



**WYDZIAŁ  
MATEMATYKI  
I FIZYKI STOSOWANEJ**  
POLITECHNIKI RZESZOWSKIEJ

Projekt 2  
Algorytmy i Struktury  
danych

Weronika Zagaja  
Inżynieria i analiza danych  
Nr indeksu 166718

08.01.2021r

## Spis treści

1. Wstęp.....	3
2. Treść zadania do wykonania .....	3
3. Teoretyczne podstawy sortowań .....	3
Sortowanie przez wstawianie .....	3
Sortowanie kubełkowe .....	4
4. Schematy blokowe algorytmów .....	4
5. Pseudokody .....	7
6. Kod programu.....	8
7. Działanie programu .....	10
8. Złożoność obliczeniowa .....	13
Sortowanie przez wstawianie .....	13
Sortowanie kubełkowe .....	13
Wykresy porównujące.....	13
Oczekiwana złożoność czasowa.....	13
Optymistyczna złożoność czasowa .....	14
Pesymistyczna złożoność czasowa.....	15
9. Wnioski .....	16
Rysunek 1: Schemat blokowy "sortowanie przez wstawianie" .....	5
Rysunek 2 Schemat blokowy "sortowanie kubełkowe" .....	6
Rysunek 3 Pseudokod dla sortowania przez wstawianie.....	7
Rysunek 4 Pseudokod dla sortowania kubełkowego.....	7
Rysunek 5 Sortowanie dla ciągu liczb optymistycznych .....	11
Rysunek 6 Sortowanie dla ciągu liczb pesymistycznych .....	11
Rysunek 7 Sortowanie liczb losowych dla $N=300$ .....	12
Rysunek 8 Zapis wyniku sortowań dla ciągu pesymistycznego w pliku tekstowym .....	12
Rysunek 9 Tabela czasów uzyskanych w sortowaniach .....	13
Rysunek 10 Wykres oczekiwanej złożoności czasowej dla obu algorytmów .....	14
Rysunek 11 Tabela czasów uzyskanych w sortowaniach .....	14
Rysunek 12 Wykres optymistycznej złożoności czasowej obu algorytmów .....	14
Rysunek 13 Tabela czasów uzyskanych w sortowaniach .....	15
Rysunek 14 Wykres pesymistycznej złożoności czasowej obu algorytmów .....	15

## 1. Wstęp

Celem projektu było dokonanie implementacji kodu, który polegał na napisaniu programu, którego zadaniem była realizacja czynności, opisanych wcześniej w instrukcji. Dodatkowo projekt zawiera schematy blokowe oraz pseudokody. Program został napisany w środowisku Code::Blocks IDE w języku C++. Środowisko to posiada otwarty kod źródłowy i kluczową zaletą tego oprogramowania jest jego wieloplatformowość.

## 2. Treść zadania do wykonania

Problematyka projektu polegała na implementacji sortowania liczb przez wstawianie oraz sortowania kubełkowego. Głównymi cechami programu są:

- a) program powinien mieć możliwość odczytywania danych wejściowych z pliku tekstowego i zapisu posortowanego ciągu wynikowego również do pliku tekstowego
- b) na potrzeby wykonywanych testów należy zaimplementować funkcję generującą "losowe" ciągi elementów (o zadanej długości) i zapisującą je do pliku tekstowego
- c) założyć, że sortowanymi elementami są liczby całkowite z przedziału  $[0, N]$ , gdzie  $N$  powinno być "odpowiednio dużym" parametrem ustalonym wewnątrz programu
- d) kod powinien być opatrzony stosownymi komentarzami.

Na początku realizowania projektu, powinna zostać wykonana definicja problemu np.: w postaci modelu matematycznego, następnie koncepcja rozwiązania i zapis algorytmu w postaci schematu blokowego i pseudokodu w celu ułatwienia implementacji kodu. Na końcu należało przejść do urzeczywistnienia kodu w wybranym środowisku programistycznym.

## 3. Teoretyczne podstawy sortowań

### Sortowanie przez wstawianie

Algorytm **sortowania przez wstawianie** można porównać do sposobu układania kart pobieranych z talii. Najpierw bierzemy pierwszą kartę. Następnie pobieramy kolejne, aż do wyczerpania talii. Każdą pobraną kartę porównujemy z kartami, które już trzymamy w ręce i szukamy dla niej miejsca przed pierwszą kartą starszą znajdziemy takie miejsce, rozsuwamy karty i nową wstawiamy na przygotowane w ten sposób miejsce sortowanie przez wstawianie, ang. **Insertion Sort/Insert Sort**. Jeśli nasza karta jest najstarsza, to umieszczamy ją na samym końcu. Tak porządkujemy karty.

Algorytm sortowania przez wstawianie będzie składał się z dwóch pętli.

- a) Pętla główna (zewnętrzna) symuluje pobieranie kart, czyli w tym wypadku elementów zbioru. Odpowiednikiem kart na ręce jest tzw. lista uporządkowana, którą sukcesywnie będziemy tworzyli na końcu zbioru.
- b) Pętla sortująca (wewnętrzna) szuka dla pobranego elementu miejsca na liście uporządkowanej. Jeśli takie miejsce zostanie znalezione, to elementy listy są odpowiednio rozsuwane, aby tworzyć miejsce na nowy element i element wybrany przez pętlę główną trafia tam.

W ten sposób lista uporządkowana rozrasta się. Jeśli na liście uporządkowanej nie ma elementu większego od wybranego, to element ten trafia na koniec listy.

Sortowanie zakończymy, gdy pętla główna wybierze wszystkie elementy zbioru.

Najważniejszą operacją w opisywanym algorytmie sortowania jest wstawianie wybranego elementu na listę uporządkowaną. Zasady są następujące:

1. Na początku sortowania lista uporządkowana zawiera tylko jeden, ostatni element zbioru. Jednoelementowa lista jest zawsze uporządkowana.
2. Ze zbioru zawsze wybieramy element leżący tuż przed listą uporządkowaną. Element ten zapamiętujemy w zewnętrznej zmiennej. Miejsce, które zajmował, możemy potraktować jak puste.
3. Wybrany element porównujemy z kolejnymi elementami listy uporządkowanej.
4. Jeśli natrafimy na koniec listy, element wybrany wstawiamy na puste miejsce - lista rozrasta się o nowy element.
5. Jeśli element listy jest większy od wybranego, to element wybrany wstawiamy na puste miejsce - lista rozrasta się o nowy element.
6. Jeśli element listy nie jest większy od wybranego, to element listy przesuwamy na puste miejsce. Dzięki tej operacji puste miejsce wędruje na liście przed kolejny element. Kontynuujemy porównywanie, aż wystąpi sytuacja z punktu 4 lub 5.

## Sortowanie kubełkowe

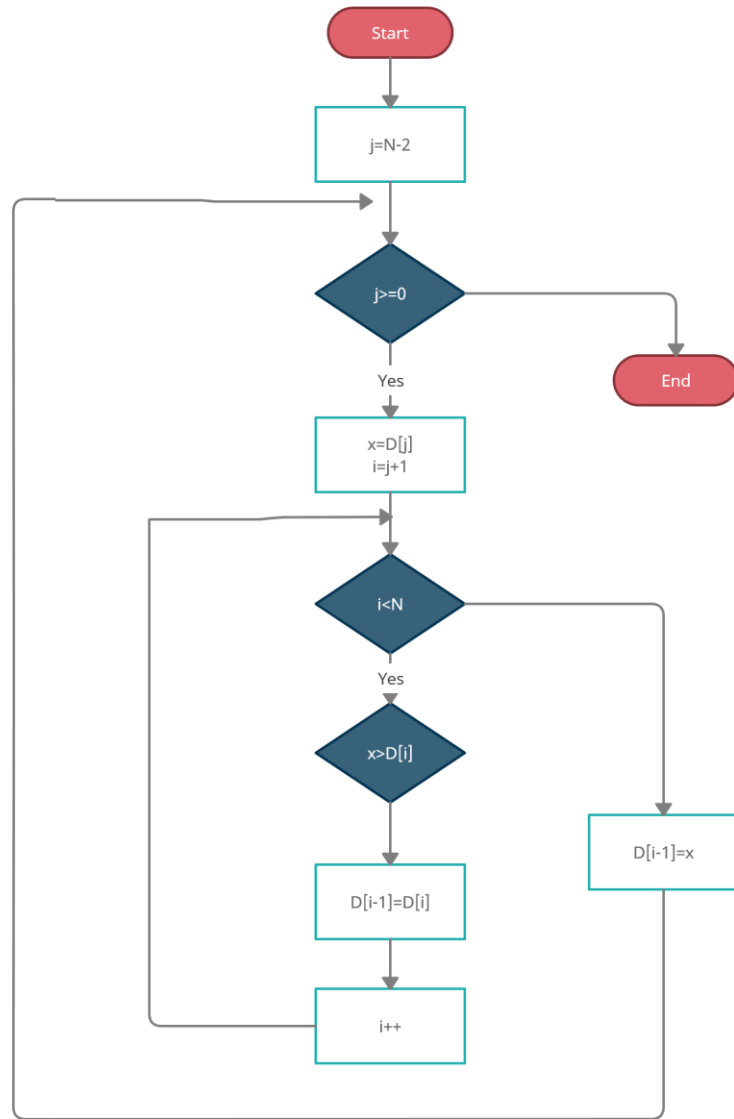
Algorytm sortowania kubełkowego pozwala sortować zbiory liczb całkowitych - najlepiej o dużej ilości elementów, lecz o małym zakresie wartości. Zasada działania jest następująca:

1. Określamy zakres wartości, jakie mogą przyjmować elementy sortowanego zbioru. Niech  $w_{\min}$  oznacza najmniejszą wartość, a  $w_{\max}$  niech oznacza wartość największą.
2. Dla każdej możliwej wartości przygotowujemy kubełek-licznik, który będzie zliczał ilość wystąpień tej wartości w sortowanym zbiorze. Liczba liczników jest równa  $(w_{\max} - w_{\min} + 1)$ . Każdy licznik jest początkowo ustawiony na wartość zero.
3. Przeglądamy kolejno elementy zbioru od pierwszego do ostatniego. Dla każdego elementu zbioru zwiększamy o jeden zawartość licznika o numerze równym wartości elementu. Na przykład, jeśli kolejny element zbioru ma wartość 3, to zwiększamy licznik o numerze 3. W efekcie po przeglądnięciu wszystkich elementów zbioru liczniki będą zawierały ilość wystąpień każdej z możliwych wartości. Jeśli dany licznik zawiera 0, to wartość równa numerowi licznika w zbiorze nie występuje. Inaczej wartość ta występuje tyle razy, ile wynosi zawartość jej licznika.
4. Przeglądamy kolejne liczniki zapisując do zbioru wynikowego ich numery tyle razy, ile wynosi ich zawartość. Zbiór wyjściowy będzie posortowany.

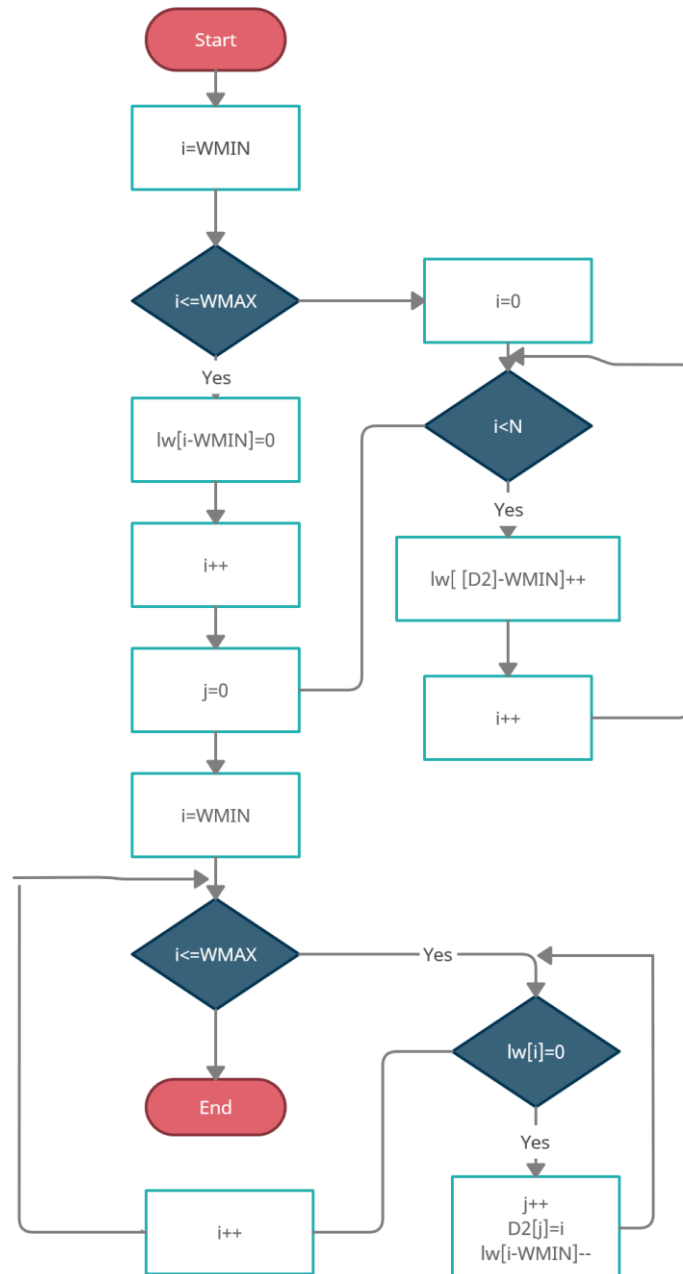
## 4. Schematy blokowe algorytmów

Schemat blokowy jest podglądową formą graficznego przedstawienia algorytmu. Tworzy się go korzystając ze ściśle określonego zbioru figur geometrycznych oraz stosując ustalone reguły ich łączenia. Każdy schemat blokowy musi być spójny tzn., że do bloku typu *początek* do bloku typu *koniec* musi prowadzić przynajmniej jedna droga.

Poniżej zostały przedstawione schematy blokowe dla obu algorytmów:



Rysunek 1: Schemat blokowy "sortowanie przez wstawianie"



Rysunek 2 Schemat blokowy "sortowanie kubelkowe"

Odzwierciedlone graficznie schematy blokowe, przedstawionego powyżej problemu zawierają wewnątrz bloków w pozorowany sposób zapis występujących w algorytmie operacji, takich jak:

- arytmetyczne,
- logiczne,
- operacje wejścia i wyjścia,
- warunki, od których zależą decyzje co do kolejności wykonywania działań.

## 5. Pseudokody

Pseudokodem nazywamy taki sposób zapisu algorytmu, który zachowując strukturę charakterystyczną dla kodu zapisanego w języku programowania, rezygnuje ze ścisłych reguł składniowych na rzecz prostoty i czytelności. W pseudokodzie nie istnieją standardy zapisu, opiera się jedynie na składni istniejących języków programowania.

Poniższe pseudokody miały na celu przedstawić skrót myślowy i pomóc w realizacji danego programu. Zostały w nim opisane krok po kroku potrzebne funkcje, aby prawidłowo wykonać implementację sortowań.

```
1. Dopóki j=N-2 do j>=0
    zmniejsz j o 1
2. Jeżeli ( j>=0 ) wykonaj
    x=D[j]
    i=j+1
    w przeciwnym wypadku
        zakończ
3. Jeżeli (i<N) i (x>D[j]) wykonaj:
    D[i-1]=D[i]
    i=i+1
    w przeciwnym wypadku
        D[i-1]=x
```

*Rysunek 3 Pseudokod dla sortowania przez wstawianie*

```
1. Dopóki i=WMIN do i <= WMAX
    zwiększ i o 1
2. Jeżeli ( i<= WMAX) wykonuj:
    lw[i-WMIN] = 0
    Dopóki i=WMIN do i <= WMAX
        zwiększ i o 1
    w przeciwnym wypadku
        dla i=0
        jeżeli i<N wykonaj:
            lw[[D2]- WMIN] zwiększ o 1
            i zwiększ o 1

2. Dla j=0
3. Jeżeli (i <=WMAX) wykonuj:
    zakończ
    w przeciwnym wypadku
        lw[i]=0
4. Jeżeli (lw[i]=0) wykonuj:
    zwiększ j o 1
    D2[j]=i
    zmniejsz lw[i-WMIN] o 1
    w przeciwnym wypadku
        zwiększ i o 1
```

*Rysunek 4 Pseudokod dla sortowania kubelkowego*

## 6. Kod programu

```
1 // biblioteki
2 #include <iostream>
3 #include <fstream>
4 #include <time.h>
5 #include <cstdlib>
6 #include <iomanip>
7 #include <windows.h>
8 using namespace std;
9
10 //deklaracja czasu
11 clock_t start, stop;
12 double czas;
13
14 // deklaracja zmiennych
15 const int N=100;
16 const int WMIN = 0;
17 const int WMAX = N;
18 int d[N];
19 int d2[N];
20
21 void losowe() //implementacja funkcji do losowania liczb
22 {
23     fstream output("wylosowane_liczby.txt", ios::out);
24
25     // Najpierw wypełniamy tablice d[] liczbami pseudolosowymi
26     srand(time(NULL));
27     cout<<"Przed sortowaniem: "<<endl;
28     output<<"Przed sortowaniem: "<<endl;
29     for(int i=0; i<N; i++)
30     {
31         d[i] = WMIN + rand() % (WMAX - WMIN + 1);
32         cout << d[i]<<" ";
33         output << d[i]<<" ";
34     }
35     cout << endl;
36
37     //wypisanie tablicy d[]
38     for(int i=0; i<N; i++)
39     {
40         d2[i] = WMIN + rand() % (WMAX - WMIN + 1);
41     }
42     for(int i=0; i<N; i++)
43     {
44         d2[i]=d[i];
45     }
46
47     output.close();
48 }
49
```



```

49
50 void Insertion_Sort( int *D, int N) //implementacja funkcji do sortowania przez wstawianie
51 {
52     //sortujemy przez wstawianie
53     int j, i, x;
54     {
55         for(j = N - 2; j >= 0; j--)
56         {
57             x = D[j];
58             i = j + 1;
59             while((i < N) && (x > D[i]))
60             {
61                 D[i - 1] = D[i];
62                 i++;
63             }
64             D[i - 1] = x;
65         }
66     }
67 }
68
69
70 void bucket_sort(int *D2, int WMIN, int WMAX) //implementacja funkcji do sortowania kubelkowego
71 {
72     // stworzenie wskaźnika
73     int *lw;
74     lw = new int [WMAX - WMIN + 1];
75     int i, j;
76     {
77         // sortujemy kubelkowo
78
79         for(i = WMIN; i <= WMAX; i++) lw[i - WMIN] = 0;
80         for(i = 0; i < N; i++) lw[D2[i] - WMIN]++;
81         j = 0;
82         for(i = WMIN; i <= WMAX; i++) while(lw[i - WMIN]--) D2[j++] = i;
83     }
84     delete [] lw;
85     cout<<endl;
86 }
87
88 void Sort(string inputFile) //funkcja umocliwiająca sortowanie dla ciągów
89                             //optymistycznych pesymistycznych i oczekiwanych
90 {
91     fstream output("wyniki_sortowania.txt", ios::out);
92     if(inputFile != "expected")
93     {
94         int x;
95         fstream input(inputFile, ios::in);
96
97         cout << "Przed sortowaniem:" << endl;
98         output << "Przed sortowaniem:" << endl;
99         int i=0;
100         while (input >> x) // zaczytuję posortowane liczby z pliku "optymistyczne.txt"
101         {
102             d[i] = x;
103             cout << d[i]<<" ";
104             output << d[i]<<" ";
105             i++;
106         }
107         cout << endl;
108         output << endl;
109
110         input.close();
111     }
112     else
113     {
114         losowe(); //wywołuję funkcję "losowe"
115     }
116 }

```

```

117 //liczymy czas sortowania oraz wczytanie
118 start = clock();
119 Insertion_Sort(d, N);
120
121 output<<endl;
122 stop = clock();
123 czas = (double)(stop-start)/(CLOCKS_PER_SEC/1000); // zliczamy czas w milisekundach
124
125 //wpisywanie posortowanej tablicy
126 cout << "Po sortowaniu Insertion Sort : " << endl;
127 for(int i = 0; i < N; i++){ cout << d[i]<<" ";}
128 output << "Po sortowaniu Insertion Sort : " << endl;
129 for(int i = 0; i < N; i++) output << setw(4) << d[i];
130 cout<<endl;
131
132 cout << "Czas sortowania Insertion Sort : " << czas << " msec" << endl;
133 cout<<endl;
134
135 //liczymy czas sortowania kubełkowego
136 start=clock();
137 bucket_sort(d2, WHIN, WHMAX);
138 stop = clock();
139 czas = (double)(stop-start)/(CLOCKS_PER_SEC/1000);
140
141 cout << "Po sortowaniu bucket_sort: " <<endl;
142 for(int i = 0; i < N; i++) {cout << d[i]<<" ";}
143 output<< "Po sortowaniu bucket_sort: ";
144 for(int i = 0; i < N; i++) output << setw(4) << d[i];
145
146 cout<<endl;
147 cout << "Czas sortowania bucket_sort: " << czas << " msec" << endl << endl;
148 output.close();
149 }
150
151 int main()
152 {
153     int wybor = 0;
154     cout << "1 - optimistic" << endl;
155     cout << "2 - pessimistic" << endl;
156     cout << "3 - expected" << endl;
157     cin >> wybor;
158
159     switch(wybor) //pozwała wybrać plik do sortowania
160     {
161     case 1:
162         cout << "sorting optimistic case..." << endl;
163         Sort("optymistyczne.txt");
164         break;
165     case 2:
166         cout << "sorting pessimistic case..." << endl;
167         Sort("pesymistyczne.txt");
168         break;
169     case 3:
170         cout << "sorting random set of numbers..." << endl;
171         Sort("expected");
172         break;
173     }
174     return 0;
175 }
176

```

## 7. Działanie programu

Program miał dokonać sortowania liczb dwoma metodami i zliczenie czasu sortowań w celu porównania szybkości obu algorytmów. W związku z tym, kod został napisany tak, aby użytkownik miał możliwość wybrania danego ciągu liczb - optymistycznego, oczekiwanego oraz pesymistycznego. Zestaw danych - optymistyczny zawiera liczby w dużej mierze posortowane, (zob. Rysunek 5) pesymistyczny zaś obejmuje liczby w najbardziej niekorzystnym układzie, czyli w takim gdzie algorytmy muszą sortować najwięcej liczb (zob. Rysunek 6). Oba te ciągi odczytywane są z plików tekstowych. Ostatni zestaw danych- oczekiwany, posiada liczby wylosowane z funkcji

```
"C:\Users\weron\Desktop\Projekt 2-Weronika Zagaja\bin\Debug\Projekt 2-Weronika Zagaja.exe"
1 - optimistic
2 - pessimistic
3 - expected
1
sorting optimistic case...
Przed sortowaniem:
0 2 10 12 12 13 13 14 16 18 19 20 21 21 22 23 24 25 26 27 28 28 30 30 30 32 32 33 34 36 37 39 41 42 44 45 46 46 46 47 49 49 4
9 50 51 52 52 53 54 54 54 55 56 58 58 58 60 60 61 66 66 67 67 68 69 70 70 72 72 72 72 72 73 73 74 76 78 79 79 83 85 85 87 88
88 89 89 90 92 93 95 95 95 95 97 97 97 98 99 100

Po sortowaniu Insertion Sort :
0 2 10 12 12 13 13 14 16 18 19 20 21 21 22 23 24 25 26 27 28 28 30 30 30 32 32 33 34 36 37 39 41 42 44 45 46 46 46 47 49 49 4
9 50 51 52 52 53 54 54 54 55 56 58 58 58 60 60 61 66 66 67 67 68 69 70 70 72 72 72 72 72 73 73 74 76 78 79 79 83 85 85 87 88
88 89 89 90 92 93 95 95 95 95 97 97 97 98 99 100
Czas sortowania Insertion Sort : 0 msec

Po sortowaniu bucket_sort:
0 2 10 12 12 13 13 14 16 18 19 20 21 21 22 23 24 25 26 27 28 28 30 30 30 32 32 33 34 36 37 39 41 42 44 45 46 46 46 47 49 49 4
9 50 51 52 52 53 54 54 54 55 56 58 58 58 60 60 61 66 66 67 67 68 69 70 70 72 72 72 72 72 73 73 74 76 78 79 79 83 85 85 87 88
88 89 89 90 92 93 95 95 95 95 97 97 97 98 99 100
Czas sortowania bucket_sort: 0 msec

Process returned 0 (0x0)   execution time : 4.186 s
Press any key to continue.
_
```

*Rysunek 5 Sortowanie dla ciągu liczb optymistycznych*

```
"C:\Users\weron\Desktop\Projekt 2-Weronika Zagaja\bin\Debug\Projekt 2-Weronika Zagaja.exe"
1 - optimistic
2 - pessimistic
3 - expected
2
sorting pessimistic case...
Przed sortowaniem:
100 99 98 97 97 97 95 95 95 95 93 92 90 89 89 88 88 87 85 85 83 79 79 78 76 74 73 73 72 72 72 72 72 70 70 69 68 67 67 66 66 6
1 60 60 58 58 58 56 55 54 54 54 53 52 52 51 50 49 49 49 47 46 46 46 45 44 42 41 39 37 36 34 33 32 32 30 30 30 28 28 27 26 25
24 23 22 21 21 20 19 18 16 14 13 13 12 12 10 2 0

Po sortowaniu Insertion Sort :
0 2 10 12 12 13 13 14 16 18 19 20 21 21 22 23 24 25 26 27 28 28 30 30 30 32 32 33 34 36 37 39 41 42 44 45 46 46 46 47 49 49 4
9 50 51 52 52 53 54 54 54 55 56 58 58 58 60 60 61 66 66 67 67 68 69 70 70 72 72 72 72 72 73 73 74 76 78 79 79 83 85 85 87 88
88 89 89 90 92 93 95 95 95 95 97 97 97 98 99 100
Czas sortowania Insertion Sort : 0 msec

Po sortowaniu bucket_sort:
0 2 10 12 12 13 13 14 16 18 19 20 21 21 22 23 24 25 26 27 28 28 30 30 30 32 32 33 34 36 37 39 41 42 44 45 46 46 46 47 49 49 4
9 50 51 52 52 53 54 54 54 55 56 58 58 58 60 60 61 66 66 67 67 68 69 70 70 72 72 72 72 72 73 73 74 76 78 79 79 83 85 85 87 88
88 89 89 90 92 93 95 95 95 95 97 97 97 98 99 100
Czas sortowania bucket_sort: 5 msec

Process returned 0 (0x0)   execution time : 1.273 s
Press any key to continue.
```

*Rysunek 6 Sortowanie dla ciągu liczb pesymistycznych*

W programie dodana jest funkcja generująca "losowe" ciągi elementów o zadanej długości  $N$  (zob. Rysunek 7) i zapisującą je do pliku tekstowego „wylosowane liczby.txt”.

```
"C:\Users\weron\Desktop\Projekt 2-Weronika Zagaja\bin\Debug\Projekt 2-Weronika Zagaja.exe"
1 - optimistic
2 - pessimistic
3 - expected
3
sorting random set of numbers...
Przed sortowaniem:
118 82 259 13 205 185 126 85 235 268 228 125 51 146 75 8 23 6 118 190 244 284 83 11 87 226 84 0 146 143 9 134 91 240 235 172
253 79 222 18 145 155 72 72 12 210 173 199 32 202 204 91 244 241 220 248 51 63 42 11 245 166 213 255 231 37 102 97 56 276 56
191 281 1 55 273 47 85 27 93 125 249 156 17 57 84 197 79 43 254 132 14 295 152 15 212 107 231 236 101 248 149 57 242 128 91 1
33 283 195 144 294 50 153 60 84 218 146 169 243 49 42 64 77 74 211 260 262 199 234 191 94 166 42 271 292 74 118 176 149 73 25
237 205 134 270 101 78 194 110 206 181 243 70 174 62 227 81 293 276 291 145 274 225 115 203 134 124 121 124 222 278 50 186 2
93 35 134 112 144 164 264 159 28 157 155 73 133 248 7 249 89 218 110 18 91 5 2 206 34 97 18 284 102 240 157 252 145 173 104 1
42 115 9 40 157 52 211 245 119 0 137 248 166 293 106 186 37 22 120 19 181 45 93 226 177 62 3 232 275 5 266 65 220 29 74 174 5
0 52 65 116 118 86 23 294 141 126 280 210 125 139 81 75 212 281 189 233 206 259 66 248 220 167 101 145 253 232 246 171 298 26
3 239 300 187 133 228 66 13 132 113 140 170 162 190 91 256 240 185 270 257 279 12 194

Po sortowaniu Insertion Sort :
0 0 1 2 3 5 5 6 7 8 9 9 11 11 12 12 13 13 14 15 17 18 18 18 19 22 23 23 25 27 28 29 32 34 35 37 37 40 42 42 42 43 45 47 49 50
50 51 51 52 52 55 56 56 57 57 59 60 62 62 63 64 65 65 66 66 70 72 72 73 73 74 74 74 75 75 77 78 79 79 81 81 82 83 84 84 84 8
5 85 86 87 89 91 91 91 91 93 93 94 97 97 101 101 101 102 102 104 106 107 110 110 112 113 115 115 116 118 118 118 119 1
20 121 124 124 125 125 125 126 126 128 132 132 133 133 133 134 134 134 134 137 139 140 141 142 143 144 144 145 145 145 14
6 146 146 149 149 152 153 155 155 156 157 157 157 159 162 164 166 166 166 167 169 170 171 172 173 173 174 174 176 177 181 181
185 185 186 186 187 189 190 190 191 191 194 194 195 197 199 199 202 203 204 205 205 206 206 206 210 210 211 211 212 212 213
218 218 220 220 220 222 222 225 226 226 227 228 228 231 231 232 232 233 234 235 235 236 237 239 240 240 240 241 242 243 243 2
44 244 245 245 246 248 248 248 248 248 249 249 252 253 253 254 255 256 257 259 259 260 262 263 264 266 268 270 270 271 273 27
4 275 276 276 278 279 280 281 281 283 284 284 291 292 293 293 293 294 294 295 298 300
Czas sortowania Insertion Sort : 0 msec

Po sortowaniu bucket_sort:
0 0 1 2 3 5 5 6 7 8 9 9 11 11 12 12 13 13 14 15 17 18 18 18 19 22 23 23 25 27 28 29 32 34 35 37 37 40 42 42 42 43 45 47 49 50
50 51 51 52 52 55 56 56 57 57 59 60 62 62 63 64 65 65 66 66 70 72 72 73 73 74 74 74 75 75 77 78 79 79 81 81 82 83 84 84 84 8
5 85 86 87 89 91 91 91 91 93 93 94 97 97 101 101 101 102 102 104 106 107 110 110 112 113 115 115 116 118 118 118 119 1
20 121 124 124 125 125 125 126 126 128 132 132 133 133 133 134 134 134 134 137 139 140 141 142 143 144 144 145 145 145 14
6 146 146 149 149 152 153 155 155 156 157 157 157 159 162 164 166 166 166 167 169 170 171 172 173 173 174 174 176 177 181 181
185 185 186 186 187 189 190 190 191 191 194 194 195 197 199 199 202 203 204 205 205 206 206 206 210 210 211 211 212 212 213
218 218 220 220 220 222 222 225 226 226 227 228 228 231 231 232 232 233 234 235 235 236 237 239 240 240 240 241 242 243 243 2
44 244 245 245 246 248 248 248 248 248 249 249 252 253 253 254 255 256 257 259 259 260 262 263 264 266 268 270 270 271 273 27
4 275 276 276 278 279 280 281 281 283 284 284 291 292 293 293 293 294 294 295 298 300
Czas sortowania bucket_sort: 0 msec

Process returned 0 (0x0)   execution time : 1.645 s
```

Rysunek 7 Sortowanie liczb losowych dla  $N=300$

Dodatkowo program zawiera możliwość zapisu wyników do pliku testowego „wyniki sortowania.txt” (zob. Rysunek 8 ).

```
wyniki sortowania — Notatnik
Plik Edycja Format Widok Pomoc
Przed sortowaniem:
100 99 98 97 97 97 95 95 95 95 93 92 90 89 89 88 88 87
85 85 83 79 79 78 76 74 73 72 72 72 72 70 70 69
68 67 67 66 66 61 60 60 58 58 56 55 54 54 53 52
52 51 50 49 49 49 47 46 46 46 45 44 42 41 39 37 36 34
33 32 32 30 30 30 28 27 26 25 24 23 22 21 21 20 19
18 16 14 13 13 12 12 10 2 0

Po sortowaniu Insertion Sort :
0 2 10 12 12 13 13 14 16 18 19 20 21 21 22 23 24 25 26
27 28 28 30 30 30 32 32 33 34 36 37 39 41 42 44 45 46
46 46 47 49 49 49 50 51 52 52 53 54 54 54 55 56 58 58
58 60 60 61 66 66 67 67 68 69 70 70 72 72 72 72 73
73 74 76 78 79 79 83 85 85 87 88 88 89 89 90 92 93 95
95 95 95 97 97 97 98 99 100

Po sortowaniu bucket_sort: 0 2 10 12 12 13 13 14 16 18
19 20 21 21 22 23 24 25 26 27 28 28 30 30 30 32 32 33
34 36 37 39 41 42 44 45 46 46 46 47 49 49 49 50 51 52
52 53 54 54 54 55 56 58 58 58 60 60 61 66 66 67 67 68
69 70 70 72 72 72 72 73 74 76 78 79 79 83 85 85
87 88 88 89 89 90 92 93 95 95 95 95 97 97 98 99 100

Lin 1, kol 1   100%   Windows (CRLF)   UTF-8
```

Rysunek 8 Zapis wyniku sortowań dla ciągu pesymistycznego w pliku tekstowym

## 8. Złożoność obliczeniowa

### Sortowanie przez wstawianie

Złożoność algorytmu wynosi  $O(n^2)$ . Istnieje jednak modyfikacja algorytmu, pozwalająca zmniejszyć liczbę porównań. Zamiast za każdym razem iterować po już posortowanym fragmencie (etap wstawiania elementu), można posłużyć się wyszukiwaniem binarnym. Pozwala to zmniejszyć liczbę porównań do  $O(n \log n)$ , nie zmienia się jednak złożoność algorytmu, ponieważ liczba przesunięć elementów to nadal  $O(n^2)$ .

### Sortowanie kbelkowe

Algorytm ma klasę czasowej złożoności obliczeniowej  $O(m + n)$ , gdzie  $m$  oznacza ilość możliwych wartości, które mogą przyjmować elementy zbioru, a  $n$  to ilość sortowanych elementów. Jeśli  $m$  jest małe w porównaniu z  $n$  (sortujemy dużo elementów o małym zakresie wartości), to na czas sortowania będzie miała wpływ głównie ilość elementów  $n$  i klasa złożoności uprosi się do postaci  $O(n)$ . Dzieje się tak dlatego, iż przy równomiernym rozkładzie dużej ilości elementów o małym zakresie wartości liczniki będą równomiernie wypełnione. W sytuacji odwrotnej, gdy sortujemy mało elementów o dużym zakresie wartości klasa złożoności zredukuje się z kolei do  $O(m)$ . W przypadku ogólnym pesymistyczna złożoność obliczeniowa tego algorytmu wynosi  $O(n^2)$ .

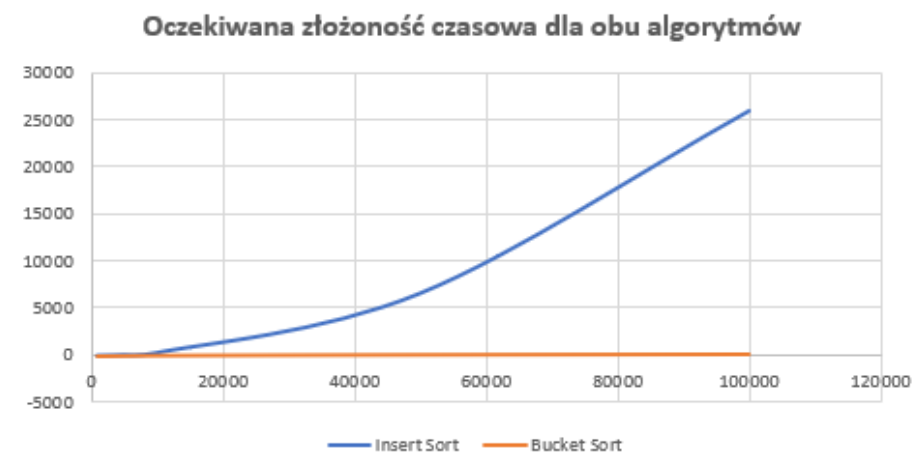
### Wykresy porównujące

W celu oszacowania czasu sortowań dla zadanej tablicy  $N$  liczb losowych została wyliczona średnia z przeprowadzonych testów (zob. Rysunek 9). Dodatkowo czas sortowań wyrażony jest w milisekundach, aby lepiej zauważyć różnice sortowań obu algorytmów. Ponadto trzeba zwrócić uwagę, że czas sortowań może się różnić na różnych procesorach komputerów.

### Oczekiwana złożoność czasowa

N	1000		5000		10000		50000		100000	
rodzaj sortowania	Insert Sort	Bucket Sort	Insert Sort	Bucket Sort	Insert Sort	Bucket Sort	Insert Sort	Bucket Sort	Insert Sort	Bucket Sort
Proba 1	10	0	65	0	278	0	6920	10	25912	6
Proba 2	0	0	65	0	280	9	6510	0	26054	6
Proba 3	2	1	65	0	320	0	6494	8	26222	8
Proba 4	0	0	66	2	276	0	6549	3	26010	5
Proba 5	0	0	77	0	267	0	6561	0	25925	7
Proba 6	3	0	75	5	275	5	6512	8	25730	8
Średnia prób	2,5	0,16	68,83	1,16	282,6	2,33	6591	4,83	25975,5	6,6

Rysunek 9 Tabela czasów uzyskanych w sortowaniach



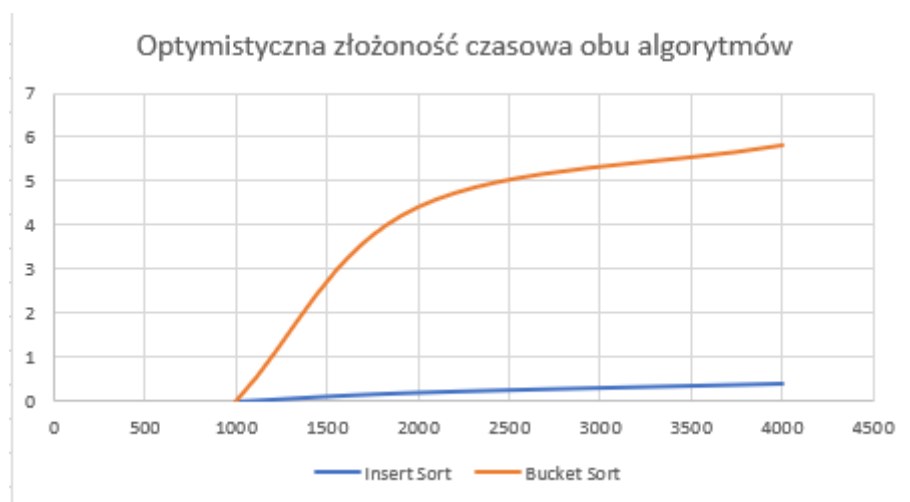
Rysunek 10 Wykres oczekiwanej złożoności czasowej dla obu algorytmów

Na wykresie można zauważyć, że wraz z rosnącymi próbkami danych  $N$  czas sortowania Insert Sort wzrasta parabolicznie, zaś dla sortowania Bucket Sort różnica czasów dla poszczególnych próbek danych jest bardzo mała, co przyczynia się do niewielkich zmian w wykresie.

#### Optymistyczna złożoność czasowa

N	1000		2000		4000	
	Insert Sort	Bucket Sort	Insert Sort	Bucket Sort	Insert Sort	Bucket Sort
próba 1	0	0	0	4	0	0
próba 2	0	0	0	2	1	11
próba 3	0	0	0	2	0	0
próba 4	0	0	1	4	0	8
próba 5	0	0	0	10	1	10
Średnia obliczeń	0	0	0,2	4,4	0,4	5,8

Rysunek 11 Tabela czasów uzyskanych w sortowaniach



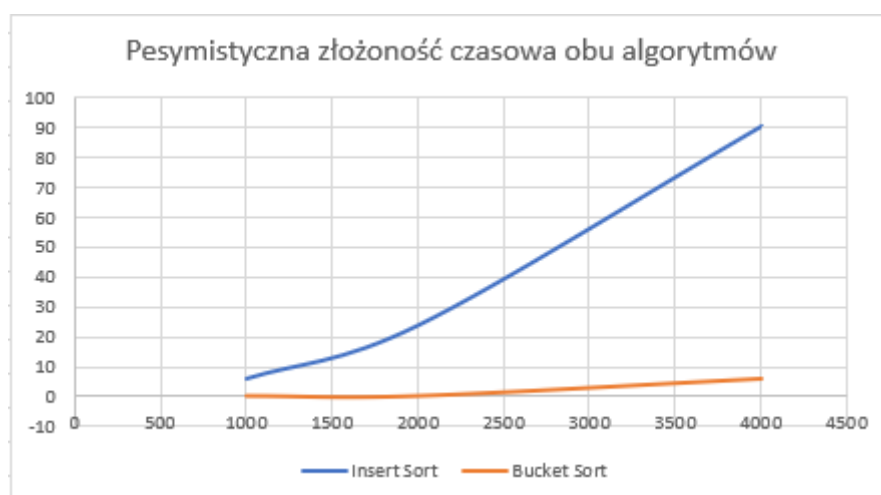
Rysunek 12 Wykres optymistycznej złożoności czasowej obu algorytmów

Na wykresie zauważyć można, że dla optymistycznych próbek danych czas sortowań obu algorytmów wzrasta logarytmicznie. Dodatkowo dostrzec można, że czas sortowania Insert Sort wzrasta szybciej niż czas sortowania Bucket Sort.

## Pesymistyczna złożoność czasowa

N	1000		2000		4000	
Rodzaj sortowania	Insert Sort	Bucket Sort	Insert Sort	Bucket Sort	Insert Sort	Bucket Sort
próba 1	10	0	30	0	90	0
próba 2	0	0	20	0	90	10
próba 3	10	0	25	0	90	10
próba 4	11	0	25	0	90	10
próba 5	10	0	20	0	92	0
Średnia obliczeń	6,2	0	24	0	90,4	6

Rysunek 13 Tabela czasów uzyskanych w sortowaniach



Rysunek 14 Wykres pesymistycznej złożoności czasowej obu algorytmów

Na ostatnim wykresie widać, że dla pesymistycznych próbek danych czas wzrasta w obu algorytmach parabolicznie. Podobnie jak dla złożoności optymistycznej czas sortowania Insert Sort wzrasta szybciej niż czas Bucket Sort.

Porównując wszystkie wykresy sortowania przez wstawianie, zauważamy, że dla oczekiwanych i pesymistycznych próbek danych wzrastają one parabolicznie. Jednak zauważyć można, że czas złożoności pesymistycznej jest dłuższy niż dla złożoności oczekiwanej. Podobnie do sortowania przez wstawianie zachowuje się sortowanie kubełkowe. Jednak różnice czasowe pomiędzy złożonością oczekiwaną, a pesymistyczną są niewielkie.

## 9. Wnioski

Projekt udało się zrealizować i pokazać jego prawidłowe działanie. Obie metody sortowania zostały zaimplementowane w osobnych funkcjach. Dodatkowo zaimplementowana została funkcja generująca losowe liczby. Program posiada możliwość zapisu i odczytu danych z plików tekstowych. Ponadto, charakteryzuje się możliwością wybierania przez użytkownika próbek danych do sortowania, dzięki temu może zauważyć różnice złożoności obliczeniowej dla różnych zestawów zarówno dla sortowania przez wstawianie, tak i dla sortowania kubełkowego. W projekcie zostało wykonanych kilka prób w celu oszacowania średniego czasu sortowań. Ponadto czas został podany w milisekundach, aby zauważyć różnice czasowe w poszczególnych próbkach danych. Warto zwrócić uwagę, że czasy te mogą się różnić na odmiennych procesorach. Dodatkowo w kodzie zostały umieszczone stosowne komentarze, które pozwalają odczytać kod. W projekcie znajdują się również pseudokody oraz schematy blokowe, pomagające zrozumienie algorytmów. W projekcie zostały zawarte wszystkie zagadnienia, dotyczące prawidłowego zrealizowania zadania.