30.12.2020

Algorytmy i struktury danych

Projekt nr 2

Katarzyna Jadowska

154319

FS0-DI

gr. P3

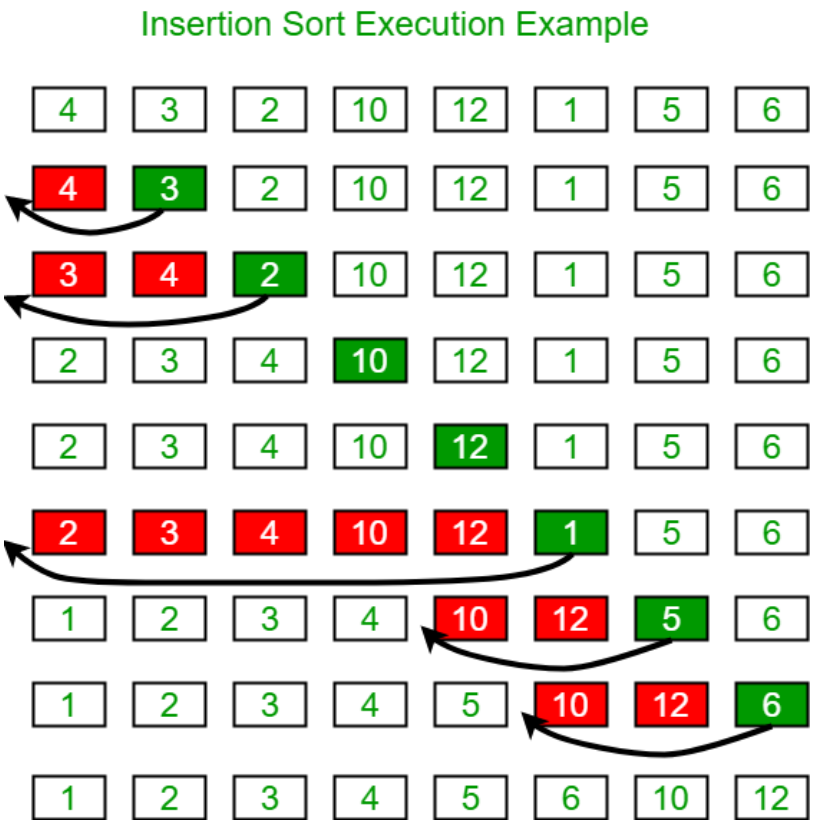
1. Wstęp

Celem projektu było stworzenie programu, który wykonuje sortowanie liczb naturalnych w oparciu o dwa algorytmy sortowania: przez wstawianie oraz kubełkowe. Następnie należało rozważyć działanie programu dla różnych przypadków i przedstawić złożoność czasową obu algorytmów.

2. Opis problemu

2.1. Sortowanie przez wstawianie

Sortowanie przez wstawianie jest prostym algorytmem sortowania, który działa podobnie do sposobu sortowania kart do gry w rękach [1]. Tablica jest praktycznie podzielona na część posortowaną i nieposortowaną. Wartości z nieposortowanej części są pobierane i umieszczane na właściwej pozycji w posortowanej części.

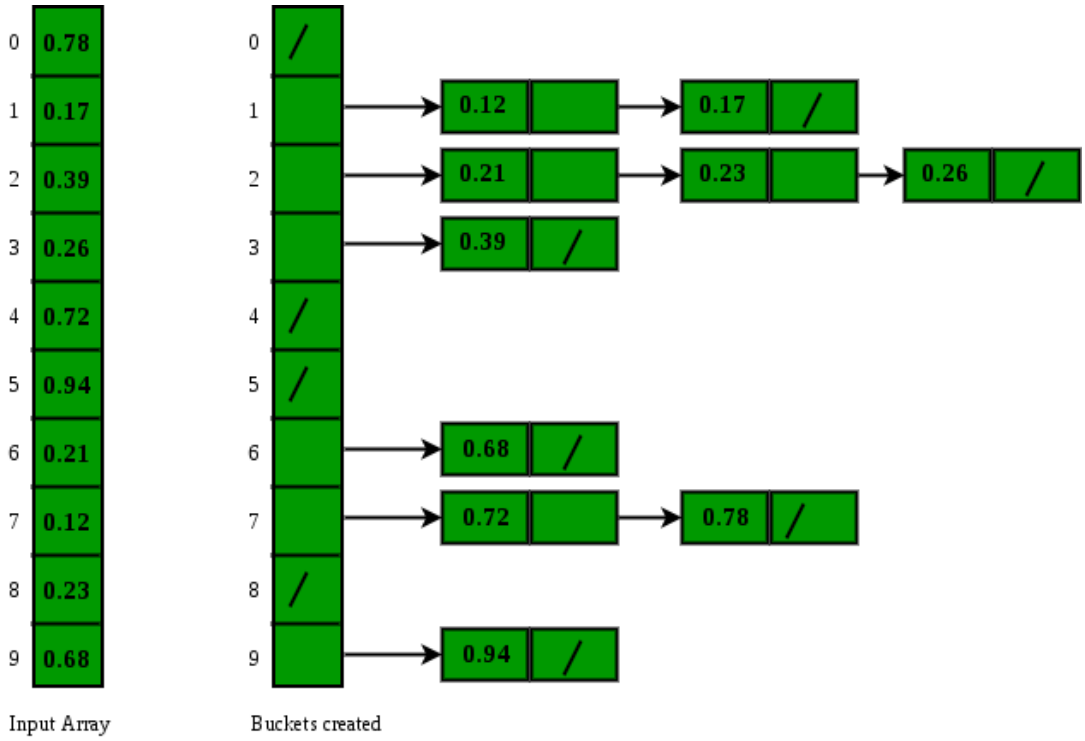


Rysunek 2.1. Przykład sortowania przez wstawianie [1]

Złożoność czasowa tego algorytmu sortowania wynosi w najgorszym przypadku $O(n^2)$. Taka złożoność wynika z zagnieżdżenia dwóch pętli for. W wewnętrznej pętli for znajduje się porównanie „if”, które przerywa pętlę, gdy wartość odniesienia jest większa od największego elementu posortowanej podlisty. W przypadku, gdybyśmy chcieli posortować już posortowaną tablicę, porównanie „if” zignoruje wewnętrzną pętlę for , jako że zostanie ono wykonane tylko raz. Zatem w najlepszym przypadku złożoność czasowa tego algorytmu wyniesie $O(n)$.

2.2. Sortowanie kubełkowe

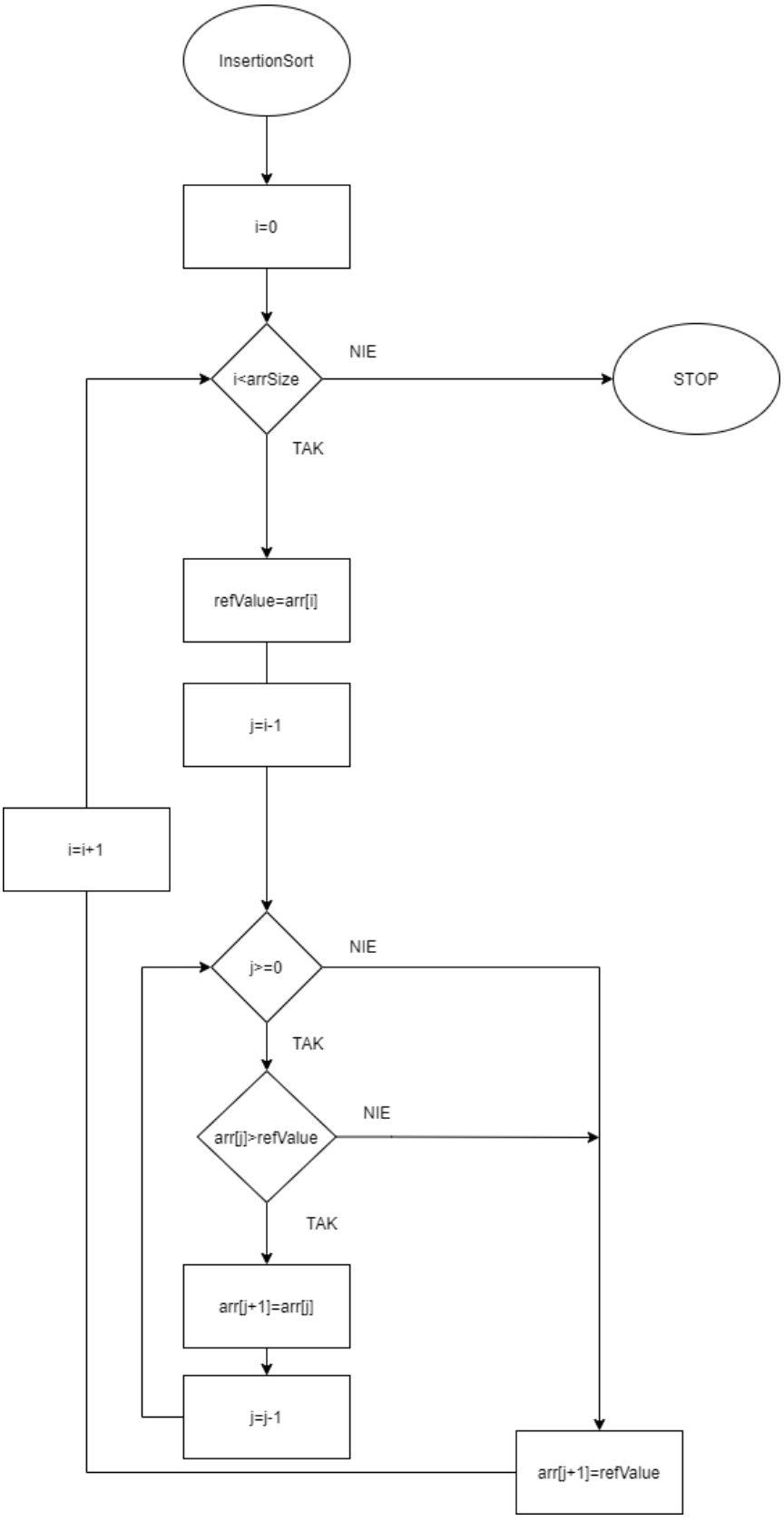
Sortowanie kubełkowe jest użyteczne głównie wtedy, gdy dane wejściowe są równomiernie rozłożone w pewnym zakresie [2].



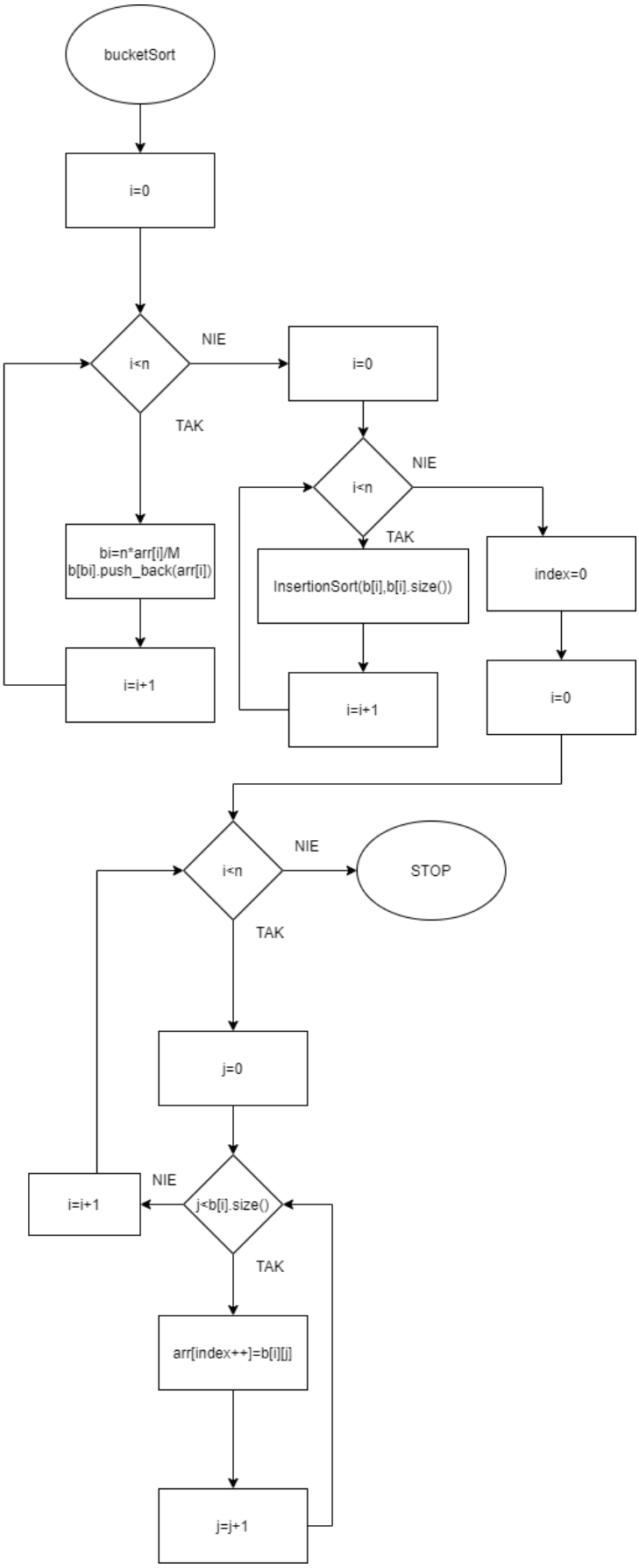
Rysunek 2.2. Zobrazowanie sortowania kubełkowego na przykładzie liczb zmiennoprzecinkowych z zakresu 0,1-1,0 [2]

Najgorszym przypadkiem jest taki, w którym wszystkie elementy trafią do jednego kubełka, złożoność takiego algorytmu wynosi wtedy $O(n^2)$ (przy założeniu wykorzystania algorytmu o takiej właśnie złożoności do sortowania elementów w kubełku). Średnia złożoność wynosi $O(n + k)$, gdzie k jest liczbą utworzonych kubełków, taką złożoność uzyskujemy dla równo rozłożonych elementów w kubełkach. Najlepszym przypadkiem jest sytuacja, w której do każdego kubełka trafia tylko jeden element lub gdy trafiające elementy są już posortowane, wystarczy wtedy jedynie połączyć kubełki w całość.

3. Schematy blokowe



Rysunek 3.1. Schemat blokowy sortowania przez wstawianie



Rysunek 3.2. Schemat blokowy sortowania kubełkowego

4. Pseudokod

4.1. Sortowanie przez wstawianie

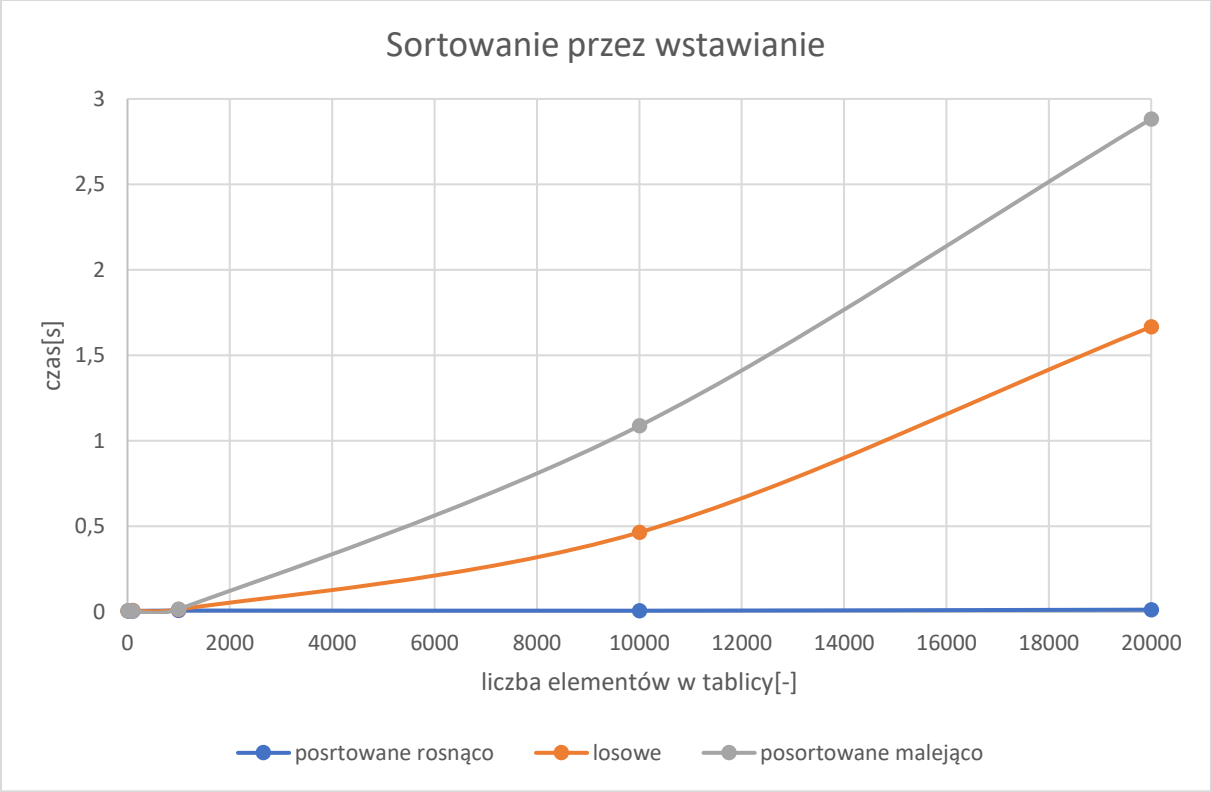
Aby posortować tablicę o rozmiarze n w porządku rosnącym:

1) Iteruj od arr [1] do arr [n] po tablicy.
2) Porównaj bieżący element (klucz) z jego poprzednikiem.
3) Jeśli klucz jest mniejszy niż jego poprzednik, porównaj go z poprzednimi elementami. Przesuń większe elementy o jedną pozycję w górę, aby zrobić miejsce na zamieniony element.

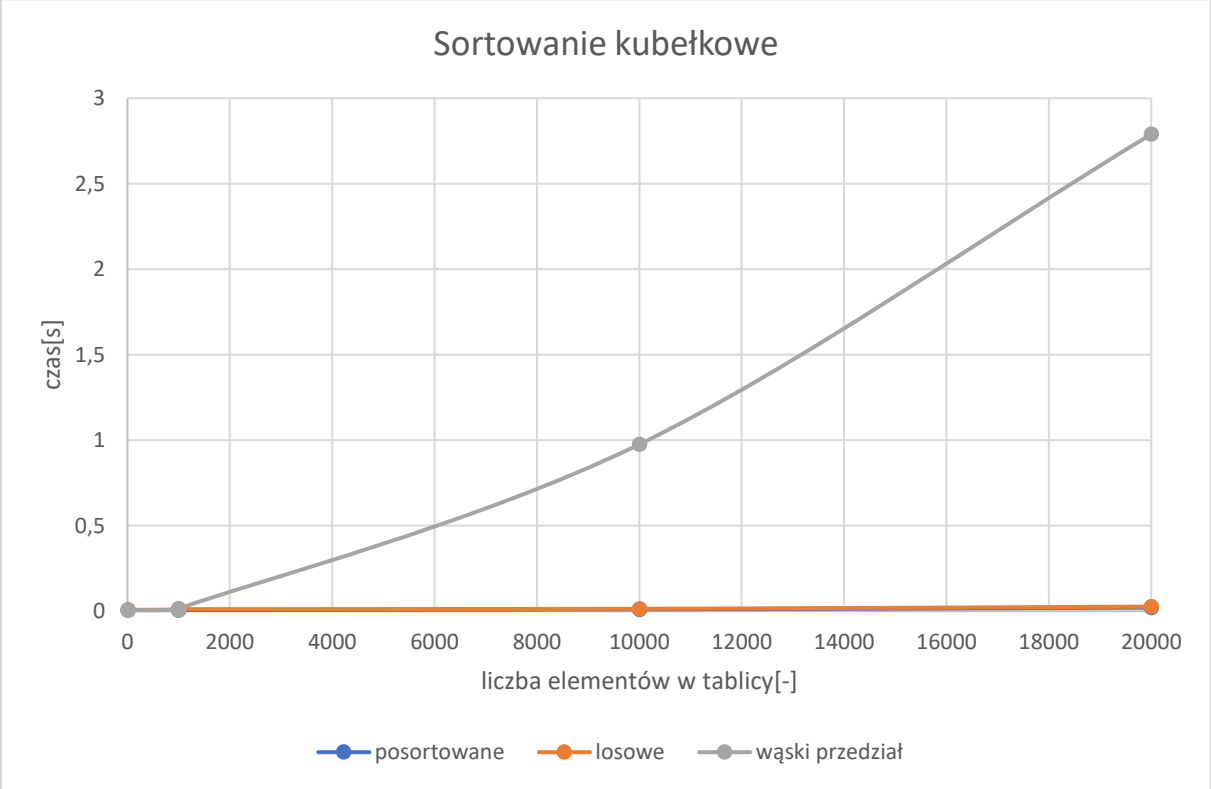
4.2. Sortowanie kubełkowe

1) Stwórz n pustych kubełków, gdzie n- liczba elementów do posortowania.
2) Dla każdego elementu arr[i] wykonuj:
    a) Wstaw i-ty element tablicy arr[i] do kubełka bucket[n*arr[i]/M], gdzie M- maksymalna wartość pojedynczego elementu w tablicy
3) Sortuj pojedyncze kubełki używając sortowania przez wstawianie.
4) Połącz wszystkie posortowane kubełki.

5. Dokumentacja z doświadczeń



Rysunek 5.1. Złożoność czasowa algorytmu sortowania przez wstawianie



Rysunek 5.2. Złożoność czasowa algorytmu sortowania kubełkowego

6. Wnioski

Udało się zrealizować cel ćwiczenia. Zaimplementowano algorytmy sortowania przez wstawianie oraz kubełkowego. W doświadczeniu rozważono zakres wartości liczb od 0 do 20000 oraz tablice o różnej pojemności z zakresu od 10 do 20000 elementów.

Na podstawie analizy wykresu 5.1, który przedstawia sortowanie przez wstawianie widoczne jest, że najlepsze wyniki osiągane są przy tablicy posortowanej rosnąco. Jest to przypadek, w którym wszystkie liczby są już posortowane, co oznacza, że nie jest wykonywane przesuwanie elementów w tablicy. Najgorszy przypadek pod względem złożoności czasowej uzyskano dla liczb posortowanych malejąco. W takiej sytuacji żadna z liczb nie znajduje się na swojej docelowej pozycji, co oznacza konieczność przestawiania każdego elementu. Ze względu na zagnieżdżenie dwóch pętli for złożoność wynosi $O(n^2)$. Za przypadek środkowy przyjęto sytuację, w której losowane są liczby z zadanego zakresu.

Na podstawie wykresu 5.2 dla sortowania kubełkowego widoczne jest, że przypadek z losowaniem liczb oraz najkorzystniejszy praktycznie nie różnią się od siebie. Wyraźna różnica widoczna jest dla przypadku najmniej korzystnego, w którym wszystkie liczby znajdują się w wąskim przedziale. W tym przypadku wszystkie liczby trafiają do jednego kubełka, w którym wykonywane jest sortowanie przez wstawianie. Liczby te uporządkowane są malejąco, co oznacza, że osiągany jest najgorszy przypadek dla sortowania przez wstawienie. W przypadku sortowania kubełkowego istotną rolę pełni algorytm wykorzystywany do sortowania pojedynczych kubełków, używając algorytmu o mniejszej złożoności można poprawić efektywność całego sortowania.

Porównując przypadek, w którym liczby są losowane widoczne jest, że dużo lepiej wypada algorytm sortowania kubełkowego. W celu lepszego porównania, w obu przypadkach wykorzystano ten sam zestaw liczb. Na tej podstawie można stwierdzić, że w przypadku znanego zakresu liczb bardziej efektywnym rozwiązaniem jest wykorzystanie sortowania kubełkowego.

Na podstawie wykresów 5.1 oraz 5.2 widoczne jest, że dla tablicy o niewielkiej pojemności, do ok.1000 elementów, nie ma większego znaczenia, który algorytm wybierzemy, ponieważ czas ich wykonywania jest zbliżony dla wszystkich przypadków.

Bibliografia

[1] „C++. Struktury danych i algorytmy”, Wisnu Anggoro

[2] https://www.geeksforgeeks.org/bucket-sort-2/, dostęp: 05.01.2021

Kod programu

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <fstream>
#include<time.h>
#include <ctime>
#include <cstdio>


using namespace std;



double obliczSekundy( clock_t czas )
{
   //wyznaczanie czasu w sekundach
   return static_cast < double >( czas ) / CLOCKS_PER_SEC;
}

//wprowadzanie liczb losowych
void Wprowadzanie(const char*nazwa, int n, int M)
{


   ofstream zapis1(nazwa);
   int *Z= new int[n];

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


      Z[i]=rand()%20000+1;
      zapis1<<Z[i]<<" ";
   }
   zapis1.close();
   delete [] Z;
}
//wprowadzanie liczb posortowanych rosnąco
void WprowadzanieSort(const char*nazwa, int n, int M)
{
```

```
    ofstream zapis1(nazwa);
    int *Z= new int[n];

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


        Z[i]=i;
        zapis1<<Z[i]<<" ";
    }
    zapis1.close();
    delete [] Z;
}


//posortowane malejąco
void WprowadzanieWaski(const char*nazwa, int n, int M)
{
    int j=0;

    ofstream zapis1(nazwa);
    int *Z= new int[n];

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


        Z[i]=20000-j;
        j++;
        zapis1<<Z[i]<<" ";
    }
    zapis1.close();
    delete [] Z;
}


//pdczytywanie licz z pliku txt

void Odczyt_txt(const char* nazwa,int &n,vector<int>&arr)
{

    ifstream zrodlo(nazwa);
    int liczba=0;
```

```
    while(zrodlo>>liczba)
        arr.push_back(liczba);

    n=arr.size();



}
//zapisywanie do pliku txt
void Zapisz(const char*nazwa, int arrSize, vector<int>arr)
{

    ofstream plik(nazwa);

    for(int i=0; i<arrSize; i++)
        plik<<arr[i]<<" ";

}
//sortowanie przez wstawianie
void InsertionSort(vector<int> &arr, int arrSize)
{
 //iteracja po wszystkich elementach tablicy
    for(int i=0; i<arrSize; i++)
    {
        //biezacy element tablicy- wartosc odniesienia
        int refValue=arr[i];

        //zmienna do przenoszenia elementow na wlasciwa pozycje
        int j;

        //iteracja po posortowanych elementach w celu umieszenia wartosci odniesiena
        // na wlasciwej pozycji
        for(j=i-1; j>=0; j--)
        {
            //jesli wartosc biezacego elementu tablicy jest wieksza od wartossci
            //odniesienia to przeniosi wartosc biezaca na prawa strone
            // w przeciwnym przypadku przerwanie petli
            if(arr[j]>refValue)
                arr[j+1]=arr[j];
            else
                break;
        }
```

```
        //umieszczenie wartosci odniesienia w biezacym indeksie, na wlasciwej pozycji
        arr[j+1]=refValue;


    }
}
//sortowanie kubelkowe
void bucketSort(vector<int> &arr, int n,int M)
{
    // tworzenie n pustych kubelkow
    vector<int> b[n];


    // umieszczanie elementów tablicy w odpowiednich kubelkach
    for (int i = 0; i < n; i++)
    {
        int bi = n * arr[i]/M; // Index in bucket
        b[bi].push_back(arr[i]);
    }

    // sortowanie pojedynczych kubelkow z wykorzystaniem sortowania przez wstawianie
    for (int i = 0; i < n; i++)
        InsertionSort(b[i],b[i].size());


    // polaczenie wszystkich kubelkow w jedna tablice arr[]
    int index = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < b[i].size(); j++)
            arr[index++] = b[i][j];
}

int main()
{
    //zakres liczb uwzgledniony w sortowaniu (od 0 do 20000)
    int M=20000;
    srand ( ( unsigned )time ( NULL ) );


    vector<int>arr;
    int n;

    //Testy
```

```
//losowe liczby, przypadek sredni

cout<<"losowe"<<endl;
Wprowadzanie("Z10.txt",10,M);
Odczyt_txt("Z10.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
InsertionSort(arr,n);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
arr.clear();
Odczyt_txt("Z10.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
bucketSort(arr,n,M);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
Zapisz("WynikiZ10.txt",n,arr);

arr.clear();

cout<<endl<<endl;

Wprowadzanie("Z100.txt",100,M);
Odczyt_txt("Z100.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
InsertionSort(arr,n);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
arr.clear();
Odczyt_txt("Z100.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
bucketSort(arr,n,M);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
Zapisz("WynikiZ100.txt",n,arr);

arr.clear();

cout<<endl<<endl;

Wprowadzanie("Z1000.txt",1000,M);
Odczyt_txt("Z1000.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
InsertionSort(arr,n);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
arr.clear();
Odczyt_txt("Z1000.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
bucketSort(arr,n,M);
```

```
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    Zapisz("WynikiZ1000.txt",n,arr);

    arr.clear();
    cout<<endl<<endl;

    Wprowadzanie("Z10000.txt",10000,M);
    Odczyt_txt("Z10000.txt",n,arr);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    InsertionSort(arr,n);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    arr.clear();
    Odczyt_txt("Z10000.txt",n,arr);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    bucketSort(arr,n,M);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    Zapisz("WynikiZ10000.txt",n,arr);

    arr.clear();
    cout<<endl<<endl;

    Wprowadzanie("Z20000.txt",20000,M);
    Odczyt_txt("Z20000.txt",n,arr);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    InsertionSort(arr,n);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    arr.clear();
    Odczyt_txt("Z20000.txt",n,arr);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    bucketSort(arr,n,M);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    Zapisz("WynikiZ20000.txt",n,arr);

    arr.clear();
    cout<<endl<<endl;

//posortowane - najlepszy przypadek bucket sort i insertion sort

    cout<<"posortowane"<<endl;

    WprowadzanieSort("S10.txt",10,M);
    Odczyt_txt("S10.txt",n,arr);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    InsertionSort(arr,n);
```

```
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
arr.clear();
Odczyt_txt("S10.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
bucketSort(arr,n,M);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
Zapisz("WynikiS10.txt",n,arr);

arr.clear();
cout<<endl<<endl;

WprowadzanieSort("S100.txt",100,M);
Odczyt_txt("S100.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
InsertionSort(arr,n);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
arr.clear();
Odczyt_txt("S100.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
bucketSort(arr,n,M);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
Zapisz("WynikiS100.txt",n,arr);

arr.clear();
cout<<endl<<endl;

WprowadzanieSort("S1000.txt",1000,M);
Odczyt_txt("S1000.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
InsertionSort(arr,n);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
arr.clear();
Odczyt_txt("S1000.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
bucketSort(arr,n,M);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
Zapisz("WynikiS1000.txt",n,arr);

arr.clear();
cout<<endl<<endl;

WprowadzanieSort("S10000.txt",10000,M);
Odczyt_txt("S10000.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
```

```
    InsertionSort(arr,n);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    arr.clear();
    Odczyt_txt("S10000.txt",n,arr);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    bucketSort(arr,n,M);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    Zapisz("WynikiS10000.txt",n,arr);

    arr.clear();
    cout<<endl<<endl;


    WprowadzanieSort("S20000.txt",20000,M);
    Odczyt_txt("S20000.txt",n,arr);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    InsertionSort(arr,n);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    arr.clear();
    Odczyt_txt("S20000.txt",n,arr);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    bucketSort(arr,n,M);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    Zapisz("WynikiS20000.txt",n,arr);

    arr.clear();
    cout<<endl<<endl;



//najgorszy przypadek bucket sort, najogrszy przypadek insertion sort

    cout<<"waski przedial, posortowane malejaco:"<<endl;
    //sztucznie zawyzona liczba elemtnow zeby wszystkie trafily do jednego kubelka
     M=300000001;
    //liczby w waskim przedziale
    WprowadzanieWaski("W10.txt",10,M);
    Odczyt_txt("W10.txt",n,arr);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    InsertionSort(arr,n);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    arr.clear();
    Odczyt_txt("W10.txt",n,arr);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
```

```
bucketSort(arr,n,M);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
Zapisz("WynikiW10.txt",n,arr);

arr.clear();
cout<<endl<<endl;

WprowadzanieWaski("W100.txt",100,M);
Odczyt_txt("W100.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
InsertionSort(arr,n);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
arr.clear();
Odczyt_txt("W100.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
bucketSort(arr,n,M);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
Zapisz("W100.txt",n,arr);

arr.clear();
cout<<endl<<endl;

WprowadzanieWaski("W1000.txt",1000,M);
Odczyt_txt("W1000.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
InsertionSort(arr,n);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
arr.clear();
Odczyt_txt("W1000.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
bucketSort(arr,n,M);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
Zapisz("WynikiW1000.txt",n,arr);

arr.clear();
cout<<endl<<endl;
WprowadzanieWaski("W10000.txt",10000,M);
Odczyt_txt("W10000.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
InsertionSort(arr,n);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
arr.clear();
Odczyt_txt("W10000.txt",n,arr);
printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
```

```
    bucketSort(arr,n,M);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    Zapisz("WynikiW10000.txt",n,arr);

    arr.clear();
    cout<<endl<<endl;

    WprowadzanieWaski("W20000.txt",20000,M);
    Odczyt_txt("W20000.txt",n,arr);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    InsertionSort(arr,n);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    arr.clear();
    Odczyt_txt("W20000.txt",n,arr);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    bucketSort(arr,n,M);
    printf( "Uplynelo %.4fsek od startu aplikacji.\n", obliczSekundy( clock() ) );
    Zapisz("WynikiW20000.txt",n,arr);

    arr.clear();
    cout<<endl<<endl;


    return 0;
}
```