**Katarzyna Wojciechowska**

Katarzyna Wojciechowska 166714

Inżynieria i analiza danych, rok I, grupa P08

Sprawozdanie z projektu numer 2.

Spis treści

[2. Opis problemu 4](#_Toc61018647)

[2.1. Treść zadania 4](#_Toc61018648)

[2.2. Sortowanie 4](#_Toc61018649)

[2.3. Krótki opis programu 4](#_Toc61018650)

[3. Sortowanie przez wstawianie (Insertion Sort) 5](#_Toc61018651)

[3.1. Podstawy teoretyczne sortowania przez wstawianie 5](#_Toc61018652)

[3.1.1. Strategia algorytmu 5](#_Toc61018653)

[3.2. Kod algorytmu w c++ 5](#_Toc61018654)

[3.3. Schemat blokowy 6](#_Toc61018655)

[3.4. Pseudokod 7](#_Toc61018656)

[3.5. Złożoność algorytmu 7](#_Toc61018657)

[3.5.1. Przypadek optymistyczny 7](#_Toc61018658)

[3.5.2. Przypadek pesymistyczny 7](#_Toc61018659)

[3.5.3. Przypadek średni 7](#_Toc61018660)

[4. Sortowanie kopcowe (HeapSort) 8](#_Toc61018661)

[4.1. Podstawy teoretyczne sortowania kopcowego 8](#_Toc61018662)

[4.1.1. Budowa kopca 8](#_Toc61018663)

[4.1.2. Rozbiór kopca 9](#_Toc61018664)

[4.2. Strategia algorytmu 9](#_Toc61018665)

[4.3. Kod algorytmu w c++ 9](#_Toc61018666)

[4.4. Pseudokod 10](#_Toc61018667)

[4.5. Schemat blokowy 12](#_Toc61018668)

[4.6. Złożoność algorytmu 13](#_Toc61018669)

[4.6.1. Złożoność optymistyczna 13](#_Toc61018670)

[4.6.2. Złożoność pesymistyczna 13](#_Toc61018671)

[4.6.3. Złożoność typowa 13](#_Toc61018672)

[5. Program 14](#_Toc61018673)

[6. Dokumentacja z doświadczeń 21](#_Toc61018674)

[6.1. Sortowanie z pliku 21](#_Toc61018675)

[6.1.1. Sortowanie przez wstawianie 21](#_Toc61018676)

[6.1.2. Sortowanie kopcowe 22](#_Toc61018677)

[6.2. Sortowanie ciągów uporządkowanych rosnąco – porównywanie czasu wykonywanie algorytmów 23](#_Toc61018678)

[6.3. Sortowanie ciągów uporządkowanych malejąco 25](#_Toc61018679)

[6.4. Sortowanie losowych ciągów o różnych długościach – porównanie czasów wykonywania algorytmu 26](#_Toc61018680)

[6.5. Wnioski z wykonanych testów 28](#_Toc61018681)

[7. Spis ilustracji 30](#_Toc61018682)

[8. Załączniki 30](#_Toc61018683)

[9. Źródła 30](#_Toc61018684)

# Opis problemu

## Treść zadania



## Sortowanie

**Sortowanie** – jeden z ważniejszych zagadnień informatyki, polegający na uporządkowaniu zbioru danych względem pewnych cech charakterystycznych każdego elementu tego zbioru.

Algorytmy sortowania są stosowane w celu uporządkowania danych, umożliwienia stosowania wydajniejszych algorytmów (np. wyszukiwania) i prezentacji danych w sposób czytelniejszy dla użytkownika. Jeśli jest należy posortować zbiór większy niż wielkość dostępnej pamięci, stosuje się **algorytmy sortowania zewnętrznego**. Algorytmy, do działania których nie jest potrzebna większa niż stała pamięć dodatkowa (elementy sortowane przechowywane są przez cały czas w tablicy wejściowej), nazywane są **algorytmami działającymi w miejscu**.

Algorytmy sortujące, które dla elementów o tej samej wartości zachowują w tablicy końcowej kolejność tablicy wejściowej, nazywamy **algorytmami stabilnymi**.

## Krótki opis programu

W programie zaimplementowane zostały **algorytmy sortowania przez wstawanie** i **sortowania kopcowego**. Program posiada możliwość odczytu danych z pliku oraz zapisu do plików wyników jego działania.

Po uruchomieniu pojawia się możliwość wyboru jednego z 5 sposobów działania. Pierwsze 3 opcje to porównywanie czasów wykonywania się algorytmu – w przypadku, gdy dane wejściowe są uporządkowane malejąco, w przypadku odwrotnym oraz przy sortowaniu losowych ciągów o dowolnej długości. Pozostałe 2 funkcje programu to sortowanie przez wstawianie oraz sortowanie kopcowe liczb z pliku wraz z pomiarem czasu wykonywania się algorytmu.

Po wyborze sposobu działania programu wykonywane są wybrane działania, a ich wyniki – czasy wykonywania poszczególnych algorytmów, a w dwóch przypadkach również posortowane ciągi, wyświetlane są na ekranie i zapisywane do pliku tekstowego.

# Sortowanie przez wstawianie (Insertion Sort)

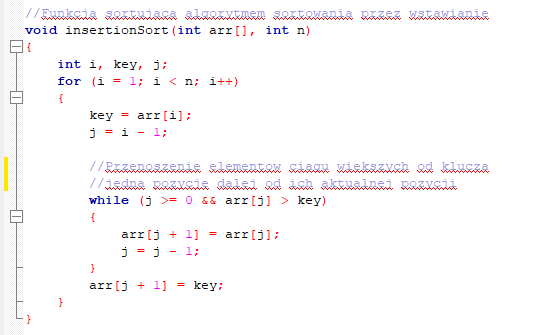
## Podstawy teoretyczne sortowania przez wstawianie [[1]](#footnote-1)

Algorytm zaliczany jest do algorytmów **stabilnych**. Ten rodzaj sortowania możemy porównać do układania kart. Pierwszą kartę wstawiamy w odpowiednie miejsce przesuwając pozostałe, następną także wstawiamy między odpowiednie karty i tak układamy zestaw kart.

### Strategia algorytmu

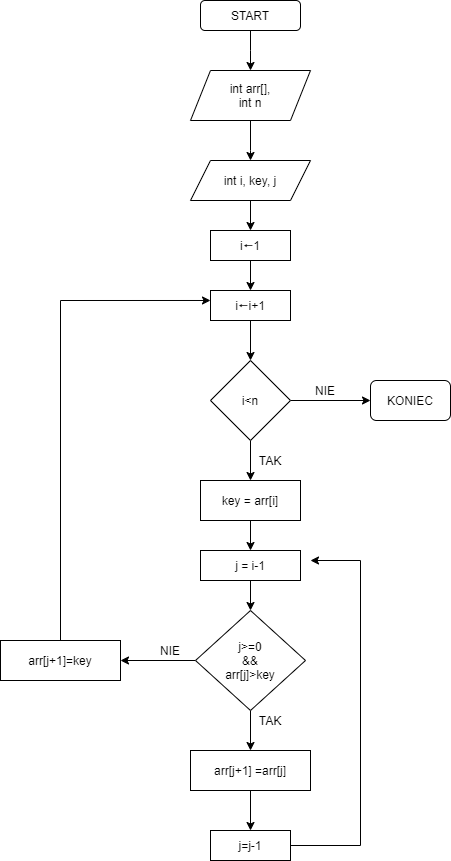
* Rozpoczynamy od drugiego elementu i porównujemy go z elementami poprzedzającymi
* Jeśli napotkamy liczbę większą, to musimy przesunąć ją o jeden w prawo.
* Czynność tą powtarzamy do momentu napotkania liczby niemniejszej lub gdy skończą nam się liczby (nie będzie spełniony warunek j>=0)
* W następnym kroku wstawiamy naszą liczbę w odpowiednie miejsce   
  i otrzymujemy podzbiór uporządkowany.
* Powyższe czynności powtarzamy dla reszty wyrazów.

## Kod algorytmu w c++



1 Kod sortowania przez wstawianie w c++

## Schemat blokowy



2 Schemat blokowy sortowania przez wstawianie

## Pseudokod

Funkcja insertionSort (int arr[], int n)

{ int i, key, j

dla i=1 do n wykonuj

{

key = arr[i]

j = i-1

dopóki j>=0 oraz arr[j]>key

{

arr[j+1] = arr[j]

j = j-1

}

arr[j+1]=key

}

}

## Złożoność algorytmu

Algorytm sortowania przez wstawianie posiada **klasę czasowej złożoności obliczeniowej równą O(n2)**. Sortowanie odbywa się w miejscu.

### Przypadek optymistyczny

W przypadku, gdy dane wejściowe są uporządkowane rosnąco, przy każdym sprawdzeniu pętla nie wykonuje się ani razu, więc liczba sprawdzeń warunku wynosi 1 dla wszystkich j od 1 do n – całkowity czas T(n) wyraża się funkcją liniową. Dla posortowanych tablic o długości n algorytm działa w czasie O(n).

### Przypadek pesymistyczny

W przypadku, gdy dane wejściowe są uporządkowane malejąco, w każdym przebiegu pętli głównej, przy sprawdzaniu warunku pętla wewnętrzna wykonuje się i-1 razy w każdym i-tym przebiegu. Algorytm w tym przypadku ma złożoność O(n2).

### Przypadek średni

Przy założeniu, że wszystkie możliwe wartości tablicy występują z jednakowym prawdopodobieństwem, złożoność, podobnie jak w przypadku pesymistycznym wynosi O(n2).

# Sortowanie kopcowe (HeapSort)

## Podstawy teoretyczne sortowania kopcowego

### Budowa 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).

1. Budowę kopca rozpoczyna się od pierwszego elementu zbioru, który staje się korzeniem.
2. Dołącza się następny element i sprawdza czy warunek kopca jest zachowany.
3. Jeśli warunek jest zachowany – dodajemy kolejny element, jeśli warunek nie jest zachowany – należy go przywrócić, w tym celu zamienia się węzeł nadrzędny i węzeł dodany.

7

9

5

7

5

7

7

5

9

3 Budowa kopca

### Rozbiór kopca

1. Zamiana korzenia z ostatnim liściem, który wyłączany jest ze struktury kopca. Z kopca zawsze pobiera się jego korzeń – element największy.
2. Jeśli jest to konieczne, należy przywrócić warunek kopca
3. Powtarza się od kroku 1, aż kopiec będzie pusty.

1

2

5

4

2

4

2

1

1

4

1

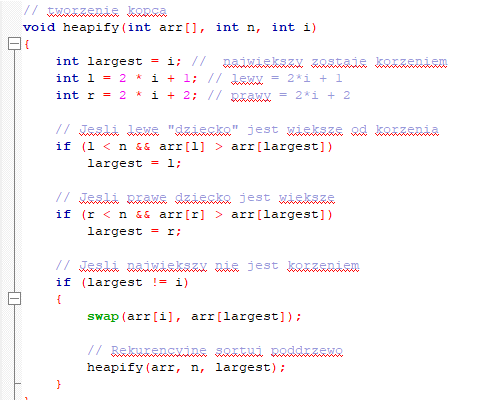
4 Fragment rozbioru kopca

## Strategia algorytmu

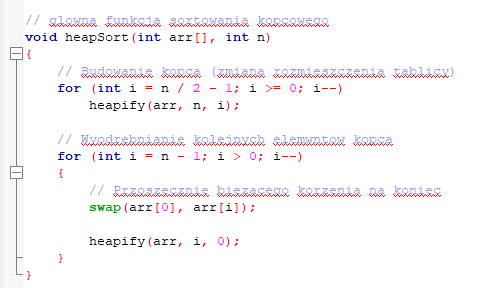
Algorytm zaliczany jest do **algorytmów niestabilnych**. Właściwe sortowanie następuje po utworzeniu kopca i opiera się na jego rozbieraniu. Z kopca usunięty zostaje największy element, w jego miejsce wstawiany jest element z końca kopca i przywracany zostaje warunek kopca. W zwolnione miejsce tuż za końcem pomniejszonego kopca wstawiany jest usunięty element maksymalny. Operacja powtarzana jest do wyczerpania elementów kopca. Sortowanie odbywa się w miejscu.

## Kod algorytmu w c++

Kod algorytmu podzielony został na 2 funkcje. Pierwsza w funkcji odpowiedzialna jest za tworzenie kopca z wprowadzonych danych. Natomiast druga to funkcja główna, która wykorzystuje utworzony w poprzedniej funkcji kopiec. To w drugiej funkcji odbywa się właściwe sortowanie.



5 Tworzenie kopca w c++



6 Główna funkcja sortowania kopcowego w c++

## Pseudokod

funkcja heapify (int arr[], int n, int i)

{ int largest = i

int l=2\*i+1

int r=2\*i+2

jeżeli l<n oraz arr[l]>arr[largest]

largest = l

jeżeli r<n oraz arr [r]>arr[largest]

largest = r

jeżeli largest jest różny od i

{

zamień arr[i] oraz arr[largest]

heapify (arr, n, largest)

}

}

funkcja heapSort (int arr[], int n)

{

dla i=n/2-1 do i>=0 wykonuj

{

heapify (arr,n,i)

i=i-1

}

dla i=n-1 do 1>0 wykonuj

{

Zamien arr[0] z arr[i]

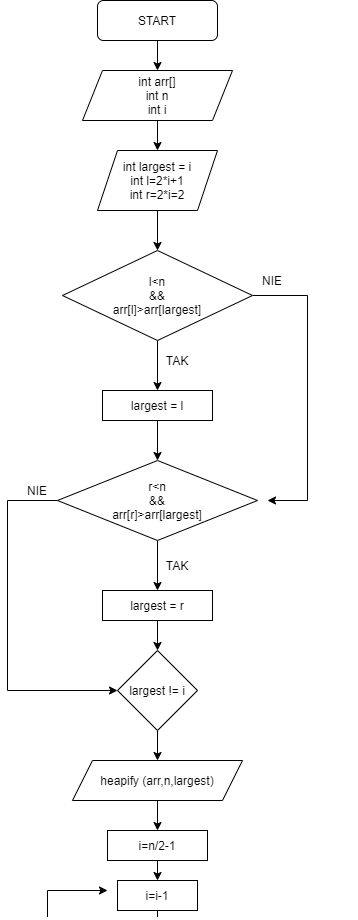
heapify (arr,i,0)

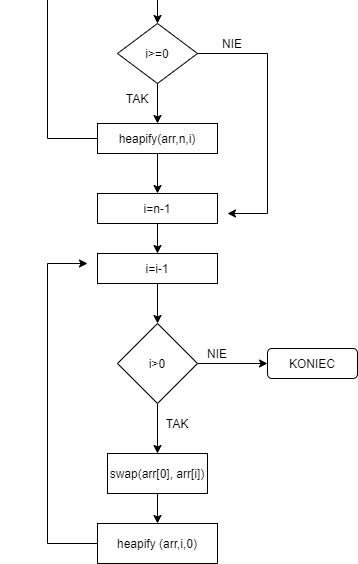
i=i-1

}

}

## Schemat blokowy





7 Schemat blokowy sortowania kopcowego

## Złożoność algorytmu

### Złożoność optymistyczna

Złożoność dla zbiorów uporządkowanych (z niewielką liczbą elementów nie na swoich miejscach) – **O(nlogn).**

### Złożoność pesymistyczna

Złożoność dla zbiorów posortowanych odwrotnie - **O(nlogn)**.

### Złożoność typowa

Złożoność dla zbiorów o losowym rozkładzie elementów - **O(nlogn)**.

# Program

#include <iostream>

#include <fstream>

#include <conio.h>

#include <cstdlib>

#include <time.h>

#include <windows.h>

using namespace std;

clock\_t start, stop;

double t1, t2, t3;

int N;

int \*tab = new int[N];

void insertionSort(int arr[], int n) //Funkcja sortujaca algorytmem sortowania przez wstawianie

//funkcja sortujaca algorytmem sortowania kopcowego

void heapify(int arr[], int n, int i) // torzenie kopca

void heapSort(int arr[], int n) // glowna funkcja sortowania kopcowego

void printResults(int arr[], int n) //zapis do pliku

//testy programu

void test\_1() //czas wykonywania algorytmu w przypadku, kiedy ciag jest posortowany

{

int N;

cout << "Ile elementow ma posiadac ciag? ";

cin >> N;

while(cin.fail() || N<=0)

{

cout << "Blad! podaj poprawna liczbe!" << endl;

cin.clear();

cin.ignore(256,'\n');

cin >> N;

}

int \*tab = new int[N];

srand(time(NULL));

for (int i=0; i<N; i++)

{

tab[i]=rand()%101; //losujemy liczby z zakresu od 0 do 100

}

heapSort (tab,N); //sortowanie wylosowanej tablicy

ofstream plik("test\_1.txt"); //otwarcie pliku w celu zapisania w nim rozwiazan

int ilosc\_testow;

cout << "Ile prob ma zostac wykonane? ";

cin >> ilosc\_testow;

for (int i=0; i<ilosc\_testow; i++)

{

cout<<endl<<"Proba nr. "<<i+1<<endl;

//sortowanie przez wstawianie

cout<<"Trwa sortowanie przez wstawianie..."<<endl;

start = clock();

insertionSort (tab, N);

stop = clock();

t1=(double)(stop-start)/CLOCKS\_PER\_SEC;

cout << "Czas sortowania przez wstawianie: "<<t1<<" s" <<endl<<endl;

//sortowanie kopcowe

cout<<"Trwa sortowanie kopcowe..."<<endl;

start = clock();

heapSort (tab,N);

stop = clock();

t2=(double)(stop-start)/CLOCKS\_PER\_SEC;

cout << "Czas sortowania kopcowego: "<<t2<<" s" <<endl<<endl;

plik << "Próba nr. "<<i+1<<endl<<endl;

plik << "Czas sortowania przez wstawianie: "<<t1<<" s"<< endl;

plik << "Czas sortowania kopcowego: "<<t2<<" s"<< endl<<endl;

}

plik.close();

}

void test\_2() //czas wykonywania algorytmu w przypadku, kiedy ciag jest posortowany odwrotnie

{

int B[20000]= {0};

int n = 20000;

fstream z;

z.open("test2\_plik(20000).txt",ios::in); //odczytywanie danych z pliku

//z.open("test2\_plik(50000).txt",ios::in);

//z.open("test2\_plik(100000).txt",ios::in);

//z.open("test2\_plik(200000).txt",ios::in);

//z.open("test2\_plik(300000).txt",ios::in);

if(!z) //jesli nie uda sie otworzyc pliku

{

cout<<"Nie udalo sie otworzyc pliku";

getchar();

}

while (!z.eof())

z>>B[n++];

ofstream plik("test\_2.txt"); //otwarcie pliku w celu zapisania w nim rozwiazan

//czas sortowania przez wstawianie

cout<<"Trwa sortowanie przez wstawianie..."<<endl;

start = clock();

insertionSort(B, n);

stop = clock();

t1=(double)(stop-start)/CLOCKS\_PER\_SEC;

cout << "Czas sortowania przez wstawianie: "<<t1<<" s" <<endl<<endl;

//czas sortowania kopcowego

cout<<"Trwa sortowanie kopcowe..."<<endl;

start = clock();

heapSort(B, n);

stop = clock();

t2=(double)(stop-start)/CLOCKS\_PER\_SEC;

cout << "Czas sortowania kopcowego: "<<t2<<" s" <<endl<<endl;

plik << "Czas sortowania przez wstawianie: "<<t1<<" s"<< endl;

plik << "Czas sortowania kopcowego: "<<t2<<" s"<<endl<<endl;

plik.close();

}

void test\_3() //czas wykonywania algorytmu w przypadku losowego ciagu

{

int N;

cout << "Ile elementow ma posiadac ciag? ";

cin >> N;

while(cin.fail() || N<=0)

{

cout << "Blad! podaj poprawna liczbe!" << endl;

cin.clear();

cin.ignore(256,'\n');

cin >> N;

}

int \*tab = new int[N];

ofstream plik("test\_3.txt"); //otwarcie pliku w celu zapisania w nim rozwiazan

int ilosc\_testow;

cout << "Ile prob ma zostac wykonane? ";

cin >> ilosc\_testow;

for (int i=0; i<ilosc\_testow; i++)

{

srand(time(NULL));

for (int i=0; i<N; i++)

{

tab[i]=rand()%101; //losujemy liczby z zakresu od 0 do 100

}

cout<<endl<<"Proba nr. "<<i+1<<endl;

//sortowanie przez wstawianie

cout<<"Trwa sortowanie przez wstawianie..."<<endl;

start = clock();

insertionSort (tab, N);

stop = clock();

t1=(double)(stop-start)/CLOCKS\_PER\_SEC;

cout << "Czas sortowania przez wstawianie: "<<t1<<" s" <<endl<<endl;

//sortowanie kopcowe

cout<<"Trwa sortowanie kopcowe..."<<endl;

start = clock();

heapSort (tab,N);

stop = clock();

t2=(double)(stop-start)/CLOCKS\_PER\_SEC;

cout << "Czas sortowania kopcowego: "<<t2<<" s" <<endl<<endl;

plik << "Próba nr. "<<i+1<<endl<<endl;

plik << "Czas sortowania przez wstawianie: "<<t1<<" s"<< endl;

plik << "Czas sortowania kopcowego: "<<t2<<" s"<< endl<<endl;

}

plik.close();

}

int main()

{

cout<<"1 - Test 1 - Tablica posortowana"<<endl<<endl;

cout<<"2 - Test 2 - Tablica posortowana odwrotnie"<<endl<<endl;

cout<<"3 - Test 3 - Tablica zlozona z liczb losowych"<<endl<<endl;

cout<<"4 - Sortowanie przez wstawianie - Liczby z pliku"<<endl<<endl;

cout<<"5 - Sortowanie kopcowe - Liczby z pliku"<<endl<<endl;

int opcja;

cout<<"Wybierz opcje: ";

cin>>opcja;

switch (opcja)

{

case 1: test\_1(); break;

case 2: test\_2();break;

case 3: test\_3();break;

case 4:

{

int t[50000]= {0};

int n = 50000;

fstream z;

z.open("plik.txt",ios::in); //odczytywanie danych z pliku

if(!z) //jesli nie uda sie otworzyc pliku

{

cout<<"Nie udalo sie otworzyc pliku";

getchar();

return 1;

}

while (!z.eof())

z>>t[n++];

z.close();

cout<<"Trwa sortowanie przez wstawianie..."<<endl;

start = clock();

insertionSort(t, n);

stop = clock();

t3=(double)(stop-start)/CLOCKS\_PER\_SEC;

cout << endl << "Czas sortowania przez wstawianie: "<<t3<<" s" <<endl;

printResults(t, n);

}

break;

case 5:

{

int t[50000]= {0};

int n = 50000;

fstream z;

z.open("plik.txt",ios::in); //odczytywanie danych z pliku

if(!z) //jesli nie uda sie otworzyc pliku

{

cout<<"Nie udalo sie otworzyc pliku";

getchar();

return 1;

}

while (!z.eof())

z>>t[n++];

z.close();

cout<<"Trwa sortowanie kopcowe..."<<endl;

start = clock();

heapSort(t, n);

stop = clock();

t3=(double)(stop-start)/CLOCKS\_PER\_SEC;

cout << endl << "Czas sortowania kopcowego: "<<t3<<" s" <<endl;

printResults(t, n);

}

}

getch();

return 0;

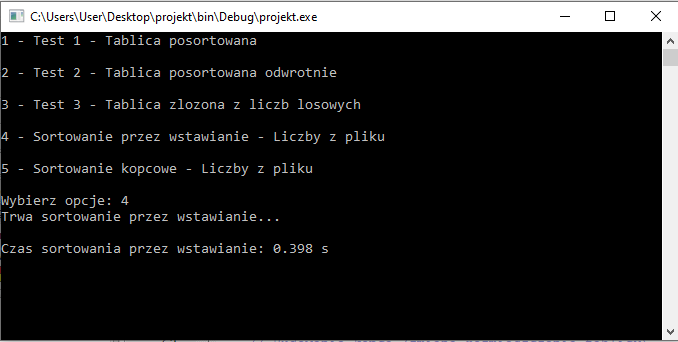
}

# Dokumentacja z doświadczeń

## Sortowanie z pliku

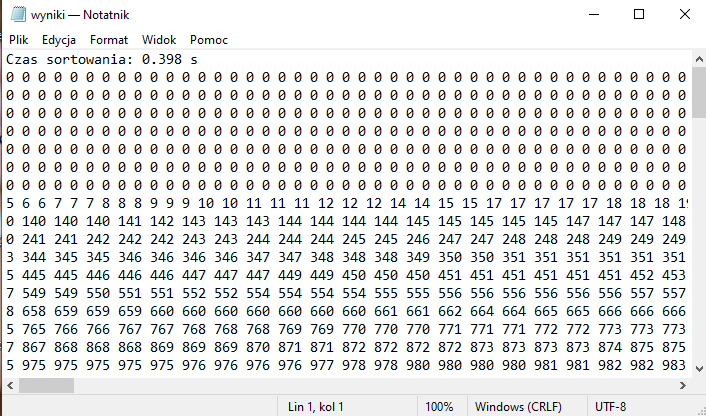
### Sortowanie przez wstawianie

Do wykonania testu użyty został plik zawierający 200000 losowych liczb. Wyniki wyświetlone na ekranie:



8 Sortowanie przez wstawianie - dane z pliku – wyniki

W pliku tekstowym został umieszczony czas sortowania oraz posortowana tablica.

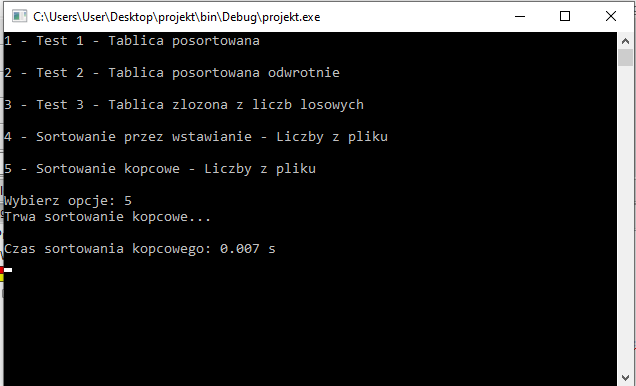


9 Sortowanie przez wstawianie - dane z pliku - wyniki w pliku tekstowym

Czas sortowania przez wstawianie wyniósł 0.398s.

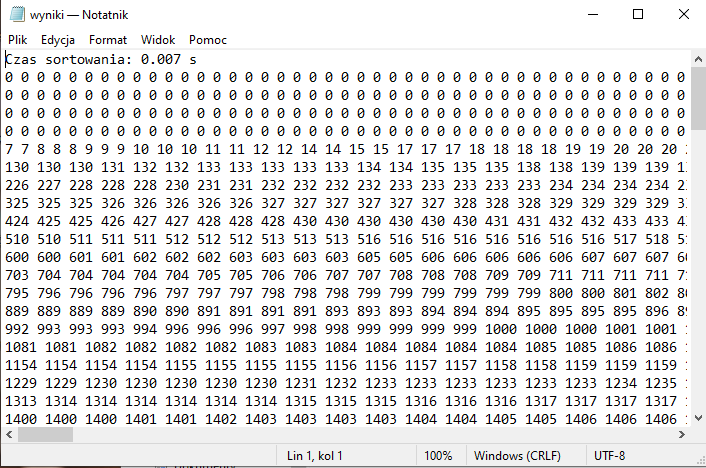
### Sortowanie kopcowe

Użyty został ten sam plik co w przypadku sortowania przez wstawianie. Wyniki wyświetlone na ekranie:



10 Sortowanie kopcowe - dane z pliku – wyniki

Podobnie jak w poprzednim przypadku, w pliku tekstowym oprócz czasu wykonywania algorytmu została umieszczona posortowana tablica.

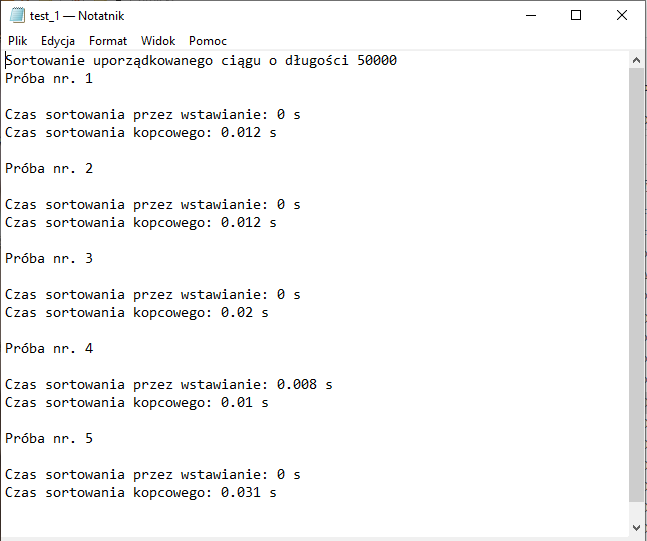


11 Sortowanie kopcowe - dane z pliku – wyniki w pliku tekstowym

Czas sortowania kopcowego w danym przypadku wyniósł 0.007 s.

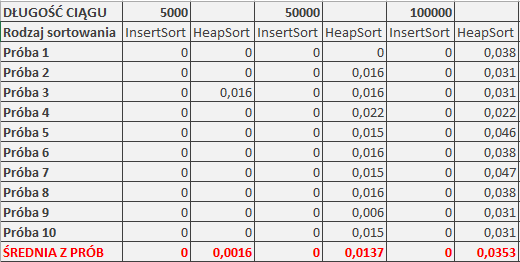
## Sortowanie ciągów uporządkowanych rosnąco – porównywanie czasu wykonywanie algorytmów

Test został wykonany na losowych ciągach o długościach 5000, 20000, 50000, 100000, 200000. Dla każdej z wymienionych długości zostało wykonane 10 prób. Po wylosowaniu ciągu, zostaje on uporządkowany, a następnie za sortowany za pomocą algorytmu sortowania przez wstawianie oraz sortowania kopcowego. Wyniki testu zapisywane są do pliku test\_1.txt (12 Test\_1 -wyniki w pliku tekstowym)

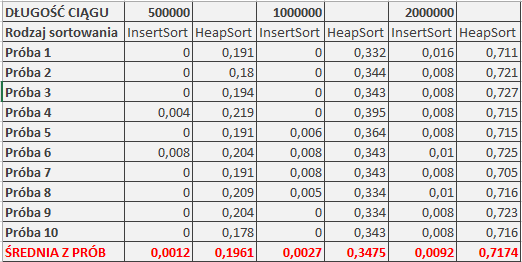


12 Test\_1 -wyniki w pliku tekstowym

W poniższych tabelach wykonanych w Excelu (plik dołączony jako załącznik) zaprezentowane zostały wyniki wszystkich 10 prób na ciągach o wyżej wymienionych długościach.



13 Test\_1 - wyniki 10 prób - tabela z Excela cz.1

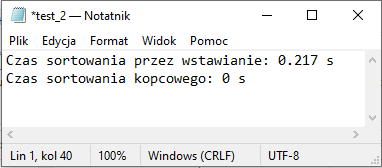


14 Test\_1 – wyniki 10 prób – tabela z Excela cz.2.

Uśrednione pomiary zostały zaprezentowane na poniższym wykresie:

## Sortowanie ciągów uporządkowanych malejąco

W tym teście program odczytuje dane z kolejnych plików o nazwach: test2\_plik(liczba\_elementów). Pliki zawierają kolejno 20000, 100000, 200000, 300000 elementów. Wyniki działania programu zapisywane są w pliku tekstowym. (18 Test\_2 -wyniki w pliku tekstowym)

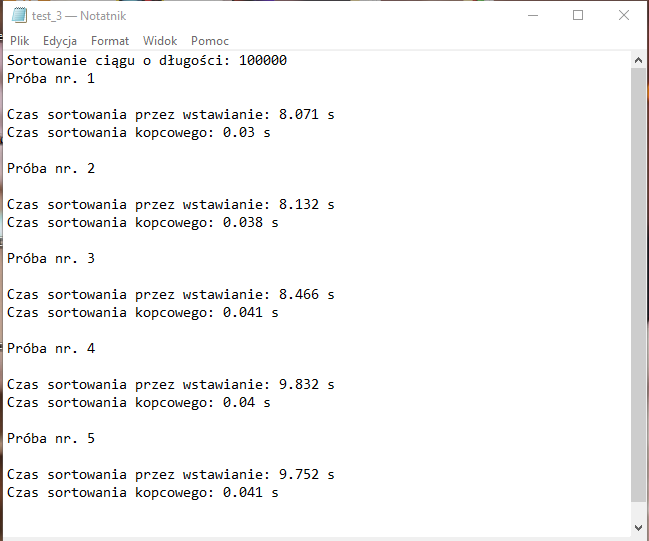


18 Test\_2 - wyniki w pliku tekstowym

Wyniki testu zostały przedstawione na poniższym wykresie:

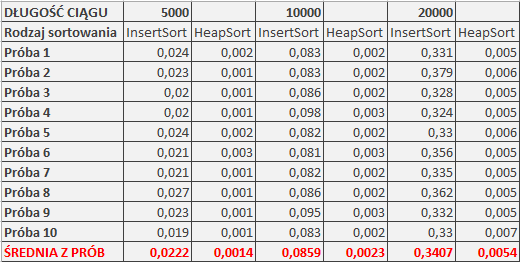
## Sortowanie losowych ciągów o różnych długościach – porównanie czasów wykonywania algorytmu

Wykonane zostały próby na losowych ciągach o długościach: 5000, 10000, 20000, 50000, 100000, 200000. Na każdej długości ciągu zostało wykonane 10 prób. Każdorazowo wyniki wykonywanego testu numer 3 zapisywane są do pliku test\_3.txt. Przykład wyników zapisanych do pliku:

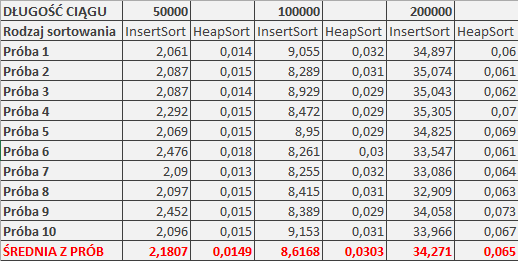


15 Test\_3 - wyniki w pliku tekstowym

Wyniki 10 prób wykonanych na ciągach o wyżej wymienionych długościach zostały zaprezentowane w tabeli wykonanej w Excelu (plik dołączony jako załącznik):



16 Test\_3 - wyniki 10 prób - tabela z Excela cz.1



17 Test\_3 - wyniki 10 prób – tabela z Excela cz.2

Uśrednione pomiary czasu sortowania poszczególnymi algorytmami ciągów o różnych długościach przedstawia poniższy wykres.

## Wnioski z wykonanych testów

Czasy wykonywania algorytmów sortowania przez wstawianie i sortowania kopcowego bardzo się różnią w zależności od długości i typu badanego ciągu.

W przypadku, gdy tablice są posortowane rosnąco, lepiej sprawdza się sortowanie przez wstawianie, co zostało zobrazowane w podrozdziale 5.2. Czas wykonywania tego algorytmu wzrasta nieznacznie nawet dla dużych tablic i wypada dużo korzystniej niż algorytm sortowania kopcowego – jego czas wykonywania rośnie znacznie szybciej dla coraz dłuższych ciągów liczb. W przypadku ciągów o długości 2000000 elementów czas sortowania przez wstawianie wynosi 0,0092 s, natomiast sortowania kopcowego aż 0,7174s. Różnica jest wyraźna.

W sytuacji odwrotnej, gdy tablice są posortowane malejąco, lepsze wydaje się zastosowanie algorytmu sortowania kopcowego. Porównanie czasu wykonywania sortowania poszczególnymi algorytmami zostało przedstawione w podrozdziale 5.3. Przy badaniu czasu w przypadku sortowania kopcowego, nawet dla długich ciągów, wyświetlany czas jest równy 0, co oznacza że czas wykonywania sortowania jest mniejszy niż 1ms. Czas wykonywania algorytmu sortowania przez wstawianie rośnie wraz ze wzrostem długości sortowanych ciągów.

Ostatni test został wykonany na ciągach losowych, a jego wyniki przestawione w podrozdziale 5.4. W tym przypadku korzystniej wypada sortowanie kopcowe. Wzrost czasu wykonywania tego algorytmu rośnie nieznacznie, nawet przy znacznym wzroście długości badanych ciągów. Przy badaniu sortowania przez wstawianie, zauważalny jest znaczny wzrostu czasu. Dla ciągu o długości 200000, czas wykonywania algorytmu wynosi aż 34,271s, przy czym czas sortowania kopcowego dla tej długości ciągu wynosi 0,065s. W omawianym przypadku, sortowanie przez wstawianie nie sprawdza się.

Czas sortowania kopcowego dla zbiorów posortowanych jest dłuższy niż dla zbioru posortowanego odwrotnie. To właśnie czas wykonywania algorytmu w przypadku zbioru posortowanego odwrotnie jest najkrótszy, można zatem przyjąć, że dla algorytmu sortowania kopcowego właśnie ten przypadek jest przypadkiem optymistycznym. Najdłużej trwa sortowanie zbioru nieposortowanego, jednak otrzymane czasy nie różnią się wyraźnie, dlatego można wnioskować, że ten algorytm nie jest zbyt czuły na postać danych wejściowych.

Algorytm sortowania przez wstawianie wykorzystuje fakt posortowania zbioru. W przypadku zbiorów uporządkowanych sortowanie nie wymaga dużo czasu. Najgorszym przypadkiem jest sortowanie zbioru posortowanego odwrotnie. Czas sortowanie zbiorów losowych, w porównaniu do czasu sortowanie kopcowego, jest dość długi.

# Spis ilustracji

[Kod sortowania przez wstawianie w c++ 3](#_Toc61017652)

[Schemat blokowy sortowania przez wstawianie 4](#_Toc61017653)

[Budowa kopca 6](file:///C:\Users\User\Desktop\projekt%20AiSD\Sprawozdanie.docx#_Toc61017654)

[Fragment rozbioru kopca 7](#_Toc61017655)

[Tworzenie kopca w c++ 8](#_Toc61017656)

[Główna funkcja sortowania kopcowego w c++ 8](#_Toc61017657)

[Schemat blokowy sortowania kopcowego 11](#_Toc61017658)

[Sortowanie przez wstawianie - dane z pliku – wyniki 19](#_Toc61017659)

[Sortowanie przez wstawianie - dane z pliku - wyniki w pliku tekstowym 19](#_Toc61017660)

[Sortowanie kopcowe - dane z pliku – wyniki 20](#_Toc61017661)

[Sortowanie kopcowe - dane z pliku – wyniki w pliku tekstowym 20](#_Toc61017662)

[Test\_1 - wyniki w pliku tekstowym 21](#_Toc61017663)

[Test\_1 - wyniki 10 prób - tabela z Excela cz.1 22](#_Toc61017664)

[Test\_1 - wyniki 10 prób – tabela z Excela cz.2. 22](#_Toc61017665)

[Test\_2 - wyniki w pliku tekstowym 23](#_Toc61017666)

[Test\_3 - wyniki w pliku tekstowym 24](#_Toc61017667)

[Test\_3 - wyniki 10 prób - tabela z Excela cz.1 25](#_Toc61017668)

[Test\_3 - wyniki 10 prób – tabela z Excela cz.2 25](#_Toc61017669)

# Załączniki

* Arkusz programu Excel "Test1\_wyniki"
* Arkusz programu Excel "Test3\_wyniki"

# Źródła

<http://www.algorytm.edu.pl/algorytmy-maturalne/sortowanie-przez-wstawianie.html>

<https://pl.wikipedia.org/wiki/Sortowanie_przez_wstawianie#Z%C5%82o%C5%BCono%C5%9B%C4%87>

<https://eduinf.waw.pl/inf/alg/003_sort/0010.php>

<https://eduinf.waw.pl/inf/alg/003_sort/0015.php>

<https://eduinf.waw.pl/inf/alg/003_sort/0017.php>

<https://www.geeksforgeeks.org/insertion-sort/>

<https://www.geeksforgeeks.org/heap-sort/>

1. <http://www.algorytm.edu.pl/algorytmy-maturalne/sortowanie-przez-wstawianie.html> [↑](#footnote-ref-1)