



**POLITECHNIKA
RZESZOWSKA**
im. IGNACEGO ŁUKASIEWICZA

POLITECHNIKA RZESZOWSKA

Wydział Matematyki i Fizyki Stosowanej
Inżynieria i analiza danych

ALGORYTMY i STRUKTURY DANYCH
PROJEKT 2

Paulina Kmiec

Grupa nr. 03

Nr. indeksu 166657

Spis treści

1. Treść zadania	2
2. Kod programu.....	3
3. Działanie programu	5
4. Sortowanie przez wstawianie.....	8
4.1 Schemat blokowy	8
4.2 Pseudokod	8
4.3 Złożoność obliczeniowa:.....	9
4.4. Pomiary czasu w jednostce ms.....	9
4.5 Złożoność czasowa przedstawiona na wykresie.....	9
5. Sortowanie przez zliczanie	10
5.1 Schemat blokowy	10
5.2 Pseudokod	11
5.3 Złożoność obliczeniowa	11
5.4 Pomiary czasu w jednostce ms.....	11
5.5 Złożoność czasowa przedstawiona na wykresie.....	12

1. Treść zadania

Zaimplementuj sortowanie przez wstawianie oraz sortowanie przez zliczanie.

2. Kod programu

```
1  #include <iostream>
2  #include <fstream>
3  #include <time.h>
4  #include <chrono>
5  using namespace std;
6
7  int N = 2000;           //liczby będą generowane z zakresu 0 - N-1
8  int ROZMIAR = 100000;  //rozmiar sortowanej tablicy
9
10 int tabN[6] = { 1000 , 10000 , 50000 , 100000 , 500000 , 5000000 };
11
12 void wygenerujLiczbyDoPliku()
13 {
14     int liczba;
15     ofstream plik("liczbyLosowe.txt");
16
17     //w pętli generuję liczby losowe
18     for (int i = 0; i < ROZMIAR; i++)
19     {
20         liczba = rand() % N;
21         //zapisuję do pliku liczby oddzielone spacją
22         plik << liczba << " ";
23     }
24
25     //zamykam plik
26     plik.close();
27 }
28
29 void zapiszTabliceDoPliku(int* tablica)
30 {
31     ofstream plik("posortowane.txt");
32
33     for (int i = 0; i < ROZMIAR; i++)
34     {
35         plik << tablica[i] << " ";
36     }
37
38     plik.close();
39 }
40
41 void wczytajTablice(int* tablica)
42 {
43     ifstream plik("liczbyLosowe.txt");
44
45     //w pętli czytuję kolejne elementy tablicy
46     for (int i = 0; i < ROZMIAR; i++)
47     {
48         plik >> tablica[i];
49     }
```

```
49 }
50
51 //zamykam plik
52 plik.close();
53 }
54
55 void sortowaniePrzezWstawianie(int* tablica)
56 {
57     for (int i = 1; i < ROZMIAR; i++)
58     {
59         int klucz = tablica[i];
60         int j = i - 1;
61
62         while (j >= 0 && tablica[j] > klucz)
63         {
64             tablica[j + 1] = tablica[j];
65             j--;
66         }
67         tablica[j + 1] = klucz;
68     }
69 }
70
71 void sortowaniePrzezZliczanie(int* tablica)
72 {
73     //alokuje pamięć do tablicy pomocniczej
74     int* posortowana = new int[ROZMIAR];
75
76     //alokuje miejsce na tablicę przechowującą zliczenia poszczególnych liczb
77     int* liczniki = new int[N+1];
78     for (int i = 0; i < N; i++)
79         liczniki[i] = 0;
80
81     //zliczam występowanie liczb w tablicy 'liczniki'
82     for (int i = 0; i < ROZMIAR; i++)
83         liczniki[tablica[i]]++;
84
85     //obliczamy pozycje końcowe elementów
86     for (int i = 1; i <= N; i++)
87         liczniki[i] += liczniki[i - 1];
88
89     //przepisuję elementy do tablicy wynikowej
90     for (int i = 0; i < ROZMIAR; i++)
91     {
92         int index = liczniki[tablica[i]] - 1;
93         posortowana[index] = tablica[i];
94         liczniki[tablica[i]]--;
95     }
96
97     for (int i = 0; i < ROZMIAR; i++)
```

```

97     for (int i = 0; i < ROZMIAR; i++)
98         tablica[i] = posortowana[i];
99
100     //zwalniam dynamicznie zaalokowan pamie
101     delete[] posortowana;
102     delete[] liczniki;
103 }
104
105 //funkcja pomocnicza do wypisywania zawartosci tablicy
106 void wypiszTablice(int* tablica)
107 {
108     for (int i = 0; i < ROZMIAR; i++)
109     {
110         cout << tablica[i] << endl;
111     }
112 }
113
114 void testDaneLosowe(int *tablica)
115 {
116     cout << "DANE LOSOWE\n";
117
118     for (int i = 0; i < 6; i++)
119     {
120         N = tabN[i];
121
122         //zliczam czas sortowanie przez zliczanie
123         wygenerujLiczbyDoPliku();
124         wczytajTablice(tablica);
125         auto start = chrono::high_resolution_clock::now();
126         sortowaniePrzezZliczanie(tablica);
127         auto stop = chrono::high_resolution_clock::now();
128         auto czasSortowaniePrzezZliczanie = chrono::duration_cast<std::chrono::microseconds>(stop - start).count();
129
130         //zliczam czas sortowania przez wstawianie
131         wygenerujLiczbyDoPliku();
132         wczytajTablice(tablica);
133         start = chrono::high_resolution_clock::now();
134         sortowaniePrzezWstawianie(tablica);
135         stop = chrono::high_resolution_clock::now();
136         auto czasSortowaniePrzezWstawianie = chrono::duration_cast<std::chrono::microseconds>(stop - start).count();
137
138         cout << "N = " << N << "S. przez zliczanie " << czasSortowaniePrzezZliczanie << "ms S. przez wstawianie " << czasSortowaniePrzezWstawianie << "ms\n";
139     }
140 }
141
142 void testDanePosortowane(int* tablica)
143 {
144     cout << "\n\nDANE POSORTOWANE\n";
145 }

```

```

146     for (int i = 0; i < 6; i++)
147     {
148         N = tabN[i];
149
150         wygenerujLiczbyDoPliku();
151         wczytajTablice(tablica);
152         sortowaniePrzezWstawianie(tablica);
153
154         //zliczam czas sortowanie przez zliczanie
155         auto start = chrono::high_resolution_clock::now();
156         sortowaniePrzezZliczanie(tablica);
157         auto stop = chrono::high_resolution_clock::now();
158         auto czasSortowaniePrzezZliczanie = chrono::duration_cast<std::chrono::microseconds>(stop - start).count();
159
160         //zliczam czas sortowania przez wstawianie
161         start = chrono::high_resolution_clock::now();
162         sortowaniePrzezWstawianie(tablica);
163         stop = chrono::high_resolution_clock::now();
164         auto czasSortowaniePrzezWstawianie = chrono::duration_cast<std::chrono::microseconds>(stop - start).count();
165
166         cout << "N = " << N << "S. przez zliczanie " << czasSortowaniePrzezZliczanie << "ms S. przez wstawianie " << czasSortowaniePrzezWstawianie << "ms\n";
167     }
168 }
169
170 void sortujOdwrotnie(int* tablica)
171 {
172     for (int i = 0; i < ROZMIAR; i++)
173     for (int j = 0; j < ROZMIAR; j++)
174     {
175         if (tablica[i] > tablica[j])
176             swap(tablica[i], tablica[j]);
177     }
178 }
179
180 void testDanePosortowaneOdwrotnie(int* tablica)
181 {
182     cout << "\n\nDANE POSORTOWANE ODWROTNIENIE\n";
183
184     for (int i = 0; i < 6; i++)
185     {
186         N = tabN[i];
187
188         wygenerujLiczbyDoPliku();
189         wczytajTablice(tablica);
190
191         sortujOdwrotnie(tablica);
192         //zliczam czas sortowanie przez zliczanie
193         auto start = chrono::high_resolution_clock::now();
194         sortowaniePrzezZliczanie(tablica);

```

```

195     auto stop = chrono::high_resolution_clock::now();
196     auto czasSortowaniePrzezZliczanie = chrono::duration_cast<std::chrono::microseconds>(stop - start).count();
197
198     sortujOdwrotnie(tablica);
199     //zliczam czas sortowania przez wstawianie
200     start = chrono::high_resolution_clock::now();
201     sortowaniePrzezWstawianie(tablica);
202     stop = chrono::high_resolution_clock::now();
203     auto czasSortowaniePrzezWstawianie = chrono::duration_cast<std::chrono::microseconds>(stop - start).count();
204
205     cout << "N = " << N << "S. przez zliczanie " << czasSortowaniePrzezZliczanie << "ms S. przez wstawianie " << czasSortowaniePrzezWstawianie << "ms\n";
206 }
207
208 int main()
209 {
210
211     //alokuje dynamicznie pamięć na tablicę
212     int* tablica = new int[ROZMIAR];
213
214     wczytajTablice(tablica);
215
216     //wypiszTablice(tablica);
217
218     //zapiszTabliceDoPliku(tablica);
219
220     //testDaneLosowe(tablica);
221     //testDanePosortowane(tablica);
222     testDanePosortowaneOdwrotnie(tablica);
223     return 0;
224 }
225

```

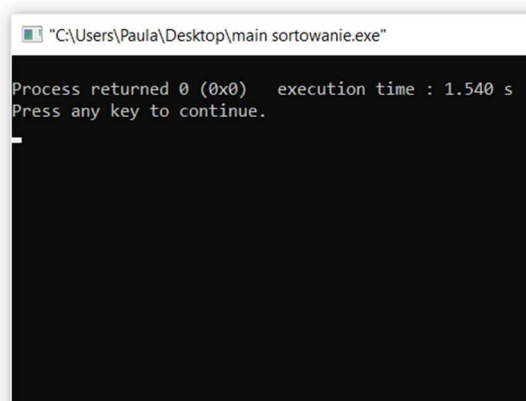
3. Działanie programu

Wczytuje tablicę

```

208 int main()
209 {
210
211     //alokuje dynamicznie pamięć na tablicę
212     int* tablica = new int[ROZMIAR];
213
214     wczytajTablice(tablica);
215
216     //wypiszTablice(tablica);
217
218     //zapiszTabliceDoPliku(tablica);
219
220     //testDaneLosowe(tablica);
221     //testDanePosortowane(tablica);
222     //testDanePosortowaneOdwrotnie(tablica);
223     return 0;
224 }
225

```



Wypisuje wczytaną tablicę

```
"C:\Users\Paula\Desktop\main sortowanie.exe"
22190
15521
10175
30513
24419
18536
30973
29442
13720
8219
15663
13771
27490
32651
10932
7935
6844
12248
24827
8162
16682
1158
16709
16703
32413
11987
17581
12633
15255
23368
29542
28197
31375
19855
24901
24502
22679
28292
21375
30457
316
142
29619
9378
6574
28208

Process returned 0 (0x0)   execution time : 27.968 s
Press any key to continue.
```

Zapisuje tablicę do pliku

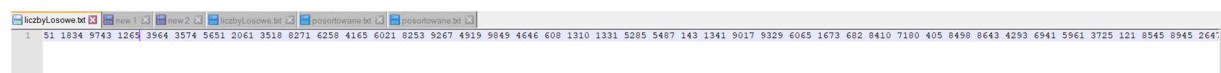
```
// STWORZENIE WYKONAWCZYCH KODÓW NA TABLICIE
int* tablica = new int[ROZMIAR];

wczytajTablice(tablica);

//wypiszTablice(tablica);

zapiszTabliceDoPliku(tablica);
```

```
"C:\Users\Paula\Desktop\main sortowanie.exe"
Process returned 0 (0x0)   execution time : 2.145 s
Press any key to continue.
```



Dane posortowane:

```
int main()
{
    //alokuje dynamicznie pamięć na tablicę
    int* tablica = new int[ROZMIAR];

    wczytajTablice(tablica);

    //wypiszTablice(tablica);

    //zapiszTabliceDoPliku(tablica);

    //testDaneLosowe(tablica);
    testDanePosortowane(tablica);
    //testDanePosortowaneOdwrrotnie(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 odwrrotnie:

```
int main()
{
    //alokuje dynamicznie pamięć na tablicę
    int* tablica = new int[ROZMIAR];

    wczytajTablice(tablica);

    //wypiszTablice(tablica);

    //zapiszTabliceDoPliku(tablica);

    //testDaneLosowe(tablica);
    //testDanePosortowane(tablica);
    testDanePosortowaneOdwrrotnie(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

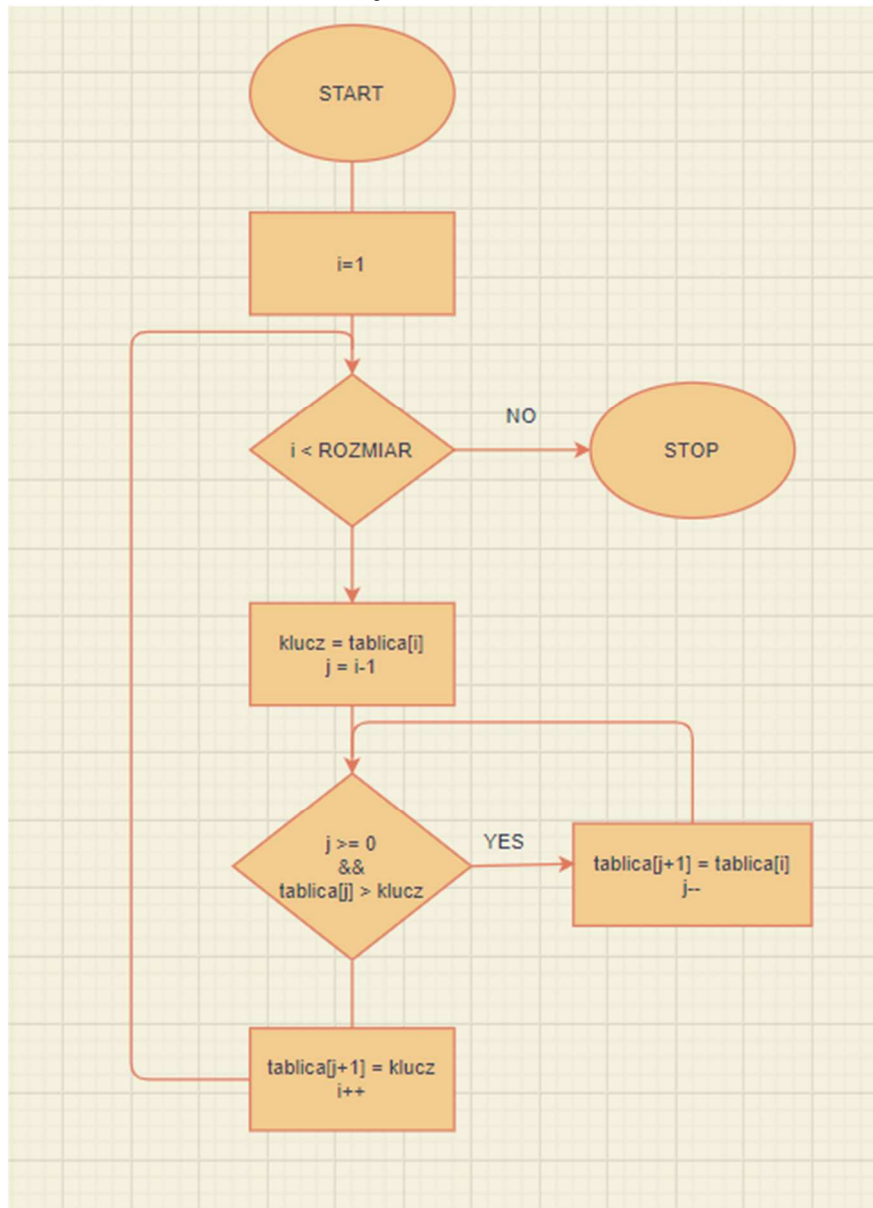
Process returned 0 (0x0) execution time : 374.441 s
Press any key to continue.

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	odwrrotnie		
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

```
1  for i=1 to ROZMIAR:
2      Wstaw tablica[i] w posortowany ciąg tablica[1 ... i-1]
3      klucz =tablica[i]
4      j = i-1
5      while j>=0 and tablica[j] > klucz :
6          tablica[j+1] = tablica[j]
7          j = j-1
8          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).

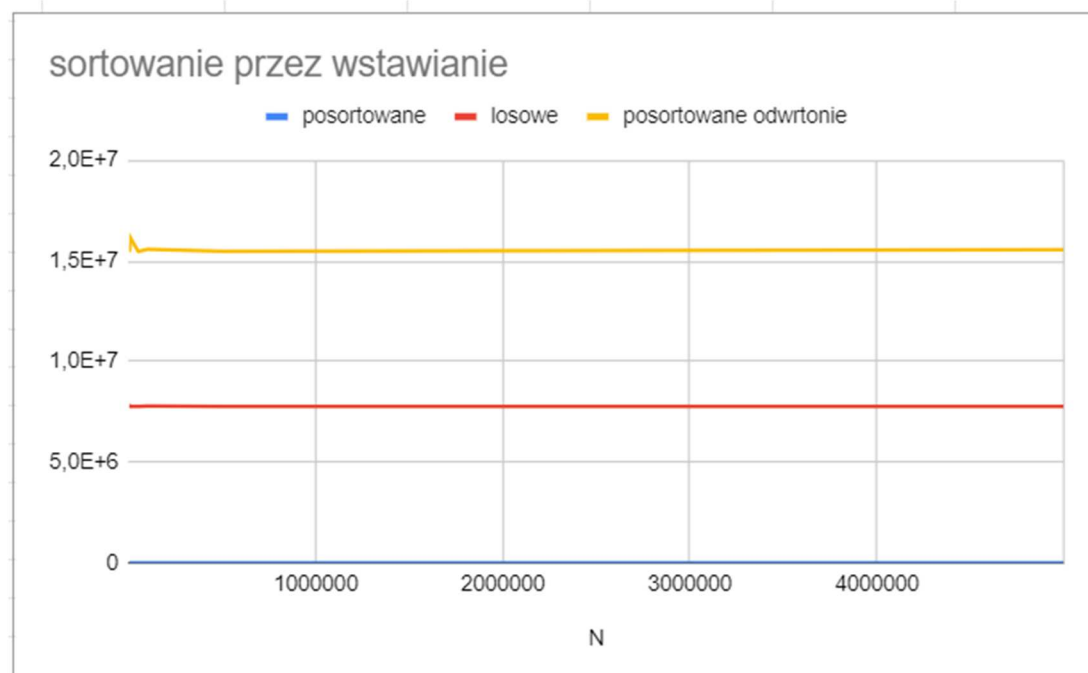
4.3 Złożoność obliczeniowa:

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

4.4. Pomiary czasu w jednostce ms

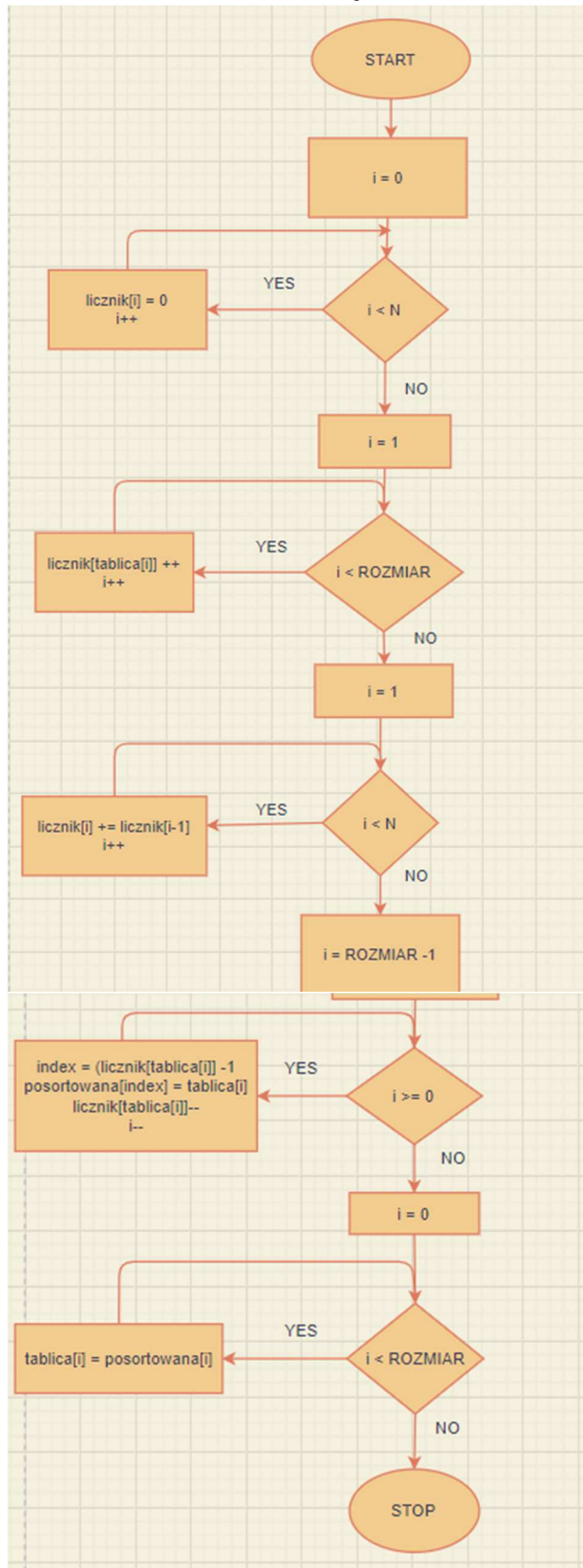
wstawianie				
N	posortowane	losowe	posortowane odwrócone	
1000	444	7800279	15475229	
10000	475	7754039	16131722	
50000	444	7757442	15490694	
100000	441	7778710	15604505	
500000	468	7756415	15509255	
5000000	430	7754493	15580538	

4.5 Złożoność czasowa przedstawiona na wykresie



5. Sortowanie przez zliczanie

5.1 Schemat blokowy



5.2 Pseudokod

```
1 posortowana = new int[ROZMIAR]
2 for i=0 to N
3     liczniki[i] = 0
4 for i=0 to ROZMIAR
5     liczniki[tabela[i]]++
6 for i=0 to N
7     liczniki[i] += liczniki[i-1]
8 for i=0 to tabela[i]
9     index = liczniki[tabela[i]]-1
10    posortowana[index] = tabela[i]
11    liczniki[tabela[i]]--
12 for i=0 to ROZMIAR
13     tabela[i] = posortowana[i]
14 delete[] posortowana
15 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 odwrócone
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

