WYDZIAŁ
MATEMATYKI
I FIZYKI STOSOWANEJ
POLITECHNIKI RZESZOWSKIEJ

Kacper Kloc

Zadanie projektowe nr 2 – Algorytmy i Struktury Danych

Rzeszów, 2021

Spis treści

1. Treść zadania

Zaimplementuj sortowanie przez wybieranie oraz sortowanie kopcowe.

1. przedstaw schematy blokowe algorytmów oraz pseudokod odpowiadający obu schematom
2. przedstaw teoretyczne podstawy obu metod
3. wykonaj testy porównujące działanie obu metod na różnych próbkach danych i przedstaw ich wyniki w sprawozdaniu
4. omów złożoność obliczeniową obu algorytmów
5. przedstaw w postaci wykresów t(N) złożoność czasową obu algorytmów dla przypadków oczekiwanego/ optymistycznego/ pesymistycznego (”odpowiednio preparując” dane do posortowania dla każdego z algorytmów) otrzymaną eksperymentalnie w wyniku serii testów dla rosnących próbek danych N.

2. Opis działania programu

W programie znajduje się napisanych 14 funkcji, wliczając w to funkcję główną, zwaną main'em. Idąc od góry są to kolejno:

1. Funkcja **losuj**, generująca pseudolosową zawartość tablicy. Pobiera ona argument N – długość tablicy, którą można ręcznie przypisać w funkcji main.
2. Funkcja **wczytanie**, wykorzystująca zmienną typu char do odczytania nazwy pliku, z którego chcemy załadować nasze dane do tablicy.

   Obie powyższe funkcje są typu std::vector<int>, ponieważ w obrębie programu posłużyłem się tym właśnie typem struktury danych.

3. Funkcja o nazwie **zapisz_do_pliku**. Jak sama nazwa wskazuje, zapisuje ona nasze dane do pliku tekstowego, którego nazwa jest argumentem tej funkcji.
4. Funkcja pomocnicza do funkcji sortującej kopcowo tablicę. W niej zachodzi właściwe sortowanie dla tej metody.
5. Główna funkcja sortowania kopcowego. W niej tworzony jest kopiec, poprzez reorganizowanie sortowanych elementów oraz w jej obrębie wywoływana jest funkcja pomocnicza.
6. Funkcja sortowania przez wybór – druga wymagana w zadaniu metoda sortowania. Polega na poszukiwaniu elementu, który ma znaleźć się na żądanej pozycji i zamianie miejscami z elementem obecnie znajdującym się w tym miejscu.
7. Funkcja sortowania przez wybór malejąco – stworzona została na potrzeby przeprowadzanych testów algorytmów.

   Zaczynając od ósmej funkcji, wchodzimy w obszar testów algorytmów.
   Pierwsze trzy z sześciu funkcji testowych operują na algorytmie sortowania kopcowego. Kolejno odbywają się testy dla oczekiwanego przypadku (funkcja nr 8), dla przypadku pesymistycznego(nr 9) oraz optymistycznego(nr 10).
   W tej samej kolejności odbywają się testy dla sortowania przez wybór – przypadek oczywisty to funkcja nr 11, pesymistyczny – 12 oraz optymistyczny - nr 13.

   Jako przypadek typowy/oczekiwany użyłem ciągu liczb wygenerowanego za pomocą funkcji **losuj**. Jest to zwykły, nie poukładany ciąg liczbowy.
   Przypadek pesymistyczny stanowi ciąg liczbowy posortowany malejąco. Tworzony on jest również za pomocą funkcji **losuj**, jednak wygenerowany za jej pomocą ciąg jest później sortowany malejąco (funkcja nr 7). Z racji, że nasze sortowanie odbywa się w sposób rosnący, ciąg malejący wymaga największej ilości obliczeń od naszych algorytmów.
   Ostatni z przypadków to przypadek optymistyczny. Tutaj mamy do czynienia z ciągiem, który najpierw losujemy, później sortujemy i mieszamy w nim tylko kilka elementów. Wówczas zaczyna się nasze sortowanie właściwe.

```
int main()
{
    std::vector<int>tab;
    tab = wczytanie("Dane.txt");
    int N=tab.size();
    sortowanie_przez_wybor_malejaco(tab, N);
    zapisz_do_pliku("zapis.txt", tab, N);

    testy_typ_kopcowe();
    testy_typ_wybor();
    testy_opt_kopcowe();
    testy_opt_wybor();
    testy_pes_kopcowe();
    testy_pes_wybor();


    return 0;
}
```

Rysunek 1 Funkcja main

W głównej części naszego programu znajdziemy niewiele kodu – zwykłe utworzenie zmiennych, wczytanie tablicy z pliku, posortowanie jej i zapisanie do pliku oraz wywołanie naszych funkcji, testujących algorytmy sortowania.

3. Sortowanie kopcowe

3.1. Teoretyczne podstawy

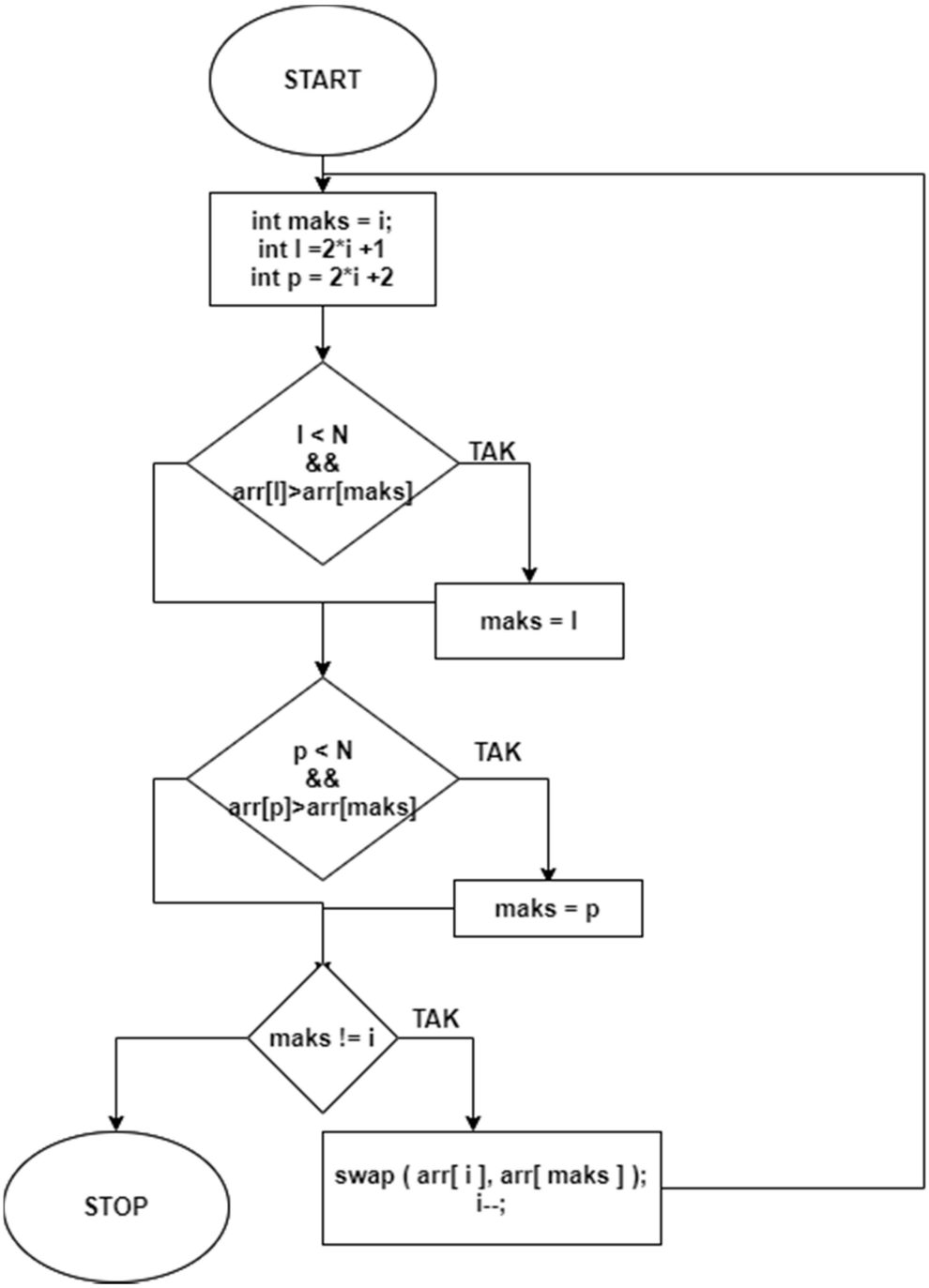
Sortowanie kopcowe składa się z dwóch faz. W pierwszej sortowane elementy reorganizowane są w celu utworzenia kopca, a w drugiej dokonywane jest właściwe sortowanie. Pomimo, że jest niestabilny, to jest szybki i nie pochłania wiele pamięci.

Złożoność czasowa sortowania kopcowego wynosi O(n log(n)).

Zaletą tego algorytmu jest to, że do stworzenia kopca można wykorzystać tę samą tablicę, w której początkowo znajdują się nieposortowane elementy. Takie rozwiązanie zapewnia nam **stałą złożoność obliczeniową O(n)**.

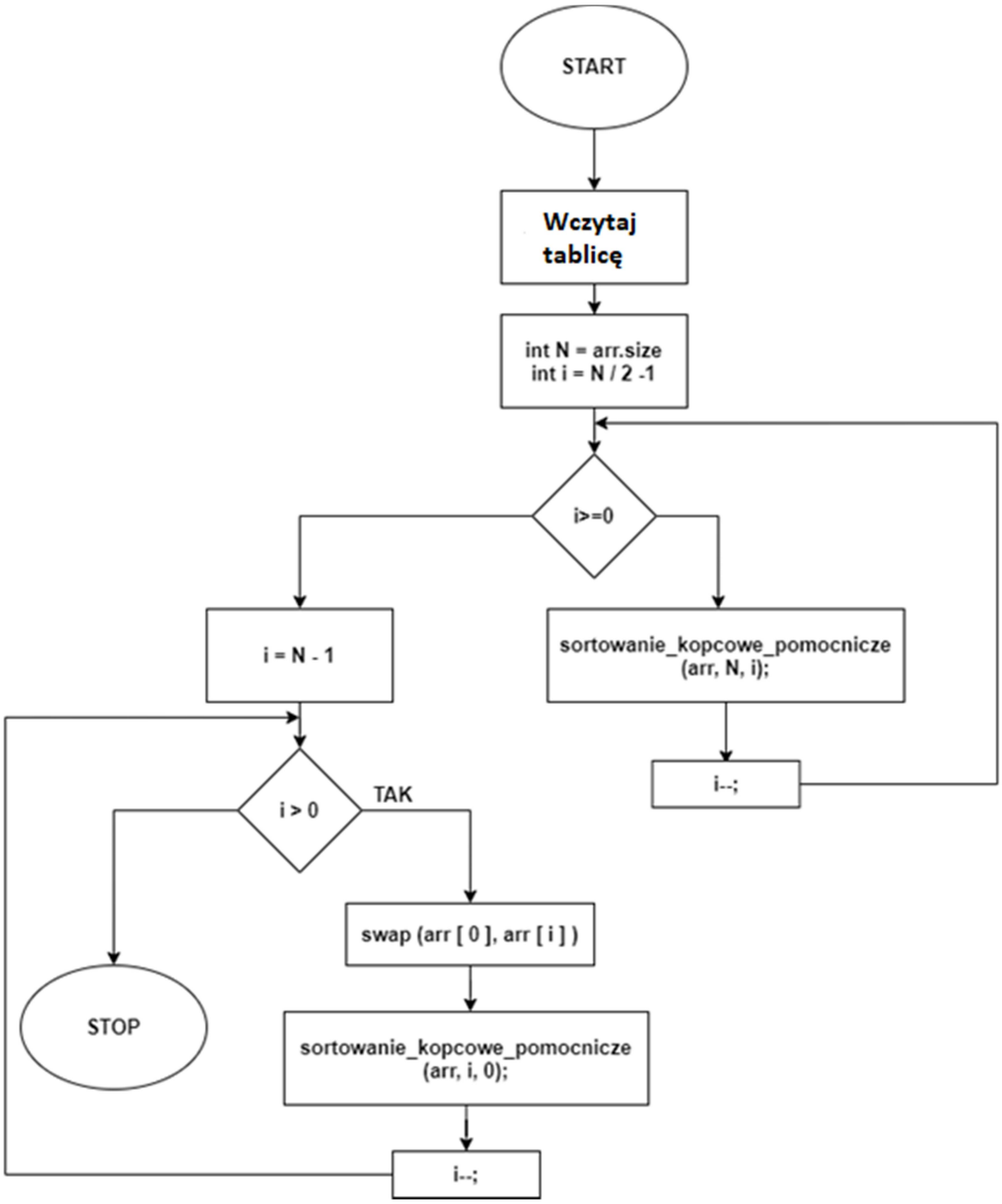
W praktyce sortowanie kopcowe okazuje się nieco wolniejsze od sortowania szybkiego, jednak ma lepszą pesymistyczną złożoność czasową.

3.2. Schemat blokowy.



Rysunek 2 Schemat blokowy sortowania kopcowego cz. I

Powyższy schemat przedstawia część pomocniczą algorytmu.



Rysunek 3 Schemat blokowy sortowania kopcowego cz. II

Powyżej przedstawiona jest część główna algorytmu (wraz z odniesieniami do części pomocniczej).

Schematy podzieliłem, aby były przejrzyste i czytelne, złączenie ich w jeden zajęłoby zbyt dużo miejsca.

3.3. Pseudokod.

Część pomocnicza (sortowanie_kopcowe_pomocnicze):

Start

int maks=i

int l = i*2 +1

int p = i*2 +2

Jeśli l < N oraz arr [l] > arr [maks]

    maks = l

W przeciwnym razie idź dalej

Jeśli p < N oraz arr [p] > arr [maks]

    maks = p

W przeciwnym razie idź dalej

Jeśli maks != i

    swap (arr [i], arr [maks])

    i=i-1

W przeciwnym razie

Koniec

Część główna:

Start

    Załaduj/Wylosuj tablicę (arr)

int N=arr.size

    Dla i=N/2 -1; i>=0; i--:

        Wykonaj sortowanie_kopcowe_pomocnicze (arr, N, i)

    Dla i=N-1; i>0; i--:

        Wykonaj swap(arr [0], arr [i])

        Wykonaj sortowanie_kopcowe_pomocnicze (arr, i, 0)

Koniec

3.4. Wykresy

W ramach przeprowadzonych ośmiu prób dla algorytmu operowałem na tablicach o długościach:

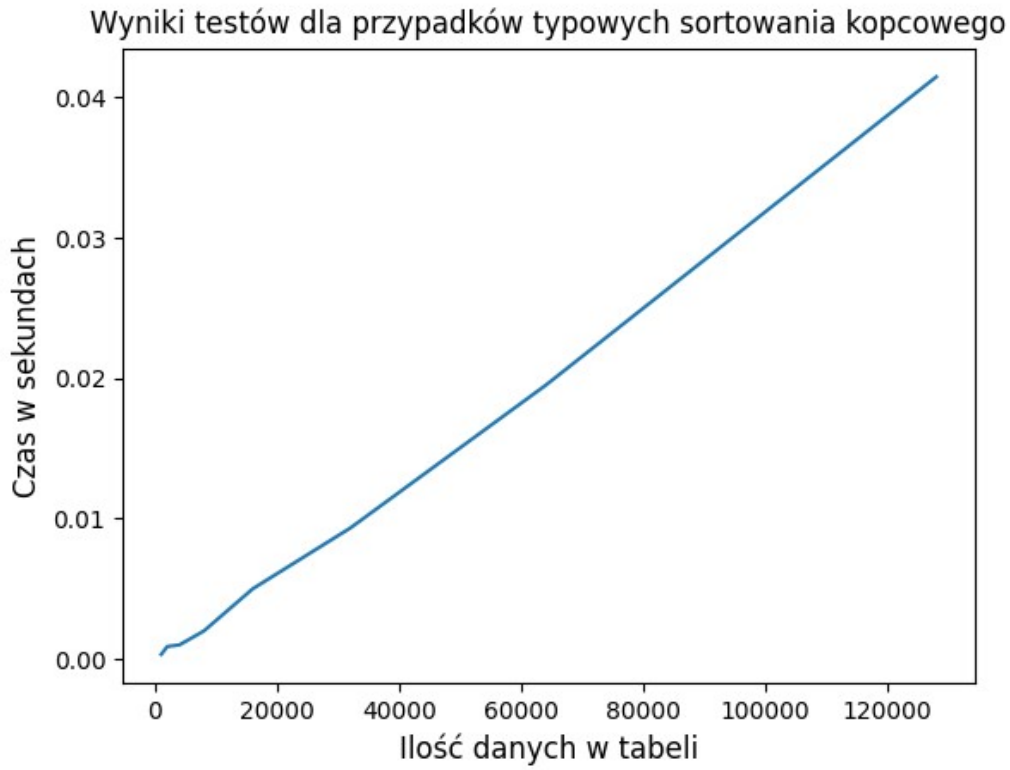
1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000 liczb.

Dla przypadku oczywistego/typowego użyłem tablic wylosowanych przy pomocy generatora liczb „losowych”.

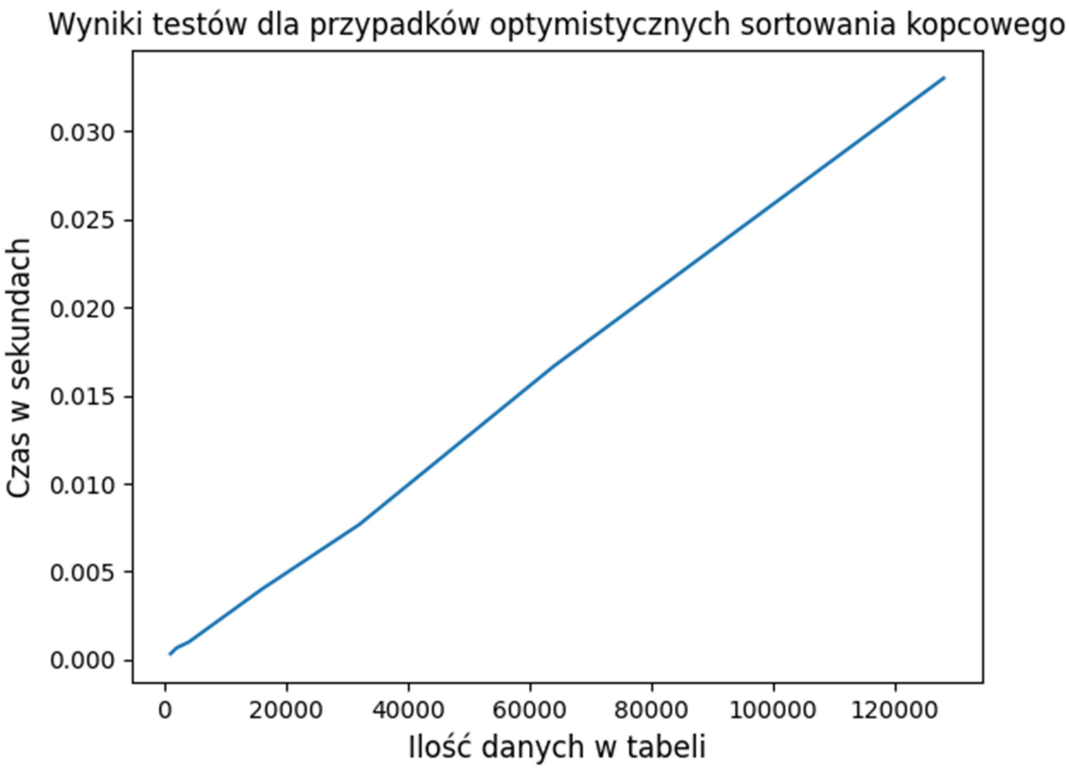
Jako przypadek optymistyczny wykorzystałem tablice już posortowaną, ze zmienionymi kilkoma (konkretnie ośmioma) elementami.

Przypadek pesymistyczny stanowiły tablice wcześniej odwrotnie posortowane, tj. malejąco.

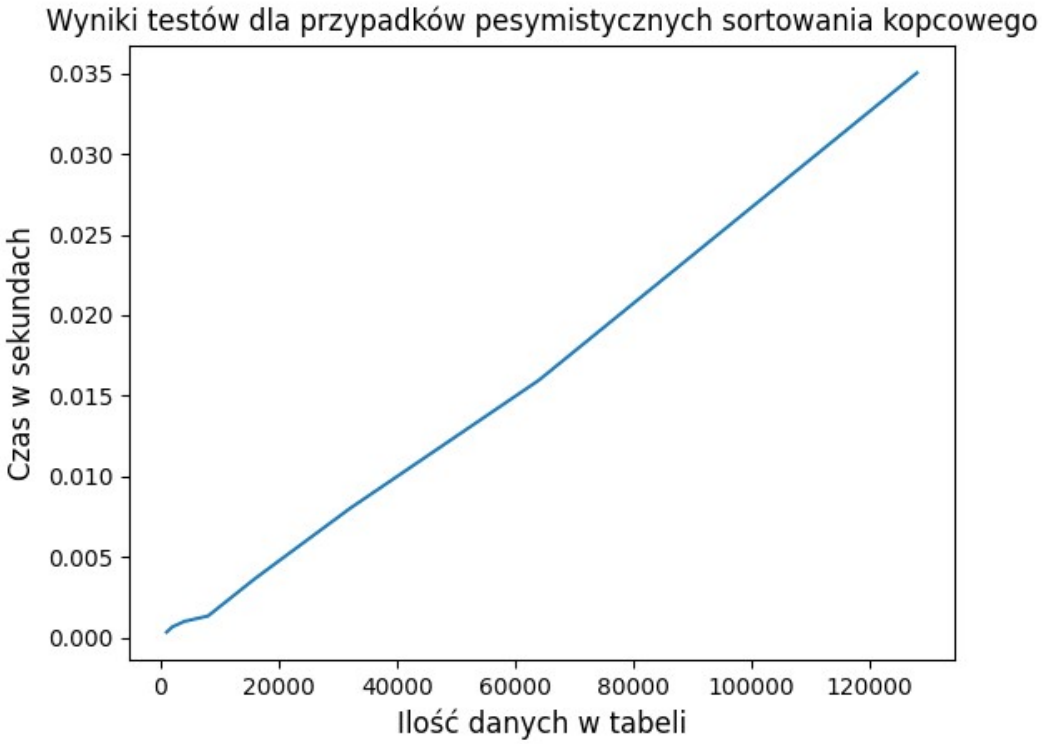
Wykres czwarty dla tego algorytmu przedstawia zestawienie uzyskanych przeze mnie wyników w postaci jednego wykresu, w ramach porównania czasów dla poszczególnych przypadków.



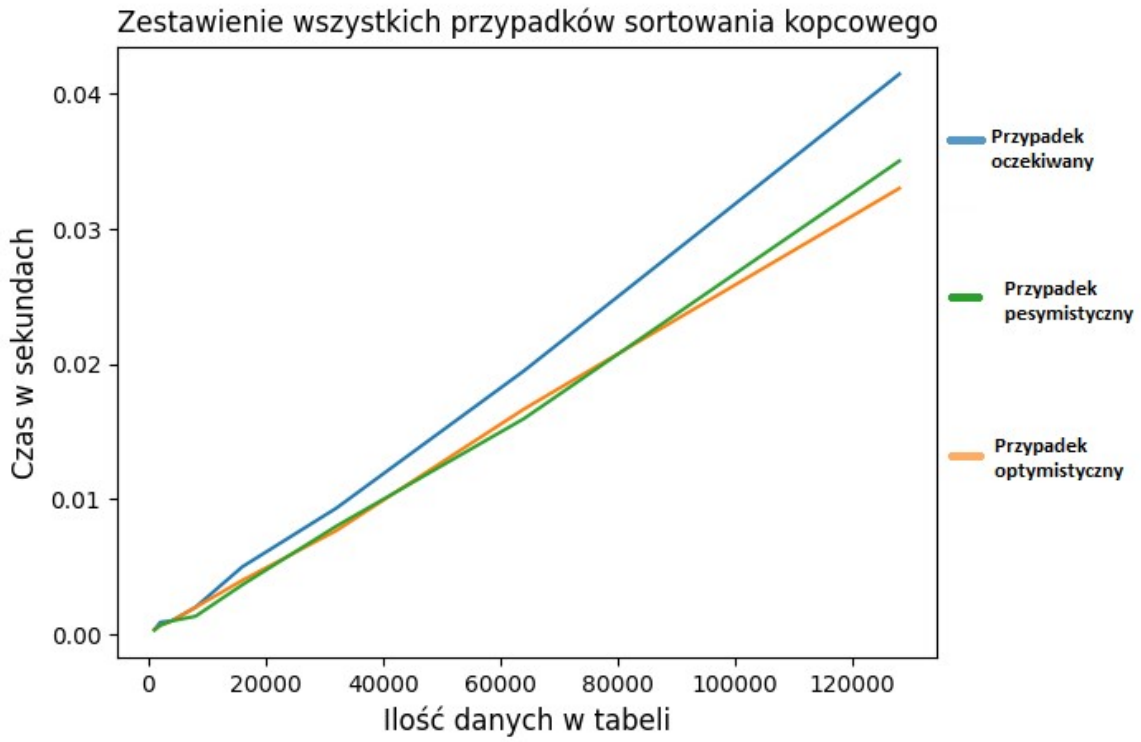
Rysunek 4 Wykres algorytmu sortowania kopcowego 1



Rysunek 5 Wykres algorytmu sortowania kopcowego 2



Rysunek 6 Wykres algorytmu sortowania kopcowego 3



Rysunek 7 Wykres algorytmu sortowania kopcowego 4

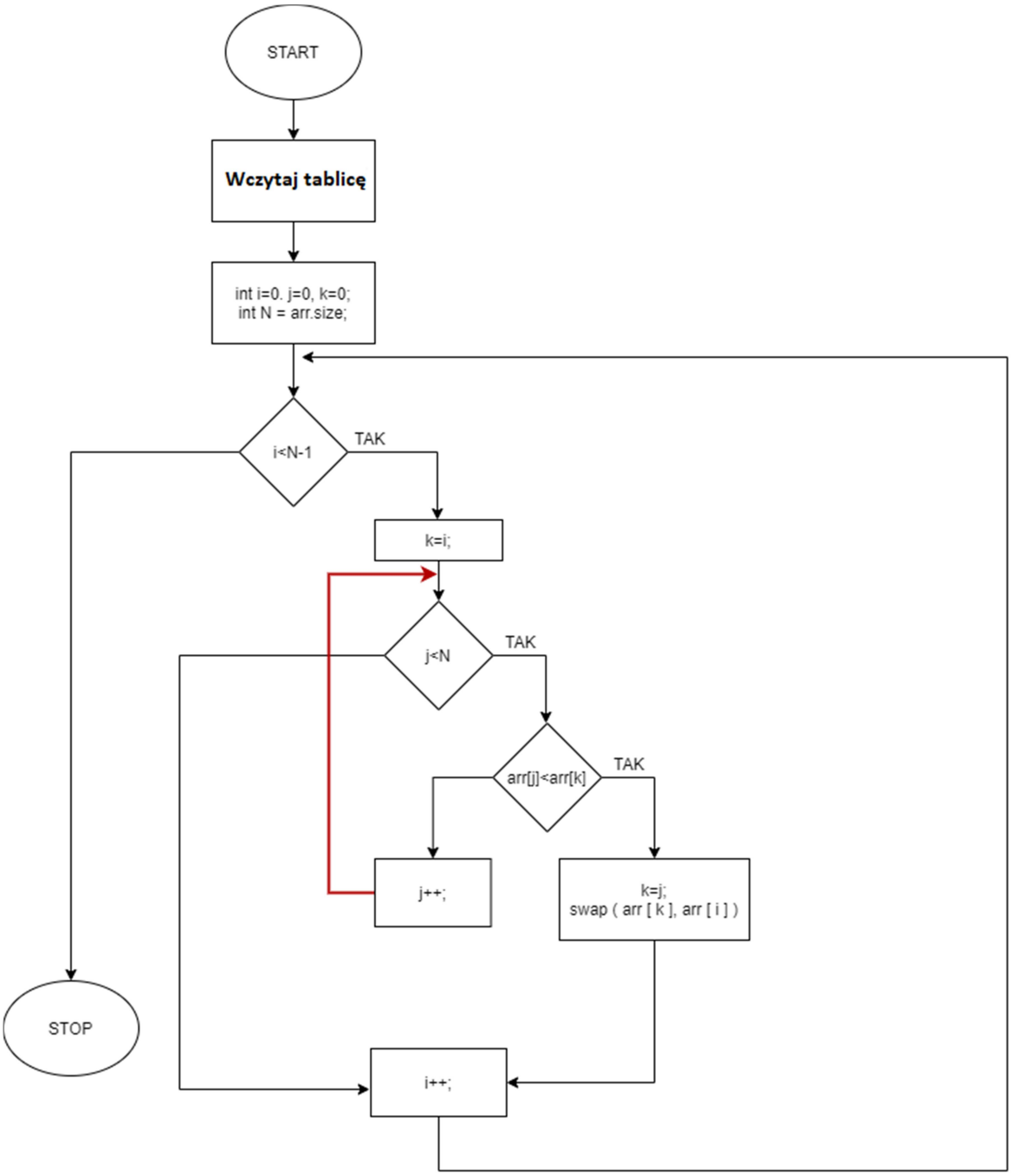
4. Sortowanie przez wybór

4.1. Teoretyczne podstawy

Sortowanie przez wybieranie jest jedną z prostszych metod sortowania. **Jej złożoność czasowa/obliczeniowa wynosi O(n^2)**. Polega na wyszukiwaniu elementu, który ma znaleźć się na żądanej pozycji i zamianie miejscami z tym, który tam obecnie jest. Czynność ta jest powtarzana dla wszystkich indeksów tablicy.

Algorytm jest niestabilny oraz jak się okazuje bardzo wolny. Nie radzi sobie dobrze z dużymi tablicami.

4.2. Schemat blokowy



Rysunek 8 Schemat blokowy algorytmu sortowania przez wybór

4.3. Pseudokod

```
Start

int i, j, k

int N = arr.size

Dla i = 0;  i < N-1; i++:

        k = i

        Dla j=0; j < N; j++:

                Jeśli arr [ j ] < arr [ k ]

                        k = j

        swap ( arr [ k ], arr [ i ] )

Koniec
```

4.4. Wykresy

W ramach przeprowadzonych ośmiu prób dla algorytmu operowałem na tablicach o długościach:

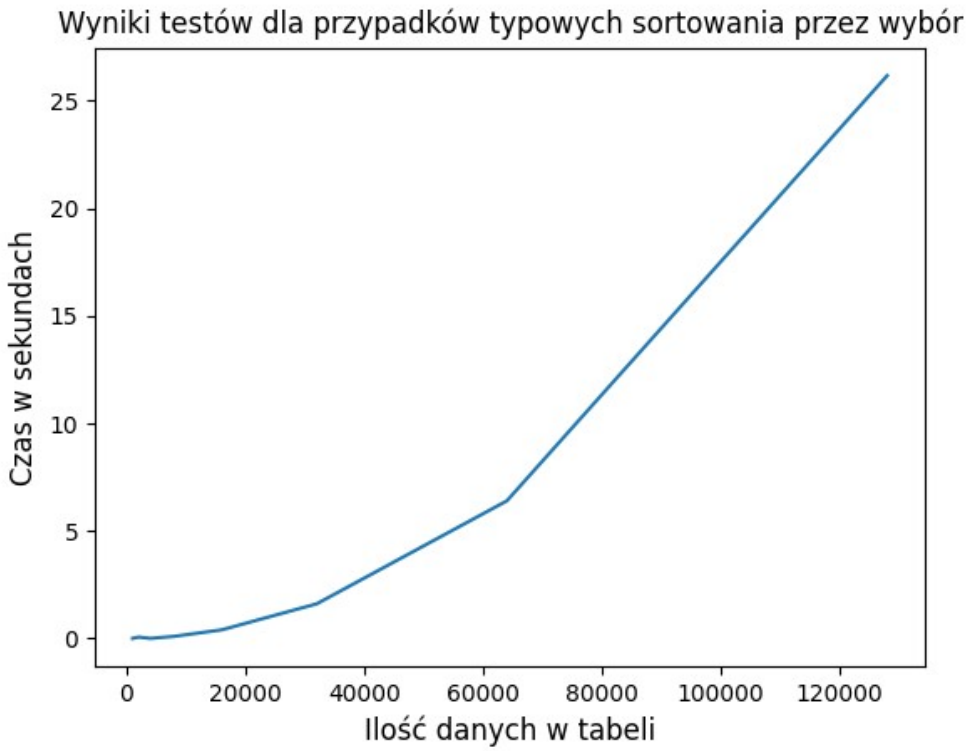
1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000 liczb.

Dla przypadku oczywistego/typowego użyłem tablic wylosowanych przy pomocy generatora liczb „losowych”.

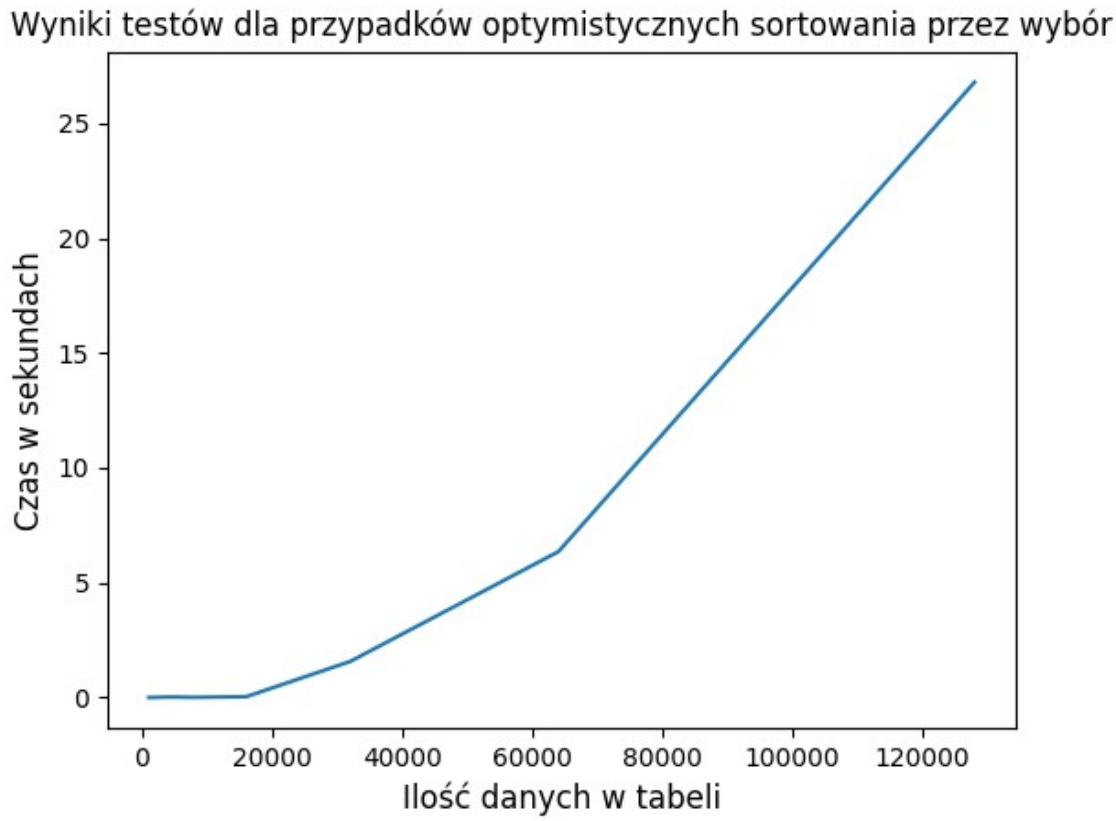
Jako przypadek optymistyczny wykorzystałem tablice już posortowaną, ze zmienionymi kilkoma (konkretnie ośmioma) elementami.

Przypadek pesymistyczny stanowiły tablice wcześniej odwrotnie posortowane, tj. malejąco.

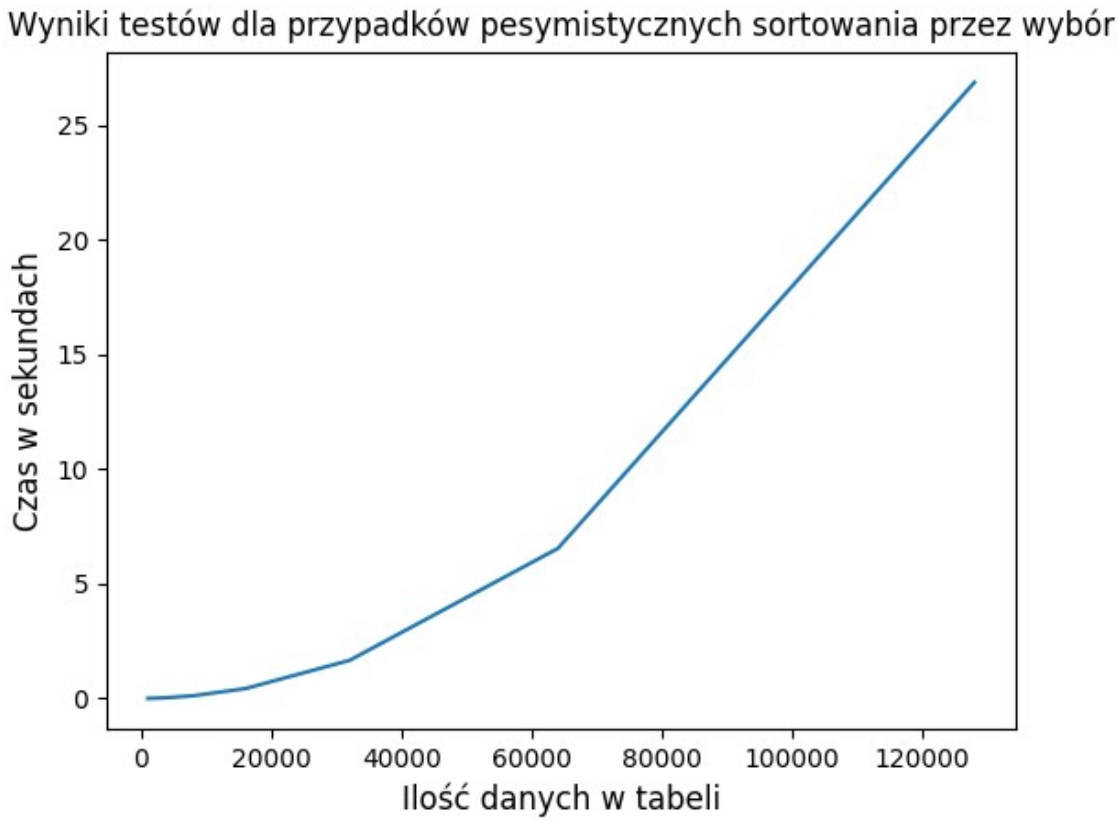
Wykres czwarty dla tego algorytmu przedstawia zestawienie uzyskanych przeze mnie wyników w postaci jednego wykresu, w ramach porównania czasów dla poszczególnych przypadków.



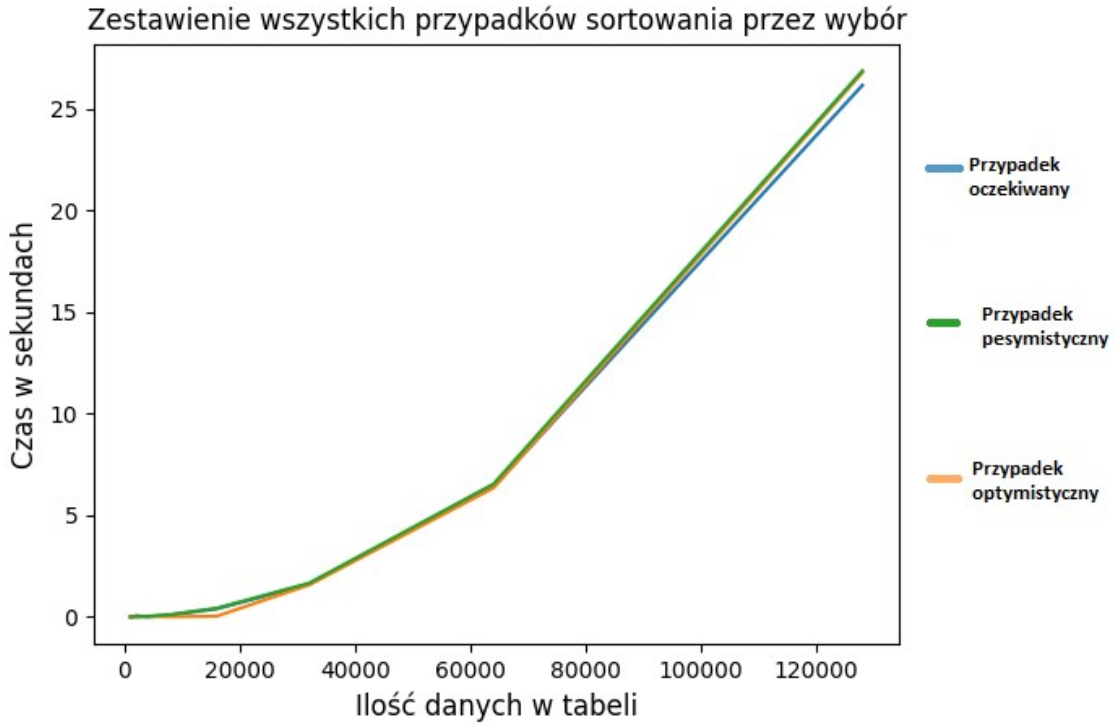
Rysunek 9 Wykres algorytmu sortowania przez wybór 1



Rysunek 10 Wykres algorytmu sortowania przez wybór 2



Rysunek 11 Wykres algorytmu sortowania przez wybór 3



Rysunek 12 Wykres algorytmu sortowania przez wybór 4

5. Kod programu

```
#include <iostream>
#include <fstream>
#include <iterator>
#include <iomanip>
#include <limits.h>
#include <cstdlib>
#include <time.h>
#include <vector>
#include <chrono>
using namespace std::chrono;
using namespace std;

// Funkcja generująca losową zawartość tablicy o długości N
std::vector<int> losuj(int N)
{
   std::vector<int>arr;
   int i, a;
   for(i=0; i<N; i++)
   {
      a = rand ( ) % 1000;
      arr.push_back(a);
   }
   return arr;
}

// Funkcja wczytania danych z pliku
std::vector<int> wczytanie(char *nazwa_pliku)
{
```

```
    std::vector<int> arr;
    ifstream plik(nazwa_pliku);        // Wczytywany plik

    // Sprawdzenie czy plik istnieje i otworzenie go
    if (plik.good())
    {
        // Wczytanie danych do tablicy
        int x = 0;
        while (plik >> x)
        {
            arr.push_back(x);
        }
        // Zamknięcie pliku.
        plik.close();
    }
    else
    {
        cout << "Error!";
        _exit(0);
    }
    return arr;
}

// Funkcja zapisu do pliku
void zapisz_do_pliku(char *nazwa_pliku, std::vector<int> arr, int arrLength)
{
    fstream plik;
    plik.open(nazwa_pliku, ios::out|ios::app);
    for (int i = 0; i < arrLength; i++)
    {
```

```
        plik<<arr[i]<<" ";
    }
    plik<<endl<<endl;
    plik.close();
}

// Funkcja pomocnicza do sortowania kopcowego
void sortowanie_kopcowe_pomocnicze(std::vector<int>&arr, int &N, int i)
{
    int maks = i; // przypisanie najwiekszej wartosci do zmiennej
    int l = 2 * i + 1; // lewy = 2*i + 1
    int p = 2 * i + 2; // prawy = 2*i + 2

    // Jeśli lewe dziecko jest większe
    if (l < N && arr[l] > arr[maks])
        maks = l;

    // Jeśli prawe dziecko jest większe
    if (p < N && arr[p] > arr[maks])
        maks = p;

    //
    if (maks != i)
    {
        swap(arr[i], arr[maks]);

        // Rekursyjne powtarzanie funkcji
        sortowanie_kopcowe_pomocnicze(arr, N, maks);
    }
}
```

```
// Główna funkcja sortowania kopcowego
void sortowanie_kopcowe(std::vector<int>&arr, int &N)
{
   // Budowa kopca
   for (int i = N / 2 - 1; i >= 0; i--)
      sortowanie_kopcowe_pomocnicze(arr, N, i);

   for (int i = N - 1; i > 0; i--)
   {
      // Przeniesienie korzenia na koniec
      swap(arr[0], arr[i]);

      // Funkcja pomocnicza do sortowania
      sortowanie_kopcowe_pomocnicze(arr, i, 0);
   }
}

// Funkcja sortowania przez wybór
void sortowanie_przez_wybor(std::vector<int>&arr, int &N)
{
   int i, j, k;

   for (i = 0; i < N-1; i++)
   {
      // Szukanie najmniejszej wartosci w nieposortowanej tablicy
      k = i;
      for (j = i+1; j < N; j++)
         if (arr[j] < arr[k])
            k = j;
```

```
        // Swap znalezionego najmniejszego elementu z pierwszym elementem
        swap(arr[k], arr[i]);
    }
}
// Funkcja sortująca malejąco - użyta przy teście pesymistycznym algorytmów
void sortowanie_przez_wybor_malejaco(std::vector<int>&arr, int &n)
{
    int i, j, k;
    for (i = 0; i < n-1; i++)
    {
        // Szukanie najmniejszej wartosci w nieposortowanej tablicy
        k = i;
        for (j = i+1; j < n; j++)
            if (arr[j] > arr[k])
                k = j;

        // Swap znalezionego najmniejszego elementu z pierwszym elementem
        swap(arr[k], arr[i]);
    }
}
// Funkcja wykonująca testy przypadku oczekiwanego dla sortowania kopcowego
void testy_typ_kopcowe()
{
    int i, N=1000;
    std::vector<int>tab;
    int liczba_testow = 8;
    cout<<"Testy sortowania przez kopcowanie dla przypadku oczekiwanego:"<<endl;
    for(i=0; i<liczba_testow; i++)
    {
```

```
        // Generowanie zawartości tablicy
        tab=losuj(N);
        // Pomiar czasu
        auto start = high_resolution_clock::now();
        sortowanie_kopcowe(tab, N);
        auto stop = high_resolution_clock::now();
        std::chrono::duration<double> czas = stop-start;
        cout<<"Czas dla "<<N<<" liczb w tablicy: ";
        std::cout << std::setw(9) << czas.count() << " s."<< endl;
        N = N * 2 ;
    }
}
// Funkcja wykonująca testy przypadku pesymistycznego dla sortowania kopcowego
void testy_pes_kopcowe()
{
    int i, N=1000;
    std::vector<int>tab;
    int liczba_testow = 8;
    cout<<endl<<"Testy sortowania przez kopcowanie dla przypadku pesymistycznego: "<<endl;
//Przypadek pesymistyczny - sortowanie tablicy posortowanej odwrotnie
    for(i=0; i<liczba_testow; i++)
    {
        // Generowanie zawartości tablicy
        tab=losuj(N);
        sortowanie_przez_wybor_malejaco(tab, N);
        // Pomiar czasu
        auto start = high_resolution_clock::now();
        sortowanie_kopcowe(tab, N);
        auto stop = high_resolution_clock::now();
        std::chrono::duration<double> czas = stop-start;
```

```
        cout<<"Czas dla "<<N<<" liczb w tablicy: ";
        std::cout << std::setw(9) << czas.count() << " s."<< endl;
        N = N * 2 ;
    }
}
// Funkcja wykonująca testy przypadku optymistycznego dla sortowania kopcowego
void testy_opt_kopcowe()
{
    int i,j, N=1000;
    std::vector<int>tab;
    int liczba_testow = 8, a=100;
    cout<<"Testy sortowania przez kopcowanie dla przypadku optymistycznego: "<<endl;
//Przypadek optymistyczny - tablica uporządkowana z niewielką liczbą elementów nie na swoich miejscach
    for(i=0; i<liczba_testow; i++)
    {
        // Generowanie zawartości tablicy
        tab=losuj(N);
        sortowanie_kopcowe(tab, N);
        for(j=0; j<liczba_testow; j++)
        {
            swap(tab[j], tab[a]);
            a = a + 100;
        }
        // Pomiar czasu
        auto start = high_resolution_clock::now();
        sortowanie_kopcowe(tab, N);
        auto stop = high_resolution_clock::now();
        std::chrono::duration<double> czas = stop-start;
        cout<<"Czas dla "<<N<<" liczb w tablicy: ";
```

```
        std::cout << std::setw(9) << czas.count() << " s."<< endl;
        N = N * 2;
    }
}
// Funkcja wykonująca testy przypadku oczekiwanego dla sortowania przez wybór
void testy_typ_wybor()
{
    int i, N=1000;
    std::vector<int>tab;
    int liczba_testow = 8;
    cout<<"Testy sortowania przez wybor dla przypadku oczekiwanego:"<<endl;
    for(i=0; i<liczba_testow; i++)
    {
        // Generowanie zawartości tablicy
        tab=losuj(N);
        // Pomiar czasu
        auto start = high_resolution_clock::now();
        sortowanie_przez_wybor(tab, N);
        auto stop = high_resolution_clock::now();
        std::chrono::duration<double> czas = stop-start;
        cout<<"Czas dla "<<N<<" liczb w tablicy: ";
        std::cout << std::setw(9) << czas.count() << " s."<< endl;
        // zwiekszenie rozmiarow tablicy
        N = N * 2 ;
    }
}
// Funkcja wykonująca testy przypadku pesymistycznego dla sortowania przez wybór
void testy_pes_wybor()
{
    int i, N=1000;
```

```
    std::vector<int>tab;
    int liczba_testow = 8;
    cout<<"Testy sortowania przez kopcowanie dla przypadku pesymistycznego: "<<endl;
//Przypadek pesymistyczny - sortowanie tablicy posortowanej odwrotnie
    for(i=0; i<liczba_testow; i++)
    {
        // Generowanie zawartości tablicy
        tab=losuj(N);
        sortowanie_przez_wybor_malejaco(tab, N);
        // Pomiar czasu
        auto start = high_resolution_clock::now();
        sortowanie_przez_wybor(tab, N);
        auto stop = high_resolution_clock::now();
        std::chrono::duration<double> czas = stop-start;
        cout<<"Czas dla "<<N<<" liczb w tablicy: ";
        std::cout << std::setw(9) << czas.count() << " s."<< endl;
        // zwiekszenie rozmiarow tablicy
        N = N * 2 ;
    }
}
// Funkcja wykonująca testy przypadku optymistycznego dla sortowania przez wybór
void testy_opt_wybor()
{
    int i,j, N=1000;
    std::vector<int>tab;
    int liczba_testow = 8, a=100;
    cout<<"Testy sortowania przez wybor dla przypadku optymistycznego: "<<endl; //Przypadek
optymistyczny - tablica uporządkowana z niewielką liczbą elementów nie na swoich miejscach
    for(i=0; i<liczba_testow; i++)
    {
```

```
        // Generowanie zawartości tablicy
        tab=losuj(N);
        sortowanie_kopcowe(tab, N);
        for(j=0; j<liczba_testow; j++)
        {
           swap(tab[j], tab[a]);
           a = a + 100;
        }
        // Pomiar czasu
        auto start = high_resolution_clock::now();
        sortowanie_przez_wybor(tab, N);
        auto stop = high_resolution_clock::now();
        std::chrono::duration<double> czas = stop-start;
        cout<<"Czas dla "<<N<<" liczb w tablicy: ";
        std::cout << std::setw(9) << czas.count() << " s."<< endl;
        // zwiekszenie rozmiarow tablicy
        N = N * 2 ;
    }
}

int main()
{
    std::vector<int>tab;
    tab = wczytanie("Dane.txt");
    int N=tab.size();
    sortowanie_przez_wybor_malejaco(tab, N);
    zapisz_do_pliku("zapis.txt", tab, N);

    testy_typ_kopcowe();
    testy_typ_wybor();
```

```
    testy_opt_kopcowe();
    testy_opt_wybor();
    testy_pes_kopcowe();
    testy_pes_wybor();



    return 0;
}
```