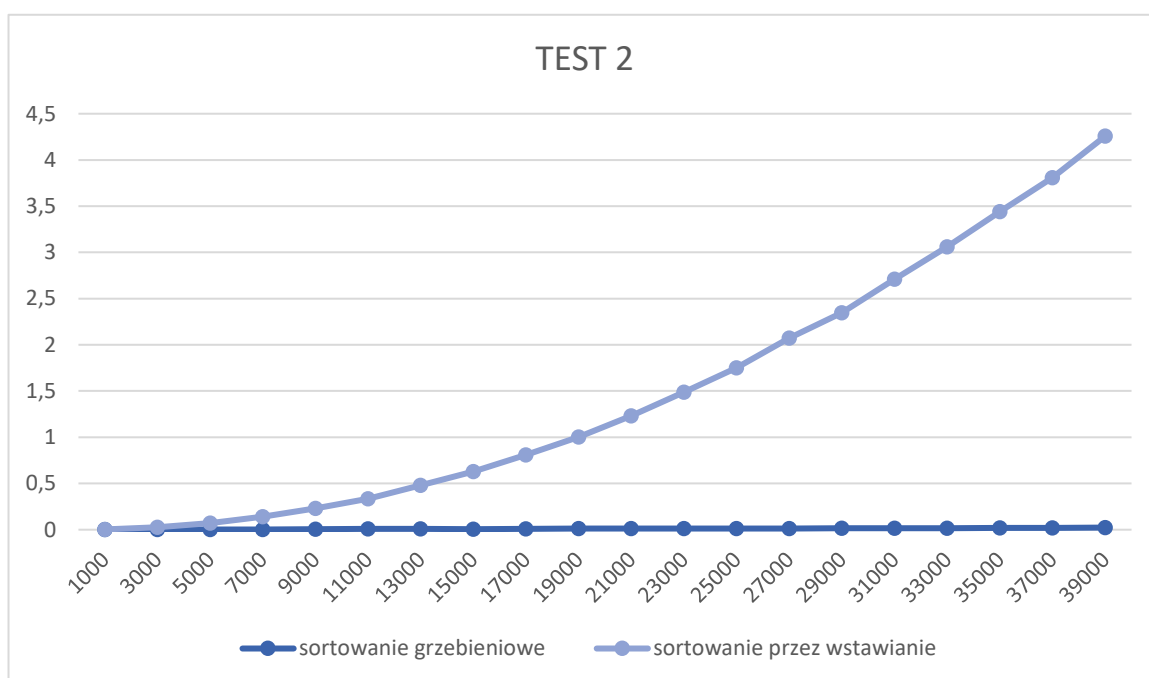
### Porównanie złożoności w tabeli

	Przypadek optymistyczny	Przypadek oczekiwany	Przypadek pesymistyczny
Sortowanie grzebieniowe	$O(n \log n)$	$O(n^2/2^p)$	$O(n^2)$
Sortowanie przez wstawianie	$O(n)$	$O(n^2)$	$O(n^2)$

### Losowy ciąg liczb

W programie zaimplementowana została opcja nr 2, której implementacja opisana jest w powyższym podpunkcie. Na podstawie wykonanych testów oraz mierzonego czasu względem ilości elementów w sortowanym ciągu stworzyłam testy, które pokazują o ile szybsze jest sortowanie grzebieniowe w porównaniu z sortowaniem przez wstawianie.





## Wnioski

Program działa poprawnie. Algorytmy i przewidywane działania na stworzenie programu okazały się w większości przypadków trafne. Zadanie projektowe pozwoliło mi na lepsze zrozumienie pojęcia funkcji w programowaniu oraz poćwiczenie podstawowych pętli jak i instrukcji warunkowych. Jednocześnie byłam w stanie przekonać się jak łatwo można utworzyć tablice, której rozmiar wybiera sam użytkownik. Podsumowując jestem zadowolona, że byłam w stanie przećwiczyć tyle zagadnień z kategorii programowania oraz nauczyć się i pojąć wiedzę z zastosowań sortowania. Uważam, że była to możliwość do rozwijania kreatywnego myślenia jak i szukania różnorodnych rozwiązań.



## Kod programu

```
#include <iostream>
#include <fstream>
#include <string>
#include <time.h>
#include <cstdlib>
#include <stdio.h>

using namespace std;

int ile;
clock_t start, stop;
double czas;

void sortowanie_grzebieniowe (int*tab, int n)
{
    int roz = n, zamiana = 0; //roz=rozpietosc

    bool spr = true; //spr-zmienna sprawdzajaca czy w danej iteracji zostal3y zamienione miejscami
    jakies dwa elementy

    while (roz > 1 || spr)
    {
        roz = roz*10/13;

        if(roz==0)
            roz=1;

        spr = false;

        for (int i=0; i + roz< n; ++i)
        {
            if (tab[i + roz] < tab[i])
            {
                zamiana = tab[i];
```

```

        tab[i] = tab[i + roz];

        tab[i + roz] = zamiana ;

        spr = true;
    }

}

}

}

```

```

void sortowanie_przez_wstawianie(int *tab, int n)
{
    int el, j;
    for (int i=2; i<n ; i++)
    {
        el=tab[i]; //wstawienie elementu w odpowiednie miejsce
        j=i-1;
        while ((j>0) && (el<tab[j]))//przesuwanie elementów większych od el
        {
            tab[j+1]=tab[j];

            j--;
        }
        tab[j+1]=el;//wstawienie el w odpowiednie miejsce
    }
}

```

```

int main()
{
    fstream plik, wyniki;

    plik.open("tablica.txt",ios::in); // Otwarcie/utworzenie pliku, z którego pobierana będzie tablica
    wyniki.open("wyniki.txt",ios::out); // Otwarcie/utworzenie pliku, do którego zapisywany będzie
    wynik

```

```
cout<<"Program sortujacy grzbieniowo oraz przez wstawianie, porownujacy dzialanie obu
metod"<<endl;
```

```
int x;
```

```
cout<<"Jesli chcesz posortowac dowolna tablice obiema metodami wpisz 1 "<<endl;
```

```
cout<<"Jesli chcesz porownac czas dzialania obu metod wpisz 2 "<<endl;
```

```
cout<<"Jesli chcesz odczytac tablice z pliku wpisz 3"<<endl;
```

```
cin >> x;
```

```
while((x!=1)&&(x!=2)&&(x!=3))//jezeli x nie jest rowny 1,2 lub 3 postepuj zgodnie z petla
```

```
{
```

```
    cout<<"Wprowadz liczbe 1, 2 lub 3! " <<endl;
```

```
    cin>>x;
```

```
}
```

```
if(x==1)
```

```
{
```

```
    cout << "Program sortujacy grzebieniowo i przez wstawianie" << endl<<endl;
```

```
    cout<<"Ile losowych liczb w tablicy: ";
```

```
    cin>>ile;
```

```
int *tablica; //dynamiczna alokacja tablicy
```

```
tablica=new int [ile];
```

```
int *tablica2;
```

```
tablica2=new int [ile];
```

```
srand(time(NULL)); //inicjowanie generatora
```

```

for(int i=0; i<ile; i++) //wczytywanie losowych liczb do tablicy
{
    tablica[i] = rand()%100000+1;
}

for(int i=0; i<ile; i++) //przepisanie tablicy do tablicy2
{
    tablica2[i]=tablica[i];
}

cout<<"Przed posortowaniem: "<<endl;
wyniki<<"Przed posortowaniem: "<<endl;
for(int i=0; i<ile; i++)
{
    cout<<tablica[i]<<" ";
    wyniki<<tablica[i]<<" ";
}

cout<<endl<<endl<<"Sortuje grzebieniowo."<<endl;
sortowanie_grzebieniowe (tablica,ile);

cout<<"Po posortowaniu grzebieniowym: "<<endl;
wyniki <<"\n"<< "Po posortowaniu grzebieniowym: "<<endl; //wczytywanie tablicy do pliku
for(int i=0; i<ile; i++)
{
    cout<<tablica[i]<<" ";
    wyniki<< tablica[i] <<" "; //wczytywanie tablicy do pliku
}

cout<<endl<<endl<<"Sortuje przez wstawianie."<<endl;
sortowanie_przez_wstawianie (tablica2,ile);

```

```

cout<<"Po posortowaniu przez wstawianie: "<<endl;
wyniki <<"\n"<< "Po posortowaniu przez wstawianie: "<<endl;
for(int i=0; i<ile; i++)
{
    cout<<tablica[i]<<" ";
    wyniki<< tablica[i] <<" ";
}

cout<<endl;
delete [] tablica;
delete [] tablica2;
}

if (x==2)
{
    int liczba_testow=20;

    for(int i=0; i<liczba_testow; i++)
    {
        cout<<endl<<"Ile losowych liczb w tablicy: ";
        cin>>ile;

        int *tablica; //dynamiczna alokacja tablicy
        tablica=new int [ile];
        int *tablica2;
        tablica2=new int [ile];

        srand(time(NULL)); //inicjowanie generatora

        for(int i=0; i<ile; i++) //wczytywanie losowych liczb do tablicy
        {

```

```

    tablica[i] = rand()%100000+1;
}
for(int i=0; i<ile; i++) //przepisanie tablicy do tablicy2
{
    tablica2[i]=tablica[i];
}

//testowe wypisanie losowo wygenerowanej tablicy
/*  cout<<"Przed posortowaniem: "<<endl;
    wyniki<<"Przed posortowaniem: "<<endl;
    for(int i=0; i<ile; i++)
    {
        cout<<tablica[i]<<" "; //mozliwosc zmiany na tablica2[i] w celu sprawdzenia
poprawnošci sortowania metoda przez wstawianie
        wyniki<<tablica[i]<<" ";

    }

    /*testy algorytmu sortowania grzebieniowego*/
    cout<<endl<<endl<<"Sortuje grzebieniowo."<<endl;
    start = clock();
    sortowanie_grzebieniowe (tablica,ile);
    stop = clock();
    czas = (double)(stop-start) / CLOCKS_PER_SEC;
    cout<<"Ilošc danych w tablicy: "<<ile<<endl;
    cout<<"Czas sortowania grzebieniowego: "<<czas<<" s"<<endl;
    wyniki<<"\n"<<endl<<"Czas sortowania grzebieniowego: "<<czas<<" s"<<endl;

    /*testy algorytmu sortowania przez wstawianie*/
    cout<<endl<<endl<<"Sortuje przez wstawianie."<<endl;
    start = clock();
    sortowanie_przez_wstawianie (tablica2,ile);

```

```

stop = clock();

czas = (double)(stop-start) / CLOCKS_PER_SEC;

cout<<"Ilosc danych w tablicy: "<<ile<<endl;

cout<<"Czas sortowania przez wstawianie: "<<czas<<" s"<<endl;

wyniki<<endl<<"Czas sortowania przez wstawianie: "<<czas<<" s"<<endl;


//testowe wypisanie posortowanej, losowo wygenerowanej tablicy
/*  cout<<"Po posortowaniu: "<<endl;

wyniki<<"\\n"<<"Po posortowaniu: "<<endl;

for(int i=0; i<ile; i++)
{
    cout<<tablica[i]<<" "; //mozliwosc zmiany na tablica2[i] w celu sprawdzenia
poprawnosc sortowania metoda quicksort
    wyniki<<tablica[i]<<" ";
}
*/
}

}

if(x==3)
{
    int ile=100; //liczba elementow w pliku
    int tablica[ile]; //zadeklarowanie dwoch tablic
    int tablica2[ile];

    for(int i=0; i<ile; i++) //petla wpisujaca elementy z pliku do tablicy
    {
        plik>>tablica[i];
    }

    /*  cout<<"Przed posortowaniem: "<<endl;

    wyniki<<"Przed posortowaniem: "<<endl;

```

```

    for (int i=0; i < ile; i++)
    {
        cout << tablica[i] << " ";
        wyniki<<tablica[i]<<" ";
    }
*/

cout<<endl<<endl<<"Sortuje grzebieniowo."<<endl;
start = clock();
sortowanie_grzebieniowe (tablica,ile);
stop = clock();
czas = (double)(stop-start) / CLOCKS_PER_SEC;
cout<<"Ilosc danych w tablicy: "<<ile<<endl;
cout<<"Czas sortowania grzebieniowego: "<<czas<<" s"<<endl;
wyniki<<"\n"<<endl<<"Czas sortowania grzebieniowego: "<<czas<<" s"<<endl;

/*testy algorytmu sortowania przez wstawianie*/
cout<<endl<<endl<<"Sortuje przez wstawianie."<<endl;
start = clock();
sortowanie_przez_wstawianie (tablica2,ile);
stop = clock();
czas = (double)(stop-start) / CLOCKS_PER_SEC;
cout<<"Ilosc danych w tablicy: "<<ile<<endl;
cout<<"Czas sortowania przez wstawianie: "<<czas<<" s"<<endl;
wyniki<<endl<<"Czas sortowania przez wstawianie: "<<czas<<" s"<<endl<<endl;

//zapisanie do pliku
/*  cout<<"Po posortowaniu: "<<endl;
    wyniki<<"\n"<<"Po posortowaniu: "<<endl;
    for (int i=0; i < ile; i++)
    {
        cout<<tablica[i]<<" ";

```



```
        wyniki<< tablica[i] << " ";  
    }  
    */  
    plik.close();  
    wyniki.close();  
}  
return 0;  
}
```