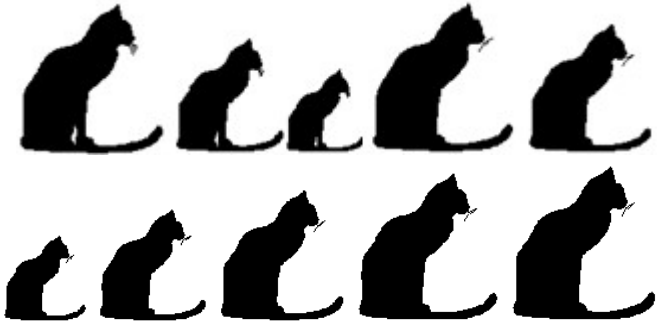
További rendező és kereső algoritmusok

- *Rendezés beszúrással*
- *Rendezés buborék algoritmussal*
- *Rendezés Shell algoritmussal*
- *Rendezés quick (gyors) algoritmussal*
- *Szekvenciális keresés*
- *Bináris keresés*
- *Összefésülés*



Rendezés beszúrással

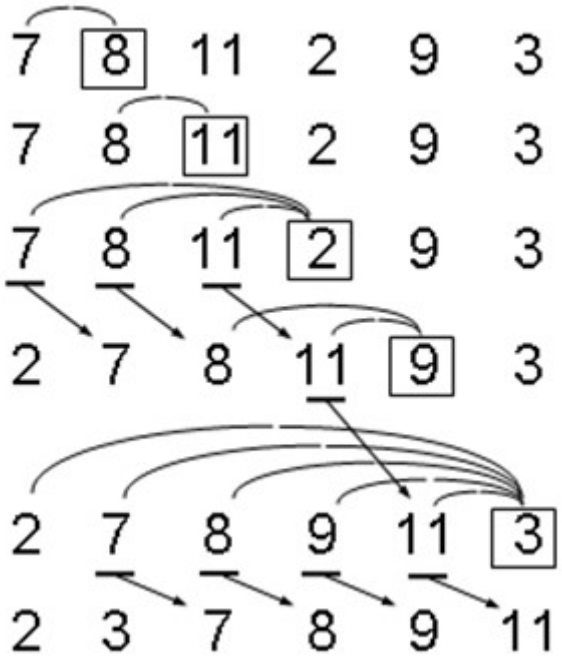
Rendezés: *célunk egy adott számsorozat növekvő sorrendbe rendezése.*

Példa:

Adott a rendezendő sorozat:
[7,8,4,9,1]

Rendezés után ezt várjuk:
[1,4,7,8,9].

Az algoritmus a kártyások módszerére emlékeztet: a felvett lapot beszúrják a már kézben lévő rendezett lapok közé a megfelelő helyre.



8 beszúrandó elem összehasonlítás eltolás

- ***A beszúrásos rendezés függénye***
  *(*Az *m* elemű *vekt* vektor rendezése*)*

```
void beszurasosRendezes(int* vekt, int m)
{
  int i, j, buf ;
  for (i =1; i < m; i++)
  {
     buf = vekt[ i ];
     j = i -1;
     while ( j >= 0 && buf < vekt[ j ] )
     {
          vekt[ j+1] = vekt[ j ];
          j--;
     }
     vekt[ j + 1] = buf;  /* beszúrás */
   }
}
```

Rendezés buborék algoritmussal

Az oszlop alakú rendezetlen vektorban a rendezés során a kisebb elemek buborék módjára emelkednek a nagyobb értékűek fölé.

		i = 1					*i* = 2				*i* = 3			*i* = 4		*i*=5	
j	0.	50	50	50	50	50	10	10	10	10	10	10	10	10	10	10	10
	1.	60	60	60	60	10	50	50	50	50	20	20	20	20	20	20	20
	2.	30	30	30	10	60	60	60	60	20	50	50	50	30	30	30	30
	3.	10	10	10	30	30	30	30	20	60	60	60	30	50	50	40	40
	4.	40	20	20	20	20	20	20	30	30	30	30	60	60	40	50	50
	5.	20	40	40	40	40	40	40	40	40	40	40	40	40	60	60	60

- ***A buborék rendezés függvénye***
  *(*Az *m* elemű *vekt* vektor rendezése*)*

```
void Buborek(int* vekt, int m)
{
  int i, j, buf ;
  for (i =1; i < m; i++)
  {
     for (j = m-1; j >= i ; j--)
     {
        if (vekt[ j-1 ] > vekt[ j ] )
        {
           buf = vekt[ j -1] ;
           vekt[ j-1] = vekt[ j ];
           vekt[ j ] = buf ;
        }
     }
  }
}
```

- ***Rendezés Shell algoritmussal***

  A Donald L. Shell nevéhez fűződő algoritmus az előzőekkel ellentétben, amelyek kb. 200 elemig hatékonyak, nagy elemszámú vektorok rendezésére előnyös. Alapgondolata, hogy ne a közvetlenül szomszédos elemek összehasonlításával végezzük a rendezést. Nagyméretű vektoroknál arányaiban kevesebb adatmozgatást kíván és ezért gyorsabb.

- ***Rendezés quick (gyors) algoritmussal***

  Ez a speciális cserealgoritmus C. A. R. Hoare találmánya. Nevéhez méltóan gyors, különösen nagyméretű vektoroknál. Lefutása függ az ún. pivot elem szerencsés megválasztásától is.

  Megjegyzések:
  A rendezés elvégezhető szövegkonstansokat tartalmazó karaktervektorokon is. Ekkor a karakterek ASCII kódját vesszük alapul az összehasonlításnál.
  A memóriában történő rendezést belső, a fájlelemeken fájlban végzett rendezést külső rendezésnek is nevezik.

- ***Kereső algoritmusok***

  Feladat: egy vektor megadott értékkel egyező elemének megtalálása, vagy annak megállapítása, hogy ilyen elem nincs. Sikeres keresés megadja az elem indexét.

- ***Szekvenciális keresés***

  Rendezetlen sorozaton is elvégezhető. Az elemeket sorra összehasonlítjuk a keresett értékkel. Egyezés esetén megadjuk az elem indexét. Lassú, a sebessége függ az elemnek a sorozatban elfoglalt helyétől. (V.ö. Kiválasztással)

- ***Bináris keresés***

  Csak rendezett sorozaton alkalmazható. Nagyon gyors. Alapgondolata, hogy minden lépésben elhagyja a maradéksorozat azon felét, mely a keresett értéket biztosan nem tartalmazza. Minden lépésben az ((alsóindex+felsőindex) / 2) *indexű elemmel hasonlítja össze a keresett értéket. Ha az elem kisebb, akkor az elemmel végződő alsó, ha nagyobb, akkor az elemmel kezdődő felső félsorozatot hagyja el. Egyenlőség esetén megtalálta a keresett elemet. Átlagos esetben* $\log_2$*(maxindex) lépés elegendő.*

Példa

A sorozat: 1 3 7 9 12 19 27
Keresett érték = **12**

1. lépés: i = (0+6)/2; (=3) 9<**12**
A maradéksorozat: 12 19 27
2. lépés: i = (4+6)/2; (=5) 19<**12**
A maradéksorozat: 12
3. lépés: i = (4+4)/2; (=4) 12==**12**
➡ megtalálta.

- ***A bináris keresés függvénye***
  *(*Az *m* elemű *vekt* vektor rendezése*)*

```
void Bin_keres(int* vekt, int m, int keresett )
{
  int a, f, k, talalt ;
  a = 0; f = m-1; talalt = 0;
  while (a <= f  &&  !talalt )
  {
     k = ( a + f ) / 2 ;
     if (keresett < vekt[ k ] )
        f = k-1;
     else if (keresett > vekt[ k ] )
                 a = k+1;
           else
                talalt = 1;
   }
   if (talalt) return k;
   else return -1;
}
```

- ***Összefésülés (merge)***
  Célja két, azonos típusú értékekből álló rendezett sorozat egyesítése a rendezettség megtartásával. A sorozat fájl is lehet.

Példa:

S1: 2,4,5,5,7
S2: 1,3,4,6,9 → S1_2: 1,2,3,4,4,5,5,6,7,9

Az összefésülés függvénye

```
void Merge(int* S1, int* S2, int* S1_2, int m1, int m2 )
{
  int i1=0, i2=0, k=0 ;
  while (k < m1 + m2 )
    if (i1 < m1  &&  i2 < m2)
    {
      if (S1[ i1 ] <= S2[ i2 ] )
      {
         S1_2[ k ] = S1[ i1 ];
         i1++;
      }
      else   { S1_2[ k ] = S2[ i2 ]; i2++; }
      k++;
    }
    else
      for (k = k; k < m1+m2; k++)
         if (i1 == m1 ) { S1_2[ k ] = S2[ i2 ]; i2++; }
            else   { S1_2[ k ] = S1[ i1 ]; i1++; }
}
```