Sprawozdanie z projektu

Klaudia Zawiślak

nr indeksu 166719

Algorytmy i struktury danych

Inżynieria i Analiza Danych

Semestr 1, Grupa nr 8

1.Wstęp.

Niniejsze sprawozdanie przedstawia wykonanie projektu z przedmiotu „Algorytmy i struktury danych”. Zawiera w sobie opis, dokładną treść zadania oraz sposób w jaki je wykonałam. Posiada także pseudokod oraz schemat blokowy. Zamieściłam przykładowe testy programu i moje wnioski, opisałam także kilka funkcji z których tworzeniem zmagałam się nieco dłużej. Celem projektu było dokonanie implementacji kodu, który rozwiązuje zadanie. Implementacja jest to proces pisania programu (kodu źródłowego), czyli programowanie, lub efekt takiego procesu, czyli program. Kod został napisany w programie Codeblocks, jest to wieloplatformowe, zintegrowane środowisko programistyczne na licencji GNU, oparte na projekcie Scintilla. Program jest napisany w języku C++.

2.Treść zadania do wykonania.

Zaimplementuj sortowanie przez scalanie 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

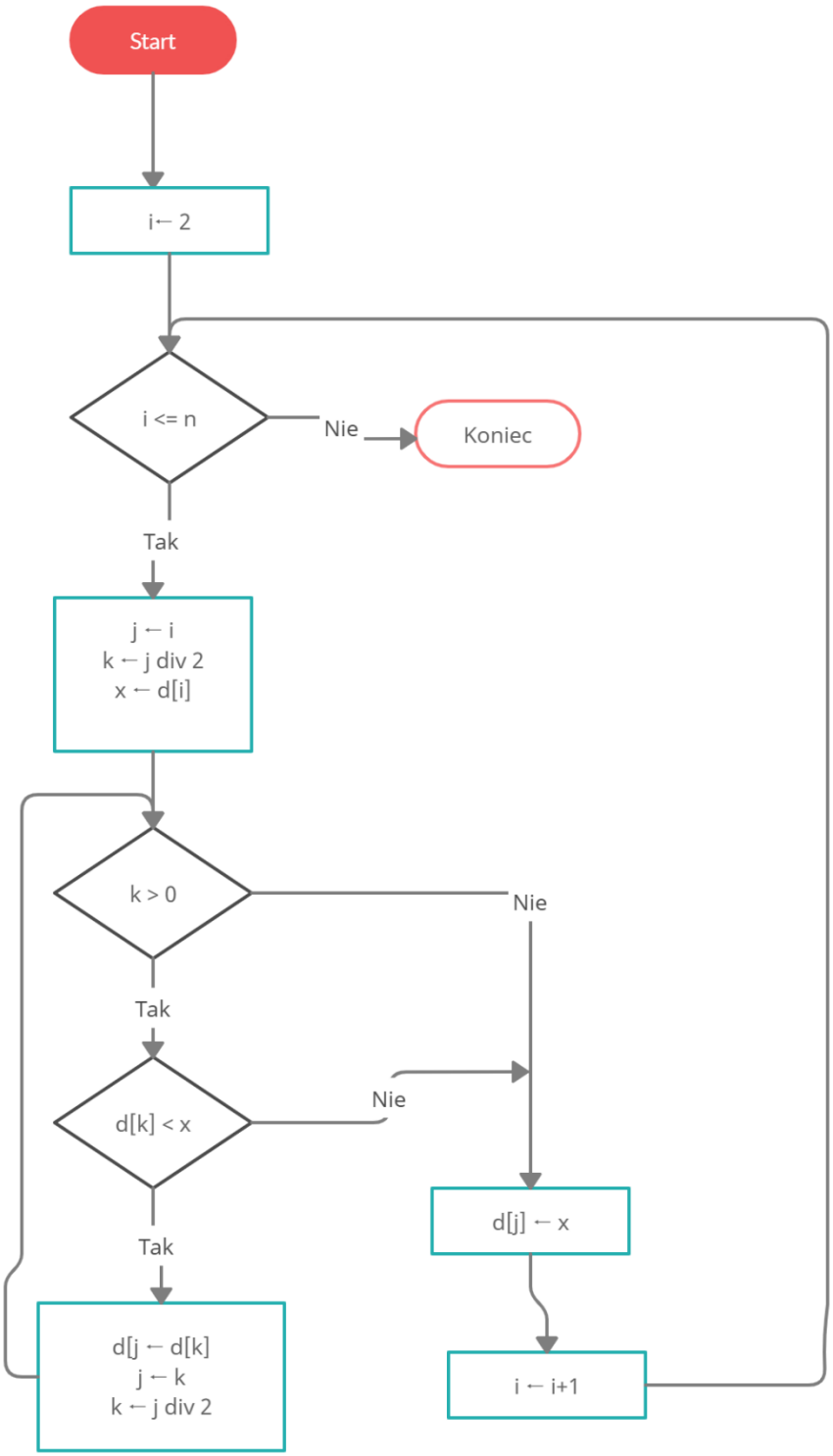
4) omów złożoność obliczeniową obu algorytmów

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

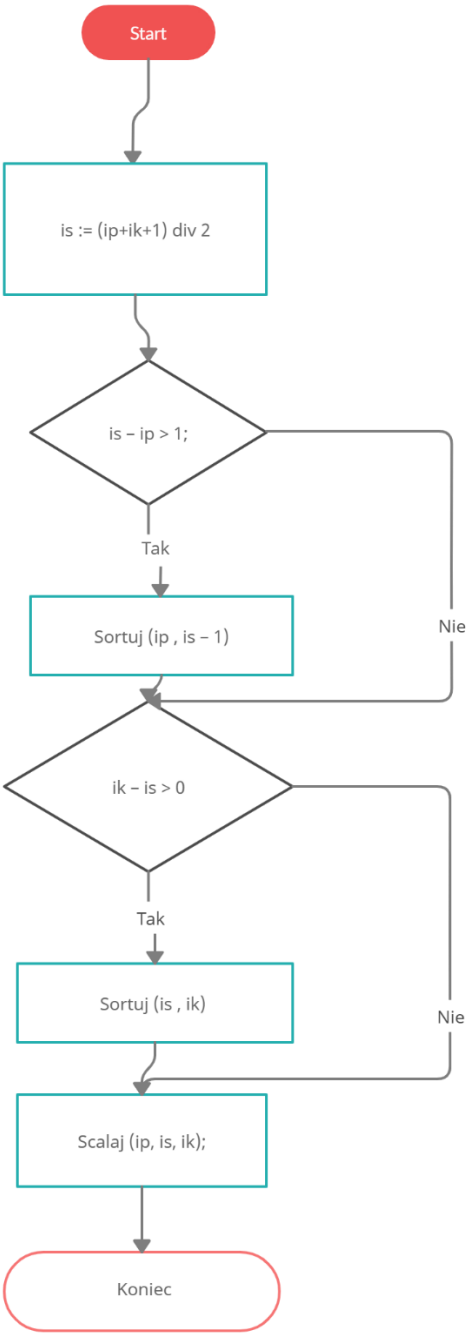
3. Schemat blokowy algorytmu.

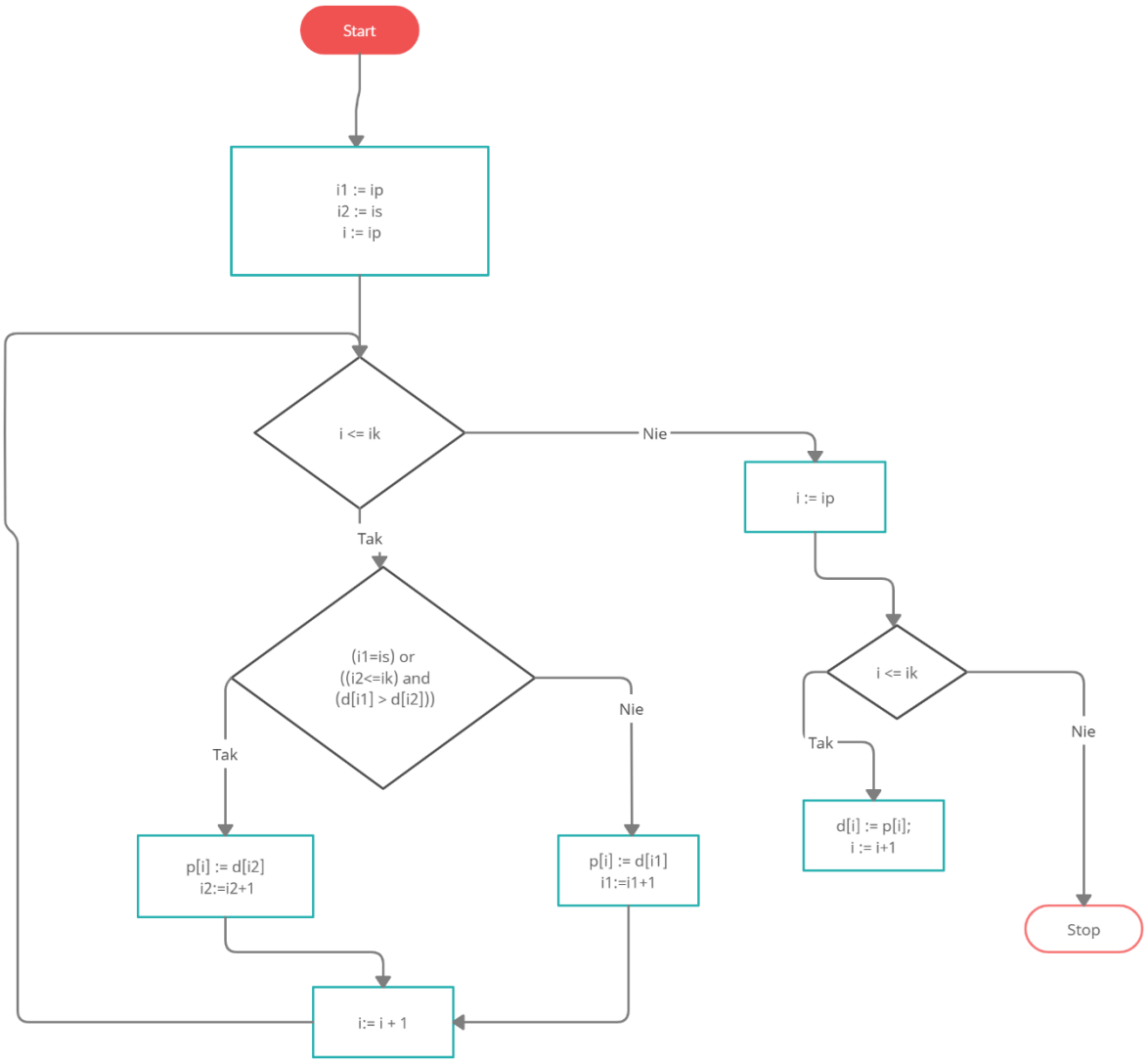
Schemat blokowy to narzędzie służące do przedstawienia kolejnych czynności w projektowanym algorytmie. Jest to diagram, na którym procedura, system lub program komputerowy są reprezentowane przez opisane figury geometryczne połączone strzałkami zgodnie z kolejnością wykonywania czynności wynikających z przyjętego algorytmu rozwiązania zadania.

Kopcowe


Start
i← 2
i <= n
Nie
Koniec
Tak
j ← i
k ← j div 2
x ← d[i]
k > 0
Nie
Tak
d[k] < x
Nie
Tak
d[j] ← x
d[j ← d[k]
j ← k
k ← j div 2
i ← i+1

Scalanie


Start
is := (ip+ik+1) div 2
is – ip > 1;
Tak
Sortuj (ip , is – 1)
Nie
ik – is > 0
Tak
Sortuj (is , ik)
Nie
Scalaj (ip, is, ik);
Koniec



4.Pseudokod.

Pseudokodem nazywany jest taki sposób zapisu algorytmu, który zachowując strukturę charakterystyczną dla kodu zapisanego w języku programowania, rezygnuje ze ścisłych reguł składniowych na rzecz prostoty i czytelności. Pseudokod nie zawiera szczegółów implementacyjnych często też opuszcza się w nim opis działania podprocedur (jeśli powinien być on oczywisty dla czytelnika), zaś nietrywialne kroki algorytmu opisywane są z pomocą formuł matematycznych lub zdań w języku naturalnym.

Dla sortowania przez scalanie:

Funkcja scal(int* tab, int from, int sr, int to) {

```
  Dla  (int i = from; i <= to; i++)
    tab_pom[i] = tab[i];
  int i = from, j = sr + 1;
  dla (int k = from; k <= to; k++) {
    jeśli (i <= sr) {
      jeśli (j <= to) {
        tab[k] = (tab_pom[j] < tab_pom[i]) ? tab_pom[j++] : tab_pom[i++];
      } w innym wypadku {
        tab[k] = tab_pom[i++];
      }
    } w innym wypadku {
      tab[k] = tab_pom[j++];
    }
  }
}
funkcja sortowaniescalanie(int* tab, int from, int to) {
  jeśli(to <= from)
    return;
  int sr = (to + from) / 2;
  sortowaniescalanie(tab, from, sr);
  sortowaniescalanie(tab, sr + 1, to);
  scal(tab, from, sr, to);
}
```

Dla sortowania kopcowego:

```
Funkcja wstaw(int tab[],int n)  //funkcja zape³niaj¹ca tablicê
{
```

```
    dla(int i=0; i<n; i++)
        tab[i]=rand()%1000;
}

funkcja drukuj(int tab[],int n)
{
    dla(int i =0; i<n; i++)
        wyświetl<<tab[i]<<" ";
    cout<<endl;
} //funkcja do wyœwietlenia tablicy

funkcja zanurzanie(int i, int kolejka[], int rozmiarKolejki)
{
    int l=2*i+1;
    int r=2*i+2;
    int wiekszy = i;
    jeśli(l<rozmiarKolejki&&kolejka[l]>kolejka[wiekszy])//l<N-1
        wiekszy=l;
    jeśli(r<rozmiarKolejki&&kolejka[r]>kolejka[wiekszy])//N-1
        wiekszy=r;
    jeśli(wiekszy!=i)
    {
        swap(kolejka[wiekszy],kolejka[i]);
        zanurzanie(wiekszy,kolejka,rozmiarKolejki);
    }
}
```

```
funkcja utworzKopiec(int tab[],int n)
{
    dla(int i =n/2; i>=0; i--)
        zanurzanie(i,tab,n);
}
Funkcja sortuj(int tab[], int n)
{
    utworzKopiec(tab,n);
     dla (int i = n; i>=1; i--)
    {
        swap(tab[0],tab[n-1]);
        --n;
        zanurzanie(0,tab,n);
    }

}
```

5. Kod programu.

```
#include <iostream>
#include <fstream>
#include <string>
#include <time.h>
#include <cstdlib>
#include <stdio.h>
#include <iomanip>
#include <cmath>
#include <conio.h>
#include<windows.h>
```

```
#include <cstdio>
#include <ctime>


using namespace std;

const int N = 100000;

int ile,i ;
clock_t start,stop;
double czas;
int*tab_pom;
int liczby[100];

void scal(int* tab, int from, int sr, int to) {
  for (int i = from; i <= to; i++)
    tab_pom[i] = tab[i];
  int i = from, j = sr + 1;
  for (int k = from; k <= to; k++) {
    if (i <= sr) {
      if (j <= to) {
        tab[k] = (tab_pom[j] < tab_pom[i]) ? tab_pom[j++] : tab_pom[i++];
      } else {
        tab[k] = tab_pom[i++];
      }
    } else {
      tab[k] = tab_pom[j++];
```

```
    }
  }
}
void sortowaniescalanie(int* tab, int from, int to) {
  if (to <= from)
    return;
  int sr = (to + from) / 2;
  sortowaniescalanie(tab, from, sr);
  sortowaniescalanie(tab, sr + 1, to);
  scal(tab, from, sr, to);
}



void wstaw(int tab[],int n)  //funkcja zape³niaj¹ca tablicê
{
    for(int i=0; i<n; i++)
        tab[i]=rand()%1000;
}

void drukuj(int tab[],int n)
{
    for(int i =0; i<n; i++)
        cout<<tab[i]<<" ";
    cout<<endl;
} //funkcja do wyœwietlenia tablicy

void zanurzanie(int i, int kolejka[], int rozmiarKolejki)
```

```
{
    int l=2*i+1;
    int r=2*i+2;
    int wiekszy = i;
    if(l<rozmiarKolejki&&kolejka[l]>kolejka[wiekszy])//l<N-1
        wiekszy=l;
    if(r<rozmiarKolejki&&kolejka[r]>kolejka[wiekszy])//N-1
        wiekszy=r;
    if(wiekszy!=i)
    {
        swap(kolejka[wiekszy],kolejka[i]);
        zanurzanie(wiekszy,kolejka,rozmiarKolejki);
    }
}

void utworzKopiec(int tab[],int n)
{
    for(int i =n/2; i>=0; i--)
        zanurzanie(i,tab,n);
}
void sortuj(int tab[], int n)
{
    utworzKopiec(tab,n);
     for(int i = n; i>=1; i--)
    {
        swap(tab[0],tab[n-1]);
        --n;
```

```
        zanurzanie(0,tab,n);
    }


}




int main()
{
    fstream plik, wyniki;
    plik.open("tablica.txt",ios::in); // Otwarcie/utworzenie pliku, z którego pobierana bedzie tablica
    wyniki.open("wyniki.txt",ios::out); // Otwarcie/utworzenie pliku, do którego zapisywany bêdzie wynik


 cout<<"Program sortujacy przez scalanie oraz kopcowo, porownujacy dzialanie obu metod"<<endl;
 int x;
    cout<<"Jesli chcesz porownac czas dzialania obu metod wpisz 9 "<<endl;
    cout<<"Jesli chcesz posortowac dowolna tablice obiema metodami wpisz 6"<<endl;
    cout<<"Jesli chcesz odczytac tablice z pliku wpisz 3"<<endl;
    cin >> x;


    while((x!=9)&&(x!=6)&&(x!=3))//jezeli x nie jest rowny p,d lub o postepuj zgodnie z petla
    {
        cout<<"Wprowadz litere : 9, 6 lub 3! " <<endl;
        cin>>x;
```

```
  }

  if(x==9)
  {


   cout << "Porownanie czasow sortowania v.1" << endl;

  cout<<"Ile losowych liczb w tablicy: ";
  cin>>ile;

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

        tab_pom = new int[ile];
int* tab_pom = new int[ile];

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

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

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

```
        tablica[i] = rand()%1000+1;
    }
//przepisanie tablicy do tablicy2
    for(int i=0; i<ile; i++)
    {
      tablica2[i]=tablica[i];
    }
     for(int i=0; i<ile; i++)
    {
      tab_pom[i]=tablica[i];
    }



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


cout<<endl<<"Sortuje teraz przez scalanie. Prosze czekac!"<<endl;
    start = clock();
      sortowaniescalanie(tablica, 0, ile - 1);
    stop = clock();
    czas = (double)(stop-start) / CLOCKS_PER_SEC;
    cout<<endl<<"Czas sortowania przez scalanie: "<<czas<<" s"<<endl;
```

```
 cout<<"Po posortowaniu: "<<endl;
        for(int i=0; i<ile; i++)
        {
            cout<<tablica[i]<<" ";
        }



    cout<<endl<<"Sortuje teraz kopcowo. Prosze czekac!"<<endl;
    start = clock();
    drukuj(tablica, 0);
    stop = clock();
    czas = (double)(stop-start) / CLOCKS_PER_SEC;
    cout<<endl<<"Czas sortowania kopcowego: "<<czas<<" s"<<endl;

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



    delete [] tablica;
    delete [] tablica2;
    delete[] tab_pom;

    }
```

```
  if(x==6)
{
  cout << "Program sortujacy przez scalanie i kopcowo" << 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];
  tab_pom = new int[ile];
  int* tab_pom = 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];
  }
  for(int i=0; i<ile; i++)
  {
```

```
    tab_pom[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 przez scalanie."<<endl;

sortowaniescalanie(tablica, 0, ile - 1);

cout<<"Po posortowaniu przez scalanie: "<<endl;
wyniki <<"\n"<< "Po posortowaniu przez scalanie: "<<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 kopcowe."<<endl;
sortuj(tablica2,ile);

cout<<"Po posortowaniu kopcowym: "<<endl;
```

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

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

if(x==3)
    {
        int ile=10000;  //liczba elementow w pliku  //funkcja do wczytywania tablic do testow
    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 przez scalanie."<<endl;

sortowaniescalanie(tablica, 0, ile - 1);

cout<<"Po posortowaniu przez scalanie: "<<endl;
wyniki <<"\n"<< "Po posortowaniu przez scalanie: "<<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 kopcowe."<<endl;
sortuj(tablica2,ile);

cout<<"Po posortowaniu kopcowym: "<<endl;
wyniki <<"\n"<< "Po posortowaniu kopcowym: "<<endl;
for(int i=0; i<ile; i++)

{
    cout<<tablica[i]<<" ";
    wyniki<< tablica[i] << " ";
}
```

```
        cout<<endl;
            plik.close();
        wyniki.close();
    }
    return 0;
}
}
```

6. Działanie programu.

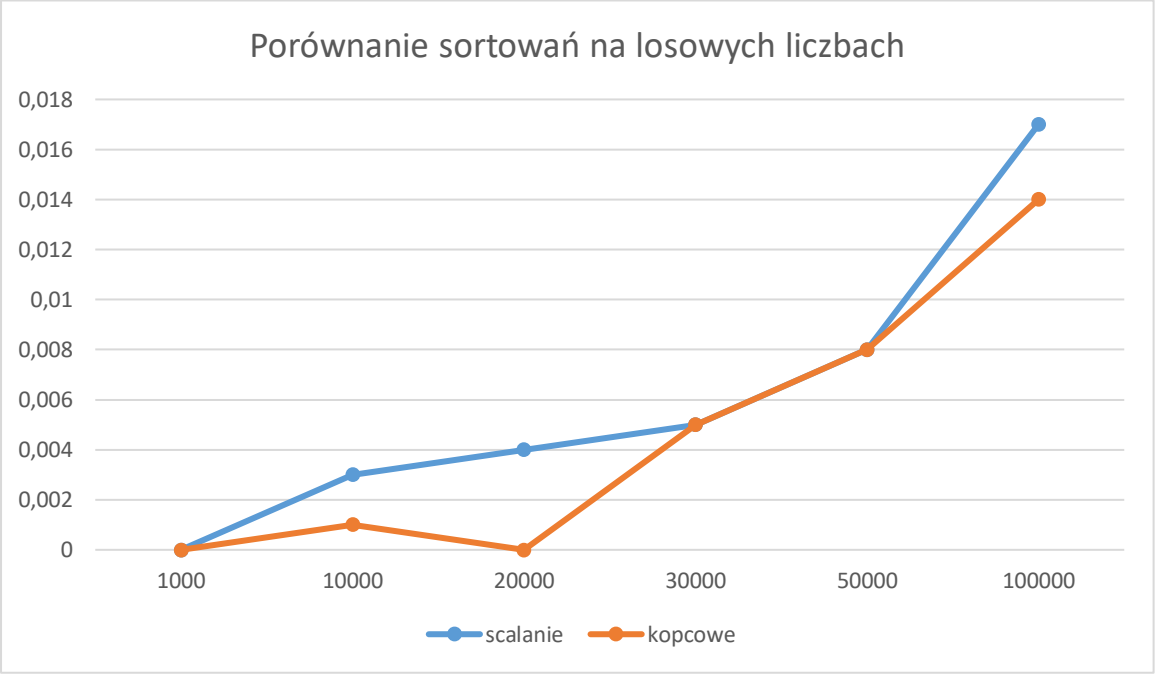
Program miał na celu pobranie liczb z pliku lub wczytanie losowych liczb i porównanie ich w konkretny sposób, a mianowicie sortowaniem przez scalanie oraz sortowaniem kopcowym.

Sortowanie przez scalanie to obecnie najszybsza znana metoda sortowania danych. Działanie algorytmu opiera się na scalaniu posortowanych już list. Posortowane listy uzyskujemy dzieląc listę elementów na pary elementów (możliwe jest wystąpienie elementu bez pary), które sortujemy w obrębie ich pod listy. Algorytm na początek musi podzielić listę na mniejsze podlisty o maksymalnie dwóch elementach. Kolejny krok polega na posortowaniu każdej z powstałych podlist. A następnie czas na kolejną część działania algorytmu – scalanie. Scalane listy są już posortowane, więc można porównać pierwsze elementy scalanych list. Dopisać na listę wynikową mniejszą liczbę i usunąć ze starej listy. Należy powtarzać krok porównywania do momentu, gdy nie będziemy mieć elementów do porównania - wtedy przepisujemy wszystkie pozostałe elementy do listy wynikowej z niepustej listy. W ten sposób uzyskujemy posortowaną listę.

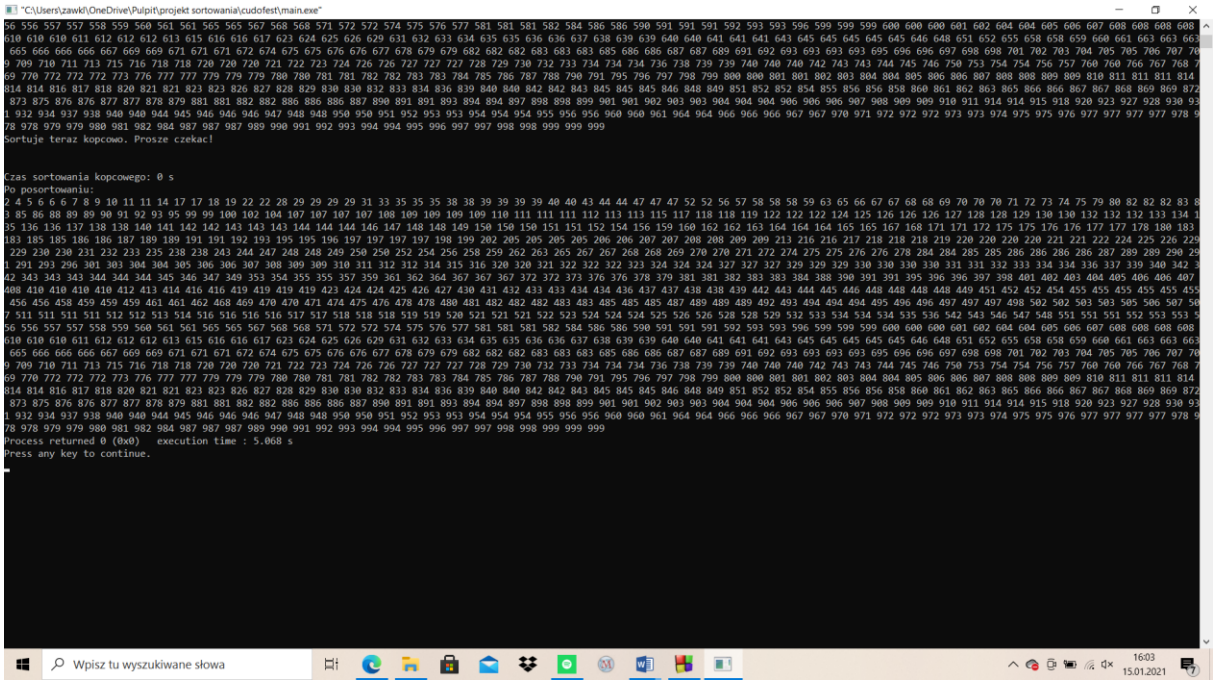
Natomiast sortowanie kopcowe składa się z kopca. Kopiec jest drzewem binarnym, w którym wszystkie węzły spełniają następujący warunek (zwany warunkiem kopca): Węzeł nadrzędny jest większy lub równy węzłom potomnym (w porządku malejącym relacja jest odwrotna - mniejszy lub równy). Konstrukcja kopca jest nieco bardziej skomplikowana od konstrukcji drzewa binarnego, ponieważ musimy dodatkowo troszczyć się o zachowanie warunku kopca. Zatem po każdym dołączeniu do kopca nowego węzła, sprawdzamy odpowiednie warunki i ewentualnie dokonujemy wymian węzłów na ścieżce

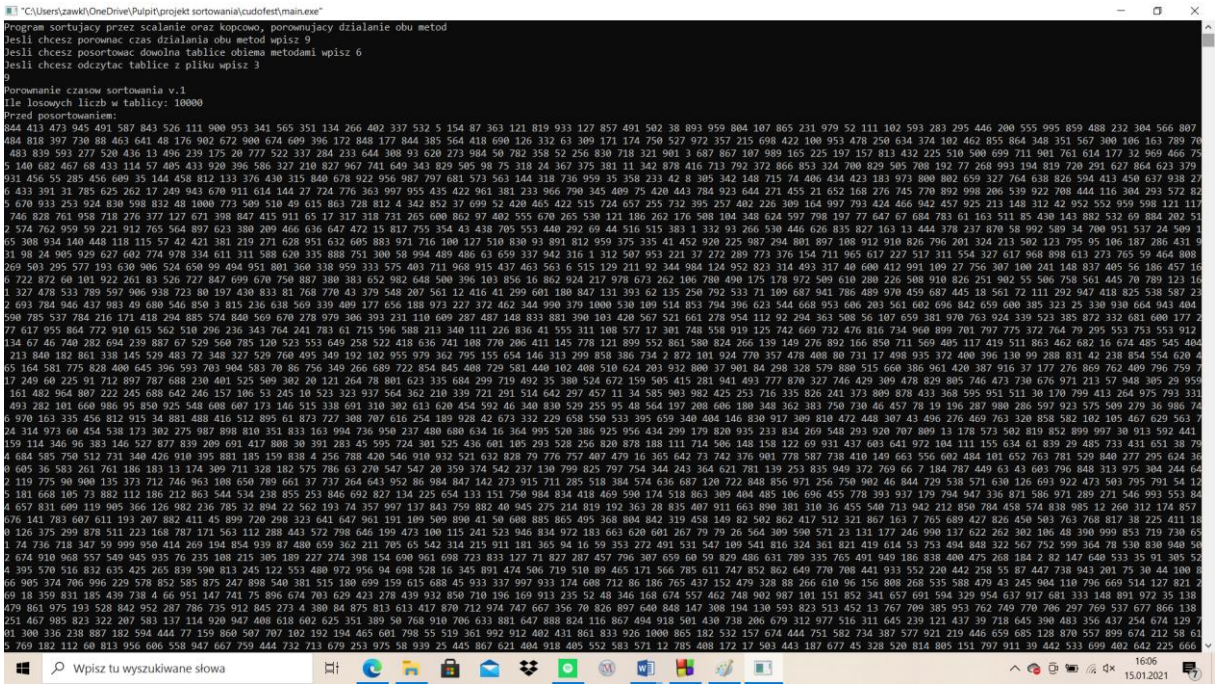
wiodącej od dodanego węzła do korzenia. Charakterystyczną cechą kopca jest to, iż korzeń zawsze jest największym (w porządku malejącym) elementem z całego drzewa.

Program posiada możliwość wczytania danych z pliku oraz posortowania tych danych. Działając na sortowaniach program ma możliwość porównania obu metod oraz pokazania różnicy czasu. Wyniki testów, które wykonałam pokazane są na wykresie poniżej.



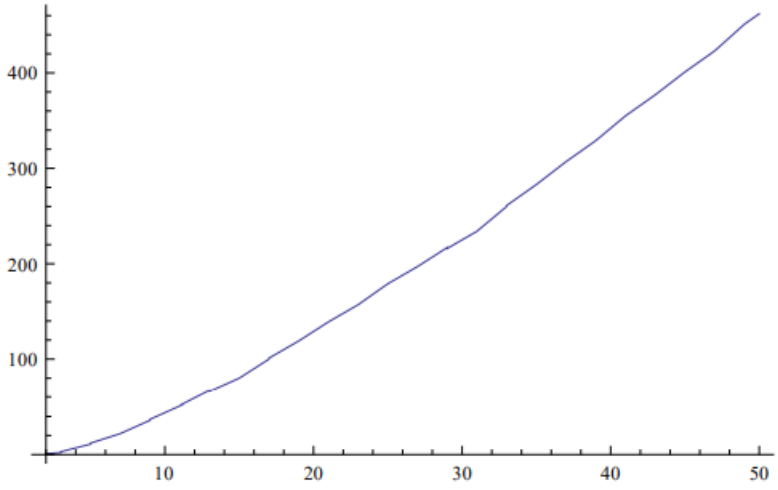
Poniżej zamieszczone są zrzuty ekranu z konsoli:


"C:\Users\zawkl\OneDrive\Pulpit\projekt sortowania\cudofest\main.exe"
Sortuje teraz kopcowo. Prosze czekac!
Czas sortowania kopcowego: 0 s
Po posortowaniu:
Process returned 0 (0x0)   execution time : 5.068 s
Press any key to continue.
Wpisz tu wyszukiwane słowa
16:03
15.01.2021


"C:\Users\zawkl\OneDrive\Pulpit\projekt sortowania\cudofest\main.exe"
Program sortujacy przez scalanie oraz kopcowo, porownujacy działanie obu metod
Jesli chcesz porownac czas działania obu metod wpisz 9
Jesli chcesz posortowac dowolna tablice obiema metodami wpisz 6
Jesli chcesz odczytac tablice z pliku wpisz 3
9
Porownanie czasow sortowania v.1
Ile losowych liczb w tablicy: 10000
Przed posortowaniem:
Wpisz tu wyszukiwane słowa
16:06
15.01.2021

Złożoność obliczeniowa:

Dokładna analiza przypadków w sortowaniu kopcowym



Sortowanie kopcowe:
Najlepszy przypadek- nlogn
Najgorszy przypadek- nlogn

Liczba porównań dokonanych faktycznie może zależeć od kolejności, w jakiej wartości są podane. Fakt, że najlepszym i najgorszym przypadkiem są każdy Θ(n log n) - zakładając, że wszystkie elementy są różne - oznacza tylko, że *asymptotycznie* nie ma różnicy między nimi, choć mogą się różnić stałym czynnikiem. Nie mam żadnych prostych przykładów tego z góry mojej głowy, ale wierzę, że można tworzyć dane wejściowe, gdzie liczba porównań różni się stałym czynnikiem między tymi dwoma podejściami. Ponieważ notacja big-O ignoruje stałe, nie znajduje to odzwierciedlenia w analizie najlepszych i najgorszych przypadków.

Sortowanie przez scalanie:

Najgorszym przypadkiem scalenia będzie ten, w którym sortowanie scalania będzie musiało wykonać **maksymalną liczbę porównań.**

Przypadek pesymistyczny:

1. Załóżmy, że tablica w ostatnim kroku po sortowaniu jest {0,1,2,3,4,5,6,7}
2. W najgorszym przypadku tablica przed tym krokiem musi być, ponieważ tutaj lewa tam_pom= i prawa tab_pom= spowoduje maksymalne

porównania. (*Przechowywanie alternatywnych elemetów w lewej i prawej podawanie*{0,2,4,6,1,3,5,7}{0,2,4,6}{1,3,5,7})

Powód: Każdy element tablicy będzie porównywany conajmniej raz.

3. Zastosowanie tej samej powyższej logiki dla lewej i prawej podarze dla poprzednich kroków: Dla tablicy najgorszym przypadkiem będzie, jeśli poprzednia tablica jest i i dla tablicy najgorszym przypadkiem będzie dla i .{0,2,4,6}{0,4}{2,6}{1,3,5,7}{1,5}{3,7}
4. Teraz zastosowanie tego samego dla poprzednich tablic kroków: *W najgorszych przypadkach:*musi być , musi być , musi być . Cóż, jeśli spojrzeć wyraźnie ten krok nie jest **konieczne,** ponieważ jeśli rozmiar tab_pom jest 2 następnie każdy element będzie porównywany conajmniej raz, nawet jeśli tablica rozmiar 2 jest sortowane.{0,4}{4,0}{2,6}{6,2}{1,5}{5,1}{3,7}{7,3}

Przypadek optymistyczny:

Co jest najlepszym przypadkiem operacji scalania, musimy tylko wziąć pod uwagę część algorytmu, gdzie "splata" dwa tab_pom razem. Podprogram ten działa w O(n+m)O(n+m) Gdzie NN I MM są długościami dwóch podabli. Dzieje się tak, ponieważ poszczególne podaże są już sortowane z poprzedniej operacji scalania. Podsumowując, operacja scalania przebiega w czasie liniowym. Między każdą parą poziomów tablicy jest poziom scalania. Nawet jeśli poprzedni poziom tablicy nie jest scalanie w jednej tablicy na następnym poziomie, nadal jest scalany do innej tablicy na tym samym poziomie.

7. Wnioski.

Zadanie projektowe rozszerzyło moją wiedzę z zakresu programowania, a w szczególności z zakresu sortowań. Program działa prawidłowo. Posiada możliwość odczytu danych z pliku tekstowego, a także zapisania wyników do pliku tekstowego. W programie umieściłam także odpowiednie komentarze, które pomagają w dokładnym zrozumieniu jego działania. Zadanie projektowe pozwoliło mi na lepsze zrozumienie pojęcia funkcji w programowaniu oraz poćwiczenie podstawowych pętli jak i instrukcji warunkowych.