POLITECHNIKA RZESZOWSKA

Wydział Matematyki i Fizyki Stosowanej

Inżynieria i analiza danych

ALGORYTMY i STRUKTURY DANYCH

PROJEKT 2

Paulina Kmieć

Grupa nr. 03

Nr. indeksu 166657

Spis treści

1. Treść zadania

Zaimplementuj sortowanie przez wstawianie oraz sortowanie przez zliczanie.

2. Kod programu

```cpp
#include <iostream>
#include <fstream>
#include <time.h>
#include <chrono>
using namespace std;

int  N = 2000;            //liczby bêd¹ generowane z zakresu 0 - N-1
int  ROZMIAR = 100000;    //rozmiar sortowanej tablicy

int tabN[6] = { 1000 , 10000, 50000, 100000, 500000, 5000000 };

void wygenerujLiczbyDoPliku()
{
    int liczba;
    ofstream plik("liczbyLosowe.txt");

    //w pêtli generujê liczby losowe
    for (int i = 0; i < ROZMIAR; i++)
    {
        liczba = rand() % N;
        //zapisujê do pliku liczby oddzielone spacj¹
        plik << liczba << " ";
    }

    //zamykam plik
    plik.close();
}

void zapiszTabliceDoPliku(int* tablica)
{
    ofstream plik("posortowane.txt");

    for (int i = 0; i < ROZMIAR; i++)
    {
        plik << tablica[i] << " ";
    }

    plik.close();
}

void wczytajTablice(int* tablica)
{
    ifstream plik("liczbyLosowe.txt");

    //w pêtli zczytujê kolejne elementy tablicy
    for (int i = 0; i < ROZMIAR; i++)
    {
        plik >> tablica[i];
    }
```

```cpp
    }

    //zamykam plik
    plik.close();
}

void sortowaniePrzezWstawianie(int* tablica)
{
    for (int i = 1; i < ROZMIAR; i++)
    {
        int klucz = tablica[i];
        int j = i - 1;

        while (j >= 0 && tablica[j] > klucz)
        {
            tablica[j + 1] = tablica[j];
            j--;
        }
        tablica[j + 1] = klucz;
    }
}

void sortowaniePrzezZliczanie(int* tablica)
{
    //alokuje pamiêæ do tablicy pomocniczej
    int* posortowana = new int[ROZMIAR];

    //alokuje miejsce na tablice przechowujaca zliczenia poszczególnych liczb
    int* liczniki = new int[N+1];
    for (int i = 0; i < N; i++)
        liczniki[i] = 0;

    //zliczam wyst¹pienia liczb w tablicy 'liczniki'
    for (int i = 0; i < ROZMIAR; i++)
        liczniki[tablica[i]]++;

    //obliczamy pozycje koñcowe elementów
    for (int i = 1; i <= N; ++i)
        liczniki[i] += liczniki[i - 1];

    //przepisujê elementy do tablicy wynikowej
    for (int i = 0; tablica[i]; i++)
    {
        int index = liczniki[tablica[i]] - 1;
        posortowana[index] = tablica[i];
        liczniki[tablica[i]]--;
    }

    for (int i = 0; i < ROZMIAR; i++)
```

```
    for (int i = 0; i < ROZMIAR; i++)
        tablica[i] = posortowana[i];

    //zwalniam dynamicznie zaalokowan¹ pamiêæ
    delete[] posortowana;
    delete[] liczniki;
}

//funkcja pomocnicza do wypisywania zawartoœci tablicy
void wypiszTablice(int* tablica)
{
    for (int i = 0; i < ROZMIAR; i++)
    {
        cout << tablica[i] << endl;
    }
}

void testDaneLosowe(int *tablica)
{
    cout << "DANE LOSOWE\n";

    for (int i = 0; i < 6; i++)
    {
        N = tabN[i];

        //zliczam czas sortowanie przez zliczanie
        wygenerujLiczbyDoPliku();
        wczytajTablice(tablica);
        auto start = chrono::high_resolution_clock::now();
        sortowaniePrzezZliczanie(tablica);
        auto stop = chrono::high_resolution_clock::now();
        auto czasSortowaniePrzezZliczanie = chrono::duration_cast<std::chrono::microseconds>(stop - start).count();

        //zliczam czas sortowania przez wstawianie
        wygenerujLiczbyDoPliku();
        wczytajTablice(tablica);
        start = chrono::high_resolution_clock::now();
        sortowaniePrzezWstawianie(tablica);
        stop = chrono::high_resolution_clock::now();
        auto czasSortowaniePrzezWstawianie = chrono::duration_cast<std::chrono::microseconds>(stop - start).count();

        cout << "N = " << N << "S. przez zliczanie " << czasSortowaniePrzezZliczanie << "ms S. przez wstawianie " << czasSortowaniePrzezWstawianie << "ms\n";
    }
}

void testDanePosortowane(int* tablica)
{
    cout << "\n\nDANE POSORTOWANE\n";
```

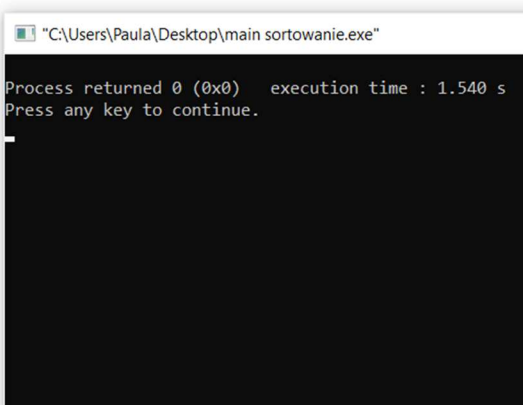
```
    for (int i = 0; i < 6; i++)
    {
        N = tabN[i];

        wygenerujLiczbyDoPliku();
        wczytajTablice(tablica);
        sortowaniePrzezWstawianie(tablica);

        //zliczam czas sortowanie przez zliczanie
        auto start = chrono::high_resolution_clock::now();
        sortowaniePrzezZliczanie(tablica);
        auto stop = chrono::high_resolution_clock::now();
        auto czasSortowaniePrzezZliczanie = chrono::duration_cast<std::chrono::microseconds>(stop - start).count();

        //zliczam czas sortowania przez wstawianie
        start = chrono::high_resolution_clock::now();
        sortowaniePrzezWstawianie(tablica);
        stop = chrono::high_resolution_clock::now();
        auto czasSortowaniePrzezWstawianie = chrono::duration_cast<std::chrono::microseconds>(stop - start).count();

        cout << "N = " << N << "S. przez zliczanie " << czasSortowaniePrzezZliczanie << "ms S. przez wstawianie " << czasSortowaniePrzezWstawianie << "ms\n";
    }
}

void sortujOdwrotnie(int* tablica)
{
    for (int  i = 0; i < ROZMIAR; i++)
        for (int j = 0; j < ROZMIAR; j++)
        {
            if (tablica[i] > tablica[j])
                swap(tablica[i], tablica[j]);
        }
}

void testDanePosortowaneOdwrotnie(int* tablica)
{
    cout << "\n\nDANE POSORTOWANE ODWROTNIE\n";

    for (int i = 0; i < 6; i++)
    {
        N = tabN[i];

        wygenerujLiczbyDoPliku();
        wczytajTablice(tablica);

        sortujOdwrotnie(tablica);
        //zliczam czas sortowanie przez zliczanie
        auto start = chrono::high_resolution_clock::now();
        sortowaniePrzezZliczanie(tablica);
```

```
        auto stop = chrono::high_resolution_clock::now();
        auto czasSortowaniePrzezZliczanie = chrono::duration_cast<std::chrono::microseconds>(stop - start).count();

        sortujOdwrotnie(tablica);
        //zliczam czas sortowania przez wstawianie
        start = chrono::high_resolution_clock::now();
        sortowaniePrzezWstawianie(tablica);
        stop = chrono::high_resolution_clock::now();
        auto czasSortowaniePrzezWstawianie = chrono::duration_cast<std::chrono::microseconds>(stop - start).count();

        cout << "N = " << N << "S. przez zliczanie " << czasSortowaniePrzezZliczanie << "ms S. przez wstawianie " << czasSortowaniePrzezWstawianie << "ms\n";
    }
}
int main()
{


    //alokuje dynamicznie pamięć na tablicę
    int* tablica = new int[ROZMIAR];

    wczytajTablice(tablica);

    //wypiszTablice(tablica);

    //zapiszTabliceDoPliku(tablica);

    //testDaneLosowe(tablica);
    //testDanePosortowane(tablica);
    testDanePosortowaneOdwrotnie(tablica);
    return 0;
}
```

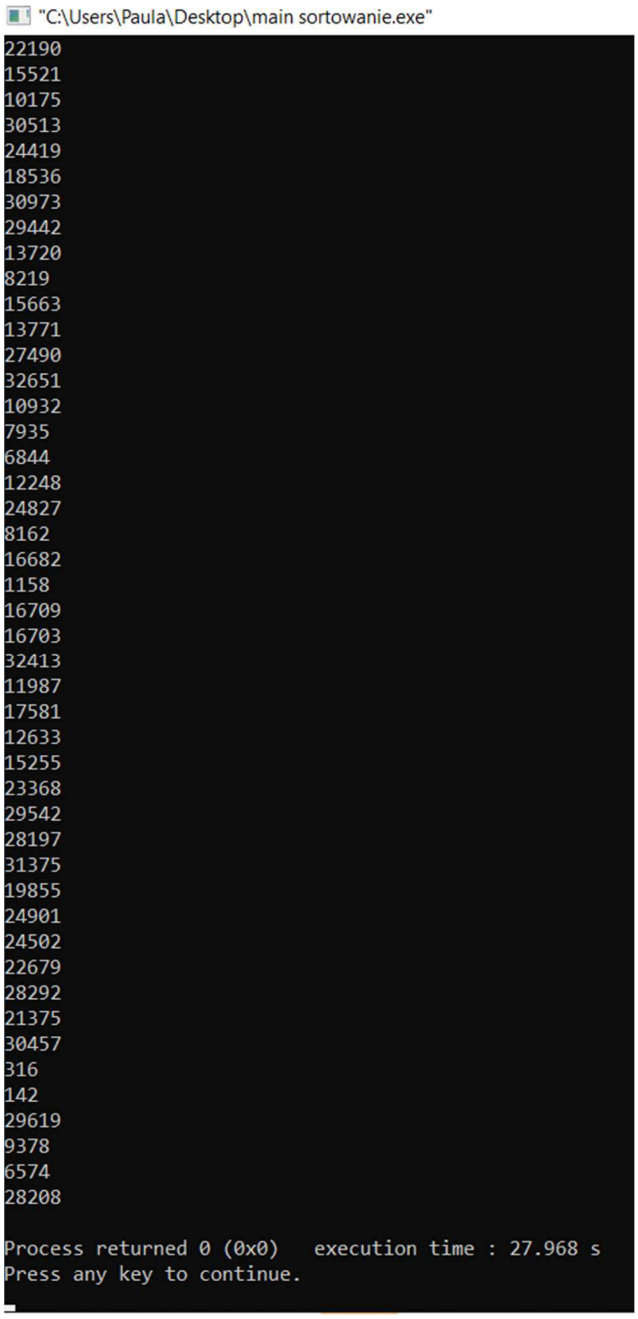
3. Działanie programu

Wczytuje tablicę

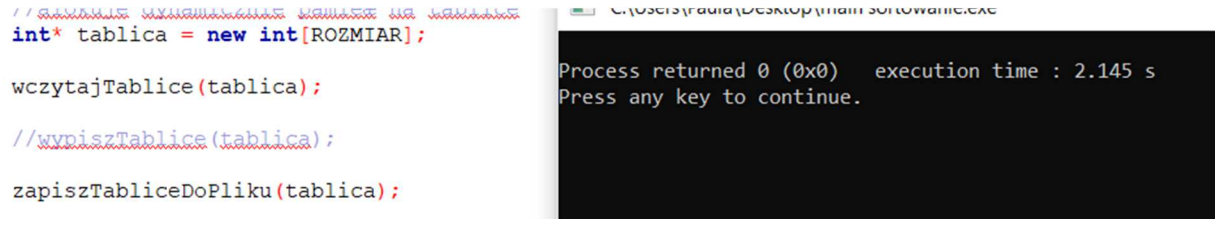
```
int main()
{


    //alokuje dynamicznie pamięć na tablicę
    int* tablica = new int[ROZMIAR];

    wczytajTablice(tablica);

    //wypiszTablice(tablica);

    //zapiszTabliceDoPliku(tablica);

    //testDaneLosowe(tablica);
    //testDanePosortowane(tablica);
    //testDanePosortowaneOdwrotnie(tablica);
    return 0;
}
```

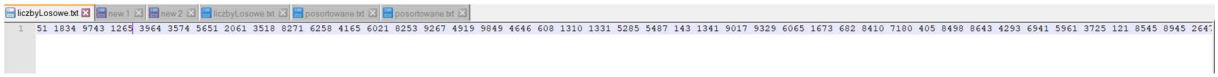


Wypisuje wczytaną tablicę



Zapisuje tablicę do pliku





Dane posortowane:

```
int main()
{


    //alokuje dynamicznie pamiêæ na tablicê
    int* tablica = new int[ROZMIAR];

    wczytajTablice(tablica);

    //wypiszTablice(tablica);

    //zapiszTabliceDoPliku(tablica);

    //testDaneLosowe(tablica);
    testDanePosortowane(tablica);
    //testDanePosortowaneOdwrotnie(tablica);
    return 0;
}
```

```
"C:\Users\Paula\Desktop\main sortowanie.exe"


DANE POSORTOWANE
N = 1000S. przez zliczanie 0ms S. przez wstawianie 2995ms
N = 10000S. przez zliczanie 0ms S. przez wstawianie 1000ms
N = 50000S. przez zliczanie 0ms S. przez wstawianie 1001ms
N = 100000S. przez zliczanie 0ms S. przez wstawianie 999ms
N = 500000S. przez zliczanie 2059ms S. przez wstawianie 1021ms
N = 5000000S. przez zliczanie 26006ms S. przez wstawianie 1001ms

Process returned 0 (0x0)   execution time : 32.931 s
Press any key to continue.
```

Dane posortowane odwrotnie:

```
int main()
{


    //alokuje dynamicznie pamiêæ na tablicê
    int* tablica = new int[ROZMIAR];

    wczytajTablice(tablica);

    //wypiszTablice(tablica);

    //zapiszTabliceDoPliku(tablica);

    //testDaneLosowe(tablica);
    //testDanePosortowane(tablica);
    testDanePosortowaneOdwrotnie(tablica);
    return 0;
}
```

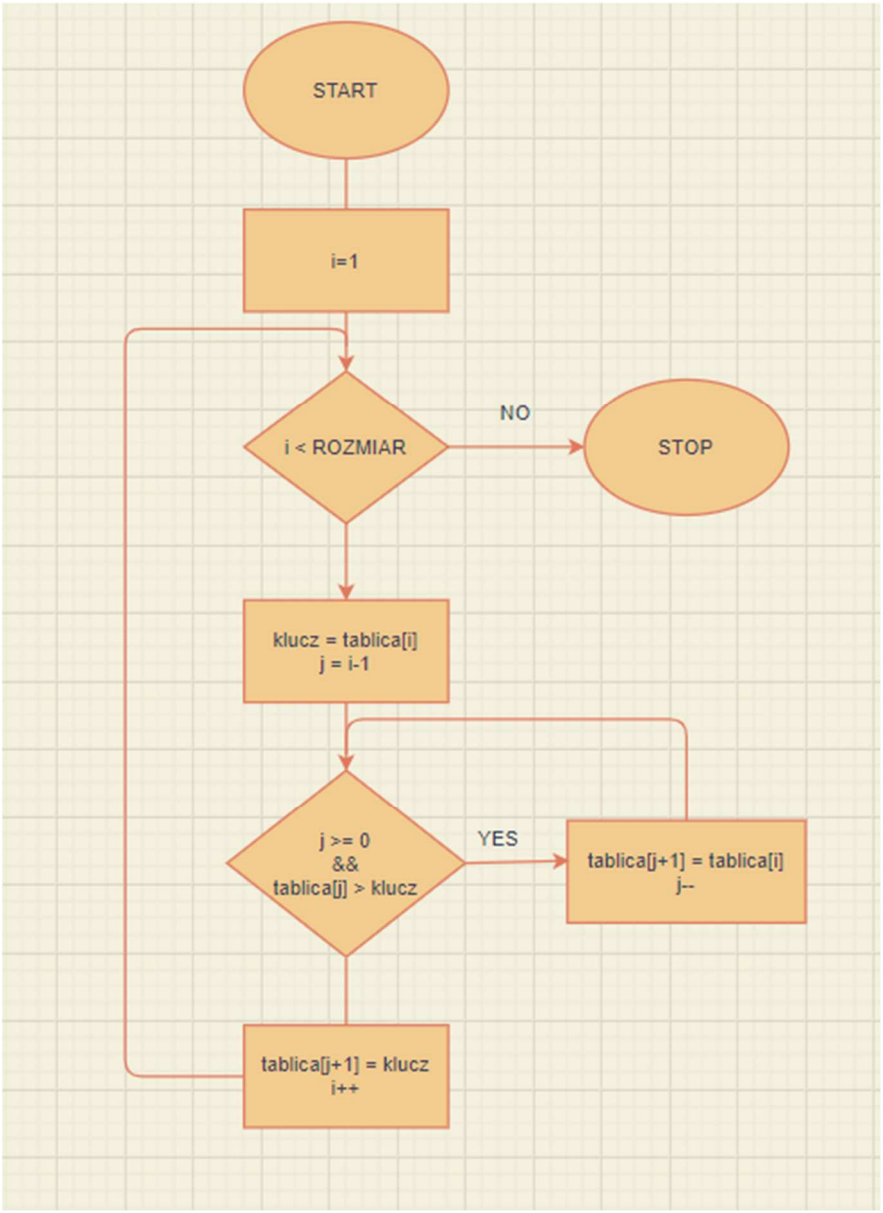
```
"C:\Users\Paula\Desktop\main sortowanie.exe"


DANE POSORTOWANE ODWROTNIE
N = 1000S. przez zliczanie 1007ms S. przez wstawianie 9403188ms
N = 10000S. przez zliczanie 1000ms S. przez wstawianie 9446739ms
N = 50000S. przez zliczanie 1000ms S. przez wstawianie 9411228ms
N = 100000S. przez zliczanie 2000ms S. przez wstawianie 9464426ms
N = 500000S. przez zliczanie 3001ms S. przez wstawianie 9453479ms
N = 5000000S. przez zliczanie 25001ms S. przez wstawianie 9445927ms

Process returned 0 (0x0)   execution time : 374.441 s
Press any key to continue.
```

Pomiary czasu w jednostce ms

losowe		
N	wstawianie	zliczanie
1000	7800279	742
10000	7754039	662
50000	7757442	1045
100000	7778710	1414
500000	7756415	5577
5000000	7754493	44158
posortowane		
N	wstawianie	zliczanie
1000	444	670
10000	475	610
50000	444	908
100000	441	1326
500000	468	4835
5000000	430	43949
posortowane	odwrotnie	
N	wstawianie	zliczanie
1000	15475229	1229
10000	16131722	1140
50000	15490694	1499
100000	15604505	1951
500000	15509255	5278
5000000	15580538	45214

4. Sortowanie przez wstawianie

4.1 Schemat blokowy



4.2 Pseudokod

```
for i=1 to ROZMIAR:
    Wstaw tablica[i] w posortowany ciąg tablica[1 ... i-1]
    klucz =tablica[i]
    j = i-1
    while j>=0 and tablica[j] > klucz :
    tablica[j+1] = tablica[j]
    j = j-1
    tablica[j+1] = klucz
```

Algorytm dla każdego elementu na liście do posortowania sprawdza gdzie jest jego miejsce na liście posortowanej. Robi to poprzez porównywanie pobranego elementu z kolejnymi elementami na liście wynikowej. Element jest wstawiany kiedy spełni warunek porządkowania lub kiedy nie ma już więcej elementów w celu porównania (wtedy wstawiamy na koniec).

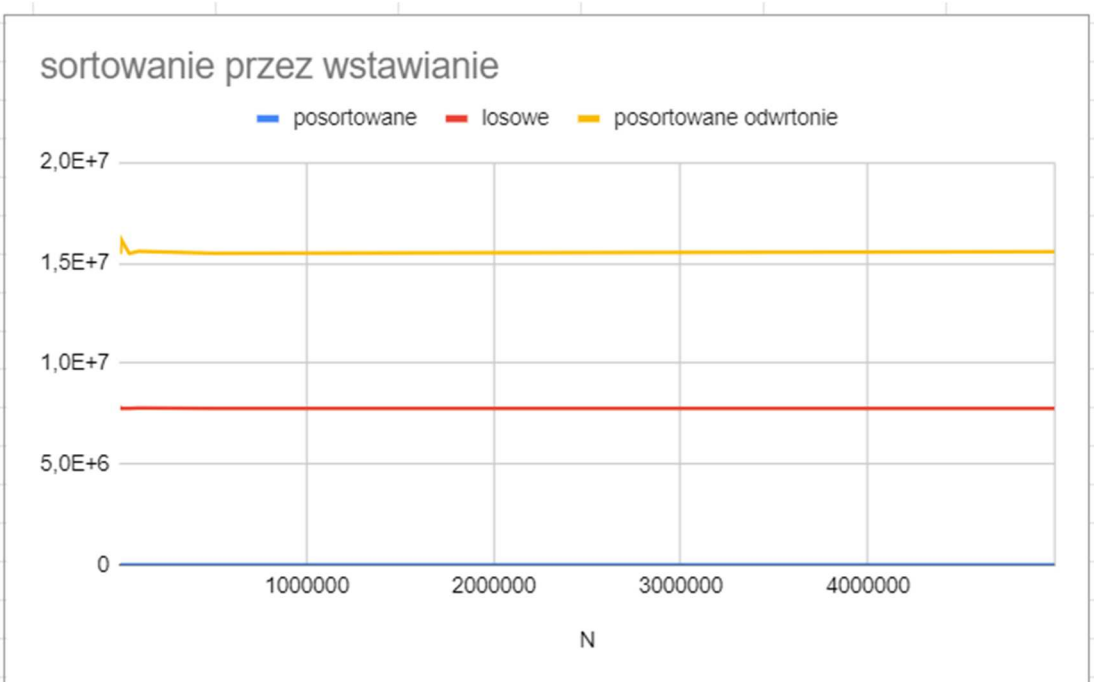
4.3 Złożoność obliczeniowa:

Dla każdego elementu w *i*-tym kroku możemy wykonać maksymalnie *i* porównań. Sumując wszystkie porównanie dla *n* elementów otrzymujemy 1+2+…+n-1=$\frac{n(n-1)}{2}$ porównań, dlatego złożoność czasowa algorytmu jest rzędu O(n^2).

4.4. Pomiary czasu w jednostce ms

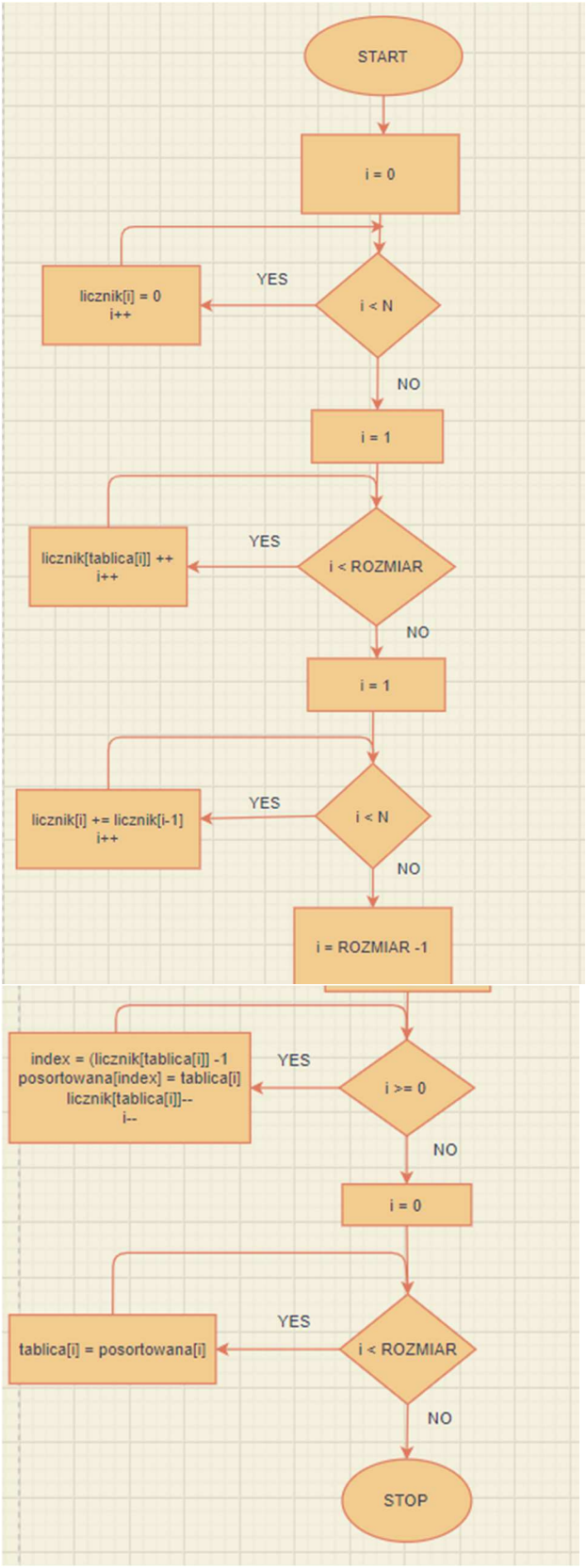
wstawianie			
N	posortowane	losowe	posortowane odwrtonie
1000	444	7800279	15475229
10000	475	7754039	16131722
50000	444	7757442	15490694
100000	441	7778710	15604505
500000	468	7756415	15509255
5000000	430	7754493	15580538

4.5 Złożoność czasowa przedstawiona na wykresie



5. Sortowanie przez zliczanie

5.1 Schemat blokowy



5.2 Pseudokod

```
    posortowana = new int[ROZMIAR]
for i=0 to N
    liczniki[i] = 0
for i=0 to ROZMIAR
    liczniki[tablica[i]]++
for i=0 to N
    liczniki[i]+= liczniki[i-1]
for i=0 tablica[i]
    index = liczniki[tablica[i]]-1
    posortowana[index] = tablica[i]
    liczniki[tablica[i]]--
for i=0 to ROZMIAR
    tablica[i] = posortowana[i]
delete[]posortowana
delete[]liczniki
```

Sortowanie przez zliczanie polega na zadeklarowaniu dodatkowej tablicy do której będziemy zliczać wystąpienia poszczególnych wartości, np. ile mamy na liście 0, 1 ... Następny krok polega na analizie wszystkich elementów listy zliczającej. *i*-tą liczbę na liście wypisujemy tyle razy jaką wartość ma lista pod indeksem *i*. W ten sposób uzyskujemy posortowaną listę. Jeśli listę zliczającą będziemy przeglądać od lewej do prawej to uzyskamy listę posortowaną rosnącą.

5.3 Złożoność obliczeniowa

Złożoność czasowa działania algorytmu przez zliczanie (który nie wykorzystuje porównań!) wynosi O(n), a więc mniej niż wynosi dolna granica sortowań za pomocą porównań. Algorytm ten jest stabilny, gdyż liczby o tych samych wartościach występują w tablicy wynikowej w takiej samej kolejności, jak w tablicy początkowej. Stabilność algorytmu sortowania jest tylko wtedy istotna, gdy z sortowanymi danymi są związane dodatkowe dane, które np. mogą być już posortowane wg innego klucza, zaś brak stabilności algorytmu spowodowałby zniszczenie uzyskanego wcześniej porządku.

5.4 Pomiary czasu w jednostce ms

zliczanie			
N	posortowane	losowe	posortowane odwrtonie
1000	670	742	1229
10000	610	662	1140
50000	908	1045	1499
100000	1326	1414	1951
500000	4835	5577	5278
5000000	43949	44158	45214

5.5 Złożoność czasowa przedstawiona na wykresie