Sortowanie bąbelkowe - wersja 1

wersja dokumentu 1.2

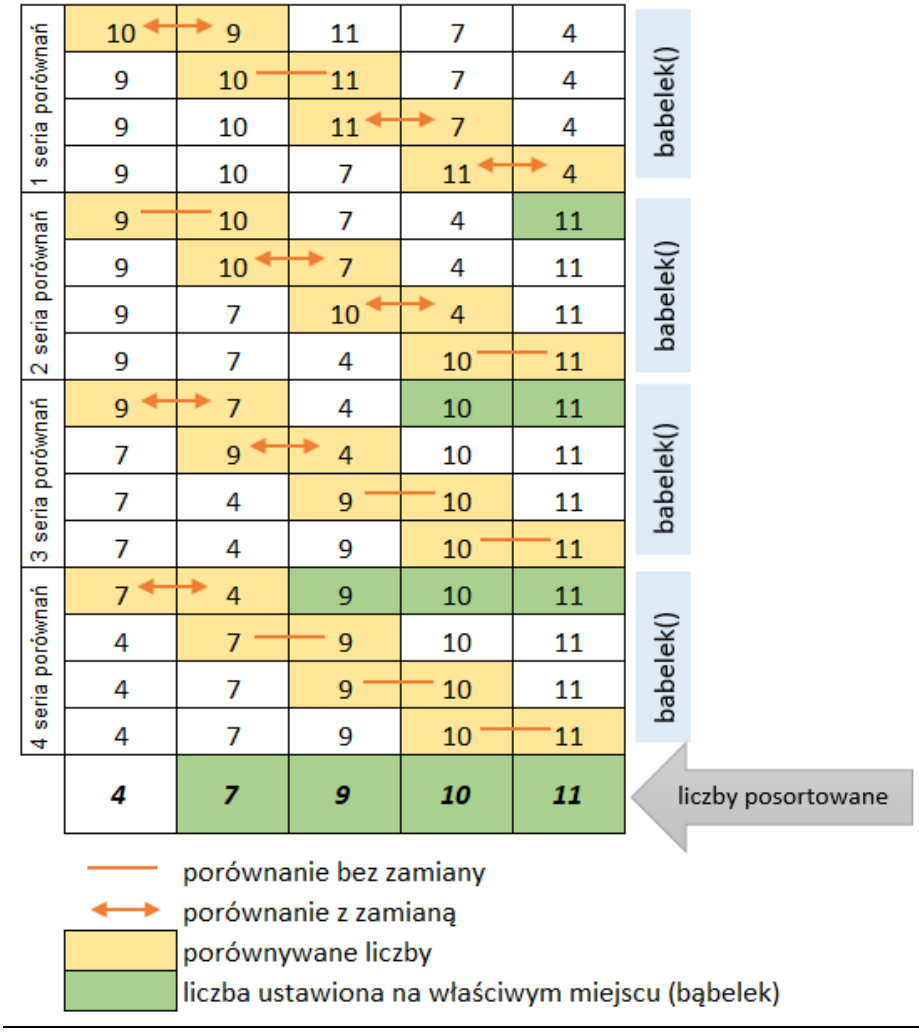
Wstęp

1. Sortowanie to uporządkowanie zbioru danych względem jakiegoś klucza. Jeden zbiór danych można różnie posortować w zależności od przyjętego kryterium i porządku.
2. Algorytmy sortowania są często stosowane w praktyce. Wiele bardziej złożonych algorytmów wykorzystuje posortowane elementy zbiorów.
3. Algorytmy sortowania ocenia się według:
    a. szybkości działania,
    b. ilości pamięci potrzebnej na ich realizację.

Sortowanie Bąbelkowe (Bubble Sort)

1. Porównujemy parami kolejne elementy zbioru i przestawiamy je (zamieniamy miejscami), jeśli są ustawione w złej kolejności (względem porządku, który chcemy uzyskać → malejąco czy rosnąco)
2. Zbiór jest przeglądany w tym samym kierunku tak długo, jak występuje w nim para elementów ustawionych w niewłaściwym kierunku.
3. Już w pierwszym cyklu sortowania jedna z liczb (**bąbelek**) zostaje ustawiona na właściwym miejscu (wypchnięta na wierzch). W zależności od oczekiwanego porządku (rosnąco czy malejąco) będzie to liczba największa lub najmniejsza.

Przykład sortowania rosnąco zbioru {10, 9, 11, 7, 4}



Uwagi:

- Po każdej serii porównań jeden *najcięższy bąbelek* jest wypchnięty na prawo (największa liczba z nieposortowanych jest na swoim miejscu).
- Dla zbioru, który ma 5 elementów w jednej serii wykonujemy 4 porównania i 4 serie porównań.
- Dla zbioru, który ma N elementów w jednej serii wykonujemy N-1 porównań i N-1 serii porównań.
- Dla zbioru o 5 elementach wykonujemy 16 operacji porównania.

Wypychamy jeden bąbelek na wierzch

1. Jedną serię porównań, w której największa liczba (z tych jeszcze nieposortowanych) trafi na swoje miejsce, możemy implementować w postaci funkcji ***babelek()***.
2. Przykładowa implementacja funkcji ***babelek()***:

```
void babelek(){
      for(int i=0; i<N-1; i++)
      if(liczby[i]>liczby[i+1]) swap(liczby[i], liczby[i+1]);
}
```

3. Uwagi:
   a. Tablicę *liczby[]* nie przekazujemy do funkcji jako argument. Jest ona zdefiniowana globalnie przed definicją funkcji *babelek()*.
   b. Funkcja ***swap(x, y)*** przypisuje wartość obiektu x obiektowi y i wartość obiektu y obiektowi x. („zamienia x i y miejscami”).
   c. Funkcja *babelek()* realizuje 1 serię porównań z przykładu.

Sortujemy

1. Aby zaimplementować sortowanie funkcję *babelek()* użyjemy N-1 razy.
2. Przykładowy kod programu:

```
#include <iostream>
using namespace std;

const int N=5; //ilość liczb w tablicy
int liczby[N]={1,9,11,7,4}; //tablica liczb do posortowania

void pokazDane(){
      for(int i=0; i<N; i++) cout << liczby[i] << " ";
}

void babelek(){
      for(int i=0; i<N-1; i++)
      if(liczby[i]>liczby[i+1]) swap(liczby[i], liczby[i+1]);
}
void sortBabelkowe(){
      //N-1 razy wywlołujemy funkcję babelek()
      for(int i=0; i<N-1; i++) babelek();
}

main(){
      cout << "\nliczby do posortowania:\n";
      pokazDane();
      sortBabelkowe();
      cout << "\nliczby posortowane rosnaco:\n";
      pokazDane();
      return 0;
}
```

Wynik działania programu w konsoli:

```
liczby do posortowania:
1 9 11 7 4
liczby posortowane rosnaco:
1 4 7 9 11
```

Sortowanie bąbelkowe - wersja 2 - trochę lepsza

1. Można zauważyć, że w każdej kolejnej serii ilość porównań można ograniczyć korzystając z tego, że zawsze po jednej serii kolejny bąbelek jest we właściwym miejscu.

ilość porównań					
4	10 ↔	9	11	7	4
	9	10 —	11	7	4
	9	10	11 ↔	7	4
	9	10	7	11 ↔	4
3	9 —	10	7	4	**11**
	9	10 ↔	7	4	**11**
	9	7	10 ↔	4	**11**
2	9 ↔	7	4	**10**	**11**
	7	9 ↔	4	**10**	**11**
1	7 ↔	4	**9**	**10**	**11**
	4	***7***	***9***	***10***	***11***

2. W porównaniu do wersji poprzedniej wykonujemy mniej operacji porównania: 4+3+2+1=10 zamiast 16.
3. Przykładowy kod programu
   a. bez użycia funkcji,
   b. z ustaleniem ilości liczb w tablicy (zmienna **N**),
   c. z losowaniem liczb do tablicy z podanego przedziału ***<start, stop>***:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
main(){
	int start, stop, N;
	cout << "podaj poczatek zakresu losowania liczb: "; cin >> start;
	cout << "podaj koniec zakresu losowania liczb: "; cin >> stop;
	cout << "podaj ilosc liczbdo posortowania: "; cin >> N;
	int liczby[N];
	//wprowadzanie losowych liczb z zakresu <start, stop> do tablicy
	for(int i = 0; i < N; i++) liczby[i]=rand()%(stop-start+1)+start;
	cout << "\nliczby do posortowania:\n";
	for(int i = 0; i < N; i++) cout << liczby[i] << " ";

	//sortowanie rosnąco
	//Zewnętrzna pętla N-1 razy. W każdym obrocie jeden najcięższy bąbelek wypływa
	na wierzch (ustawia się na dobrym miejscu).
	for(int i=0; i<N-1; i++){
		//Dla każdego i wykonujemy N-1-i operacji porównania !!!
		for(int j=0; j<N-1-i; j++)
		if(liczby[j]>liczby[j+1]) swap(liczby[j], liczby[j+1]);
	}
	//wyświetlanie wyniku
	cout << "\nliczby posortowane rosnaco:\n";
	for(int i = 0; i < N; i++) cout << liczby[i] << " ";
	return 0;
}
```

4. Wynik działania programu dla 100 liczb:

```
liczby do posortowania:
41 176 103 169 74 46 21 12 28 143 77 5 166 144 112 89 181 83 3 9 30 132 83 153 91
 121 135 23 20 197 20 18 98 81 60 13 140 169 151 45 161 93 167 183 186 41 66 73 1
13 34 20 107 100 195 138 196 80 93 193 175 55 20 80 33 87 157 196 98 169 136 110
87 11 96 49 74 132 147 72 131 132 161 165 55 63 182 142 140 62 178 60 78 104 67 1
10 31 38 72 15 59
liczby posortowane rosnaco:
3 5 9 11 12 13 15 18 20 20 20 20 21 23 28 30 31 33 34 38 41 41 45 46 49 55 55 59
60 60 62 63 66 67 72 72 73 74 74 77 78 80 80 81 83 83 87 87 89 91 93 93 96 98 98
100 103 104 107 110 110 112 113 121 131 132 132 132 135 136 138 140 140 142 143 1
44 147 151 153 157 161 161 165 166 167 169 169 169 175 176 178 181 182 183 186 19
3 195 196 196 197
```