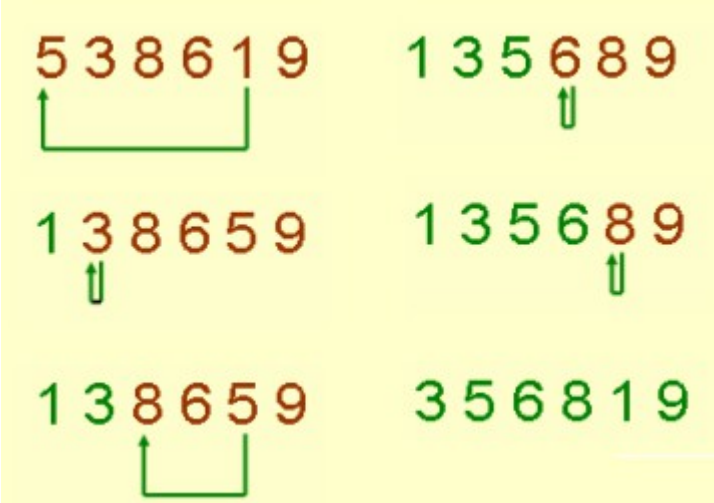
1. **Sortowanie przez proste wybieranie - (Selectionsort)**

Sposób postępowania:

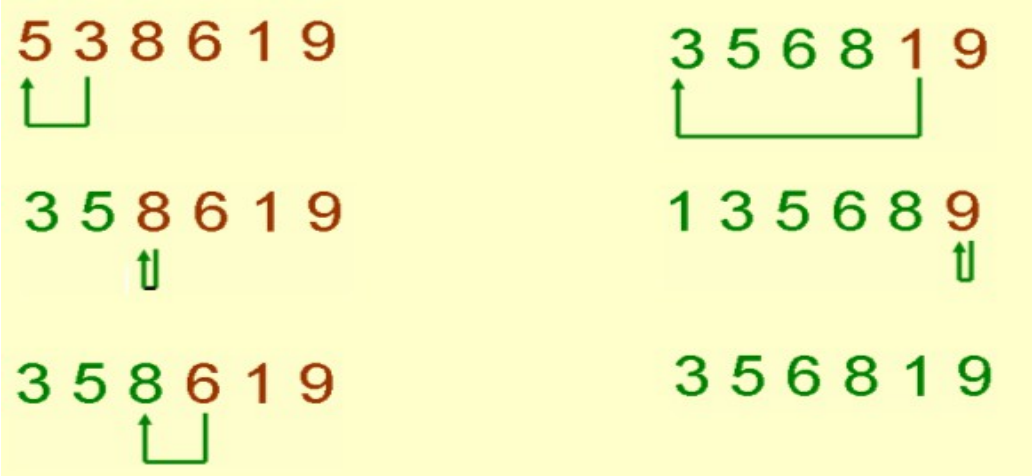
- **Wyznaczamy najmniejszy element w ciągu a[0]…a[n-1] - zamieniamy miejscami z pierwszym elementem ciągu**
- **Wyznaczamy najmniejszy element w a[1]...a[n-1] - zamieniamy z drugim elementem w ciągu…**
- **Wyznaczamy najmniejszy element w a[n-2],a[n-1] - zamieniamy z (n-1)-szym elementem w ciągu.**




```
void prostewybieranie();
int k,x;     //k-indeks minimalnego elementu; x-obiekt minimalny
{
for (int i=0;i< n;i++)
        {
          k=i;x=a[i];   //szukamy elementu minimalnego w ciągu a[i]...a[n]
          for (int j=i+1;j<n;j++)
          if (a[j]<x)
                {
                  k=j;
                  x=a[j]
                }
          a[k]=a[i];
          a[i]:=x;   //zamieniamy element i-ty z min
        }
}
```

2.Sortowanie przez proste wstawianie - (Insertionsort)

Sposób postępowania:
Dla każdego i=2, 3,...,n należy powtarzać wstawianie a[i] w już uporządkowaną część listy a[1]≤... ≤a[i-1]




```
void prostewstawianie()
int j,x;
{
        for (int i=2;i<=n;i++)
        {       x=a[i]; a[0]=x;
                j=i-1;
                while (x<a[j])
                { a[j+1]=a[j];
                  j=j-1;  //przesuwamy w prawo, robimy miejsce na
                          //nasz element
                 }
                a[j+1]=x;
        }
}
```

3. **Sortowanie przez prostą zamianę -** (Bubblesort)

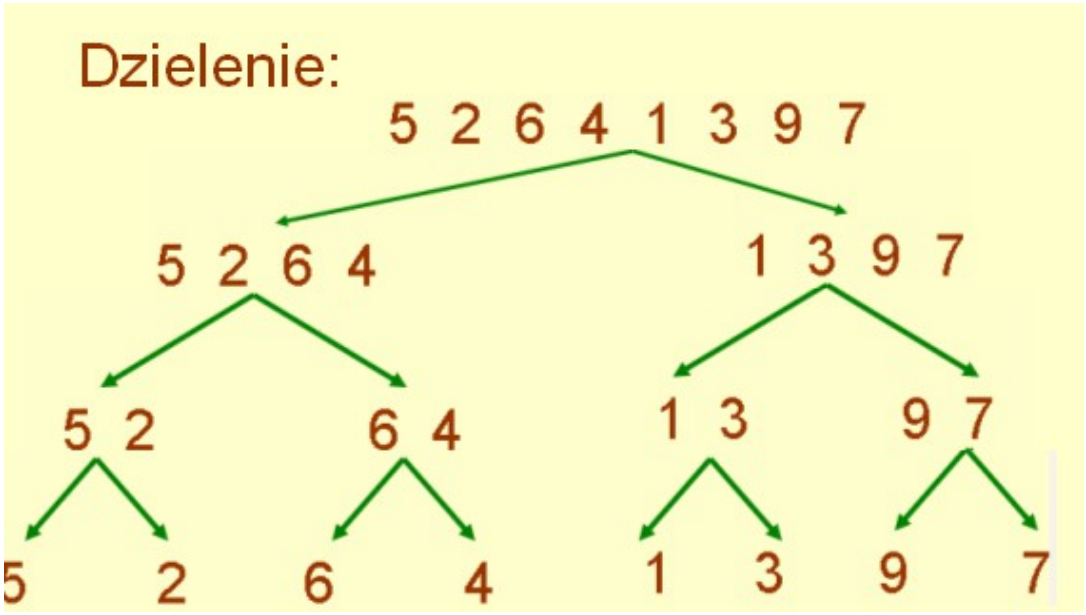
Sposób postępowania:
Algorytm opiera się na zasadzie porównywania i zamiany par sąsiadujących ze sobą obiektów (przeglądając tablice od prawej do lewej). W ten sposób po pierwszym kroku algorytmu w lewym końcu tablicy będzie najmniejszy element; po drugim kroku pierwsze dwa elementy tablicy będą posortowane itd...

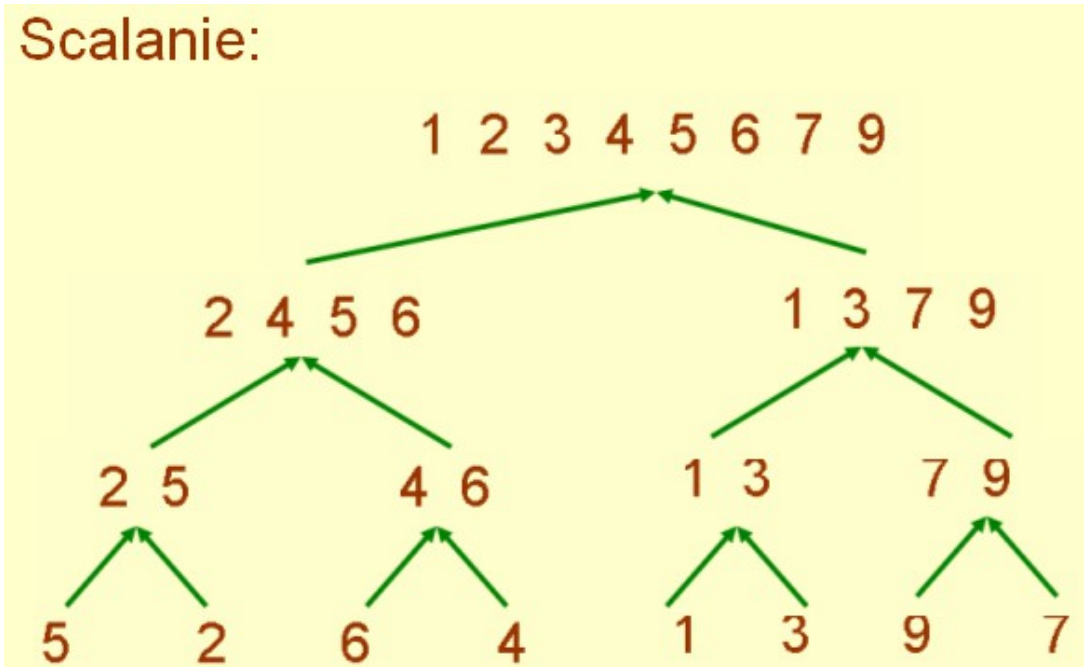

```
sortowaniebąbelkowe();
int x;
{
  for (int i=1; i<n; i++)
                  {for (int j=n-1; j>=i; j--)
                  if (a[j-1] >a[j])
                  {         //zamiana
                    x=a[j-1];
                    a[j-1]=a[j];
                a[j]=x;
                  }}
}
```

4. **Sortowanie przez scalanie -** (Mergesort)

Dane: int a[n]
Wyjście: tablica posortowana niemalejąco
Sposób postępowania:

- **Podziel ciąg a[0], a[1], a[2],…, a[n-1], na na dwa podciągi: a[0],…, a[m] oraz a[m+1],…, a[n-1]**
- **Posortuj rekurencyjnie każdy z podciągów a[0],…, a[m] oraz a[m+1],…, a[n-1]**
- **Scal oba podciągi w jeden.**





Algorytm (rekurencyjny!):

```
void mergesort (int l, int r);
{int m;
 if (l<r)
 {  m = (l+r) / 2 ;
    mergesort (l, m) ; //dzielenie
    mergesort (m+1, r) ;
    bitmerge (l, m, r) //scalanie
 }
}
```

Wywołanie procedury sortowania przez scalanie: mergesort (0, n-1) ;

Scalanie :

- **zastosujemy łatwiejszą w zaprogramowaniu**

wersje scalania:

scalanie ciągu bitonicznego.

- **tablica aux[0…n-1] – globalna tablica robocza**
- **Będziemy kopiować z tablicy a do aux oba sąsiednie ciągi, w prawym odwracając kolejność.**
- **W ten sposób powstanie ciąg bitoniczny: do połowy rosnący, potem malejący**

Zaimplementuj scalanie ciągu bitonicznego.