ALGORYTMY I STRUKTURY DANYCH – SORTOWANIE

- **dane**: kolekcja elementów $<x_1, x_2, \ldots, x_n>$
- **wynik**: kolekcja $< x_1', x_2', \ldots, x_n'>$: $x_1' \le x_2' \le \ldots \le x_n'$
  - uporządkowanie ze względu na wartość wybranej własności elementów (key)

Sortowanie przez proste wybieranie

```
for(i = 1 to n-1)
    min ← i
    for(j=i+1 to n)
        if(key[x_j]< key[x_min])
            min← j
    zamień x_i i x_min
```

- **złożoność**: $O(n^2)$

Sortowanie przez proste wstawianie

```
for(i = 2 to n)
    x ← x_i
    j ← i-1
    while(j≥1 and key[x]< key[x_j])
        przesuń x_j na pozycję x_j+1
        j← j-1
    wstaw x na pozycję x_j+1
```

- **złożoność**: $O(n^2)$

Sortowanie przez prostą zamianę

```
for(i = 2 to n)
    for(j = n downto i)
        if((key[x_j]< key[x_j-1])
            zamień x_j i x_j-1
```

- **złożoność**: $O(n^2)$

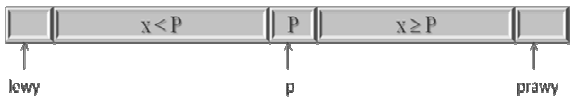
Sortowanie szybkie

QuickSort(INT lewy, prawy)

```
if(lewy < prawy)
    wyznacz P i jego pozycję p
    rozmieść elementy względem P
    QuickSort(lewy, p-1)
    QuickSort(p+1, prawy)
```

P – element osiowy, p – indeks P

Element osiowy



Wybór P

- skrajnie prawy, skrajnie lewy
- losowy
- środkowy
- mediana

- **złożoność**: O(n log(n) (pesymistyczna - $O(n^2)$)

✖ Sortowanie przez scalanie

<u>MergeSort(lewy, prawy)</u>

```
if (lewy < prawy)
  s = (lewy+prawy)/2
  MergeSort(lewy, s)
  MergeSort(s, prawy)
  merge(lewy, prawy, s)
```

+ **złożoność**: O(n log(n)) (QuickSort – pesymistyczne $O(n^2)$)

✖ Sortowanie pozycyjne

- ✖ **dane**: kolekcja n elementów $<x_1, x_2, \ldots, x_n>$: $x = x^1x^2\ldots x^l$ (l – liczba pozycji) i $x^i \in Z$ (Z – zbiór symboli)
- ✖ **wynik**: uporządkowana kolekcja

```
for (i = l downto 1)
  for (j = 1 to n)
    K^i ← x_j^i
  for (k = 1 to |K|)
    X ← K^k
```

K^i – kubełek odpowiadający symbolowi na i-tej pozycji j-tego obiektu

+ **złożoność**: O(n)

✖ Sortowanie przez zliczanie

- ✖ **dane**: kolekcja n elementów $X = <x_1, x_2, \ldots, x_n>$
- ✖ **wynik**: uporządkowana kolekcja $Y = < y_1, y_2, \ldots, y_n>$
- ✖ **założenia**:
  - ✖ C – ciąg pomocniczy o długości k: $c_i = 0$, i = 1, …, k
  - ✖ $x_i \in <1,\ldots,k>$, i = 1, …, n

1.

```
for (i = l to n)
  ip ← x_i
  c_ip ← c_ip +1
for (i = 2 to k)
  c_i ← c_i + c_i -1
for(i = n downto 1)
  ip ← x_i , jp ← c_ip
  y_jp ← x_i, c_ip ← c_ip
```

+ **złożoność**: O(n)