T: Sortowanie danych - podstawowe pojęcia.

1.**Sortowanie - uporządkowanie wyrazów**:

1.1) porównywanie elementów ciagu i ich zamiana

a) **bąbelkowe**

b) **przez wybór**

c) **przez wstawianie**

d) przez scalenie (technika *"dziel i zwyciężaj"*)

e) metoda szybka (ang. *quick sort*, technika *"dziel i zwyciężaj"*)

1.2) porządkowe - realizowane z wykorzystaniem dodatkowej tablicy pomocniczej:

a) **zliczanie**

b) sortowanie kubełkowe

2.Algorytmy sortujące w miejscu (łac. in situ) - wykorzystujace jedną tablicę

3.Algorytmy sortujące:

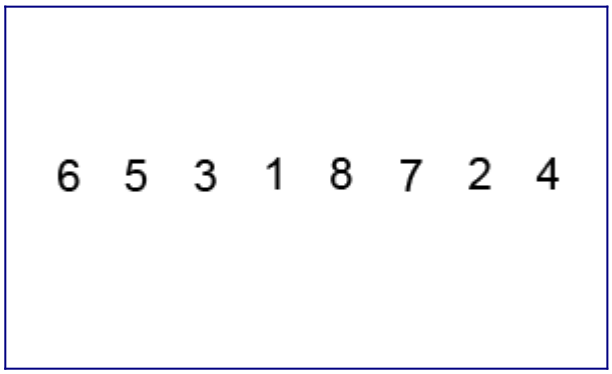
a) stabilne - zachowują kolejność elementów równych (innymi słowy nie zmieniają kolejności względem siebie tych wyrazów ciągu, które mają tę samą wartość) .

b) niestabilne - kolejność wynikowa elementów równych jest nieokreślona

Algorytm jest niestabilny. Przykładowa lista to: [2**a**,2**b**,1] → [**1**,2**b**,2**a**] (gdzie 2b=2a)

4. Sortowanie bąbelkowe (ang. *bubblesort*) prosta metoda sortowania o złożoności czasowej $O(n^2)$ i pamięciowej O(1) .

Polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli zaburza ona porządek, w jakim się sortuje tablicę. Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany.



Ciąg wejściowy **[4,2,1,5,7]** . Każdy wiersz symbolizuje wypchnięcie kolejnego największego elementu na koniec („wypłynięcie największego bąbelka”). Niebieskim kolorem oznaczono końcówkę ciągu już posortowanego.

$$[\underbrace{4,2}_{4>2},5,1,7] \rightarrow [2,\underbrace{4,5}_{4<5},1,7] \rightarrow [2,4,\underbrace{5,1}_{5>1},7] \rightarrow [2,4,1,\underbrace{5,7}_{5<7}]$$

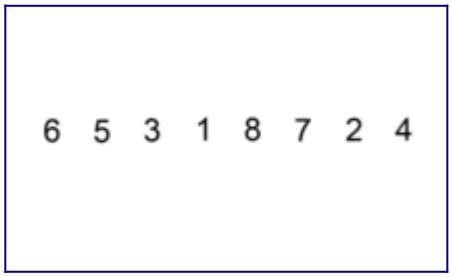
$$[\underbrace{2,4}_{2<4},1,5,7] \rightarrow [2,\underbrace{4,1}_{4>1},5,7] \rightarrow [2,1,\underbrace{4,5}_{4<5},7]$$

$$[\underbrace{2,1}_{2>1},4,5,7] \rightarrow [1,\underbrace{2,4}_{2<4},5,7]$$

$$[\underbrace{1,2}_{1<2},4,5,7]$$

```
void babelkowe(int rozmiar,int *T){
    int POM;
    for(int j=rozmiar-1;j>0;j--){
     for(int i=0;i<j;i++){
        if (T[i]>T[i+1]) {
            //zamiana
            POM=T[i];
            T[i]=T[i+1];
            T[i+1]=POM;
        }
     }
  }
}
```

5. Sortowanie przez wstawianie (ang. *Insert Sort, Insertion Sort*) – jeden z najprostszych algorytmów sortowania, którego zasada działania odzwierciedla sposób w jaki ludzie ustawiają karty – kolejne elementy wejściowe są ustawiane na odpowiednie miejsca docelowe.



	Stąd pobieramy elementy:					Tutaj wstawiamy elementy, jednocześnie sortując:				
	7	3	0	1	5					
Krok 1.		3	0	1	5	7				
Krok 2.			0	1	5	3	7			
Krok 3.				1	5	0	3	7		
Krok 4.					5	0	1	3	7	
Krok 5.						0	1	3	5	7

	7	3	0	1	5
Krok 1.	3	7	0	1	5
Krok 2.	0	3	7	1	5
Krok 3.	0	1	3	7	5
Krok 4.	0	1	3	5	7

1. Utwórz zbiór elementów posortowanych i przenieś do niego dowolny element ze zbioru nieposortowanego.
2. Weź dowolny element ze zbioru nieposortowanego.
3. Wyciągnięty element porównuj z kolejnymi elementami zbioru posortowanego, póki nie napotkasz elementu równego lub elementu większego (jeśli chcemy otrzymać ciąg niemalejący) lub nie znajdziemy się na początku/końcu zbioru uporządkowanego.
4. Wyciągnięty element wstaw w miejsce, gdzie skończyłeś porównywać.
5. Jeśli zbiór elementów nieuporządkowanych jest niepusty, wróć do punktu 2.

Lista kroków:

Krok 0. Wczytaj ***n, T[0...n-1]***.

Krok 1. Dla kolejnych wartości ***i***: ***1, 2, ..., n-1***, wykonuj kroki ***2. - 3.***, a następnie przejdź do kroku **4.**

Krok 2. Przypisz ***pom = T[i]*** (pobranie elementu do wstawienia).

Krok 3. Porównuj element ***pom*** z kolejnymi wyrazami uporządkowanego podciągu ***T[k]***, dla ***k= i-1, i-2, ..., 0,*** przesuń sprawdzone elementy, zwiększając im indeks o 1 w celu zrobienia miejsca dla ***pom***, i umieść element ***pom*** przed pierwszym elementem ***T[k]***, który będzie spełniał warunek ***T[K] ≤pom***. Przyimek „przed” należy tu rozumieć jako „na pozycji o numerze o jeden wyższym”. (Wstawienie elementu ***pom = T[i]*** do posortowanego podciągu ***T[0...i-1]*** w taki sposób, aby po dodaniu tego elementu podciąg nadal był uporządkowany).

Krok 4. Wypisz elementy tablicy ***T[0...n-1]***. Zakończ algorytm.

6. Sortowanie przez wybieranie (ang. *selection sort*)- jedna z prostszych metod sortowania o złożoności **O(n^2).** Polega na wyszukaniu elementu mającego się znaleźć na żądanej pozycji i zamianie miejscami z tym, który jest tam obecnie. Operacja jest wykonywana dla wszystkich indeksów sortowanej tablicy.

Algorytm przedstawia się następująco:

1. wyszukaj minimalną wartość z tablicy spośród elementów od `i` do końca tablicy
2. zamień wartość minimalną, z elementem na pozycji `i`

nr iteracji (wartość i)	tablica	minimum
0	[**9,1,6,8,4,3,2,0**]	0
1	[0,**1,6,8,4,3,2,9**]	1 (element znajduje się na właściwej pozycji)
2	[0,1,**6,8,4,3,2,9**]	2
3	[0,1,2,**8,4,3,6,9**]	3
4	[0,1,2,3,**4,8,6,9**]	4 (...)
5	[0,1,2,3,4,**8,6,9**]	6
6	[0,1,2,3,4,6,**8,9**]	8 (...)

```
void przez_wybor( int rozm,double T[]){
 int k;
 double pom;
 for (int i=0;i<rozm-1;i++){
  k=i;
  for (int j=i+1;j<rozm;j++)
   if (T[j]<T[k]) k=j;
  pom=T[k];
  T[k]=T[i];
  T[i]=pom;
 }
}
```

7. Sortowanie przez zliczanie (ang. *counting sort*) – metoda sortowania danych, która polega na wyznaczamy liczbę wystąpień każdej z wartości ***i = 0, 1, ..., m-1*** w tablicy zawierającej sortowany ciąg ***T[0...n-1]***. Uzyskana informacja pozwala na szybkie znalezienie pozycji elementów ciągu w sortowanej tablicy. W omawianym algorytmie sortowania nie porównujemy elementów, tylko obliczamy liczbę ich wystąpień.

Algorytm zakłada, że klucze elementów należą do skończonego zbioru (np. są to liczby całkowite z przedziału 0..100), co ogranicza możliwości jego zastosowania.

***P*:**

k	0	1	2	3	4
$P[k]$	0	0	0	0	0

Realizacja algorytmu przebiega następująco:

Krok 1.

***T*:**

4 3 3 4 0 2 1 4

***P*:**

k	0	1	2	3	4
$P[k]$	0	0	0	0	**1**

Krok 2.

***T*:**

4 **3** 3 4 0 2 1 4

***P*:**

k	0	1	2	3	4
$P[k]$	0	0	0	**1**	1

Krok 3.

***T*:**

4 3 **3** 4 0 2 1 4

***P*:**

k	0	1	2	3	4
$P[k]$	0	0	0	**2**	1

Krok 4.

***T*:**

4 3 3 **4** 0 2 1 4

***P*:**

k	0	1	2	3	4
$P[k]$	0	0	0	2	**2**

Krok 5.

***T*:**

4 3 3 4 **0** 2 1 4

***P*:**

k	0	1	2	3	4
$P[k]$	**1**	0	0	2	2

Krok 6.

***T*:**

4 3 3 4 0 **2** 1 4

***P*:**

k	0	1	2	3	4
$P[k]$	1	0	**1**	2	2

Krok 7.

***T*:**

4 3 3 4 0 2 **1** 4

***P*:**

k	0	1	2	3	4
$P[k]$	1	**1**	1	2	2

Krok 8.

***T*:**

4 3 3 4 0 2 1 **4**

***P*:**

k	0	1	2	3	4
$P[k]$	1	1	1	2	**3**

Wynik:

***T*:** **0 1 2 3 3 4 4 4**

```
void zliczanie (int rozm,int T[], int m)
{
/* m – wartosc od ktorej wszystkie wartosci w tablicy sa
mniejsze */
 int P[m]={0};
 for (int i=0;i<rozm;i++) P[T[i]]++;
 int k=0;
 for (int i=0;i<m;i++)
  for (int j=P[i];j>=1;j--)
  {
   T[k]=i;
   k++;
  }
}
```

8. Ćwiczenie na lekcji - samodzielna implementacja na podstawie listy kroków ***sortowania przez wstawianie***