A tömbök deklarálásakor Pascal és C/C++ nyelvekben minden esetben meg kell adni az indexelést (Pascal) vagy az elemszámot (C/C++). Ritkán dolgozunk az összes elemmel, ezért általában van egy változónk az aktuális elemszámra – általában ***n*** a jelölése.
A következő algoritmusokban ***a*** lesz a tömb és ***n*** lesz az aktuális elemszám.

Alapműveletek

1.) Tömb elemeinek a beolvasása

```
Beolvas n
Minden i←1,n-re végezd el
│   Beolvas a[i]
■
```

2.) Tömb elemeinek a kiíratása az eredeti sorrendben: a tömb elemeit illik egymástól szóközzel elválasztani vagy adott számú pozícióra kell kiíratni őket

```
Minden i←1,n-re végezd el
│   Kiír a[i], ' '
■
```

3.) Tömb elemeinek a kiíratása fordított sorrendben:

```
Minden i←n,1,-1 -re végezd el
│   Kiír a[i], ' '
■
```

4.) Tömb elemi közül a legkisebb – minimumkeresés: feltételezzük, hogy a tömb első eleme a legkisebb. Megvizsgáljuk a többi elemet is, és ha az eddigi minimumnál kisebbet találunk, akkor kicseréljük.

```
min ← a[1]
Minden i←2,n-re végezd el
│   Ha a[i] < min akkor
│   │         min ← a[i]
│   ■
■
Kiír min
```

5.) Tömb elemi közül a legnagyobb – maximumkeresés: feltételezzük, hogy a tömb első eleme a legnagyobb. Megvizsgáljuk a többi elemet is, és ha az eddigi maximumnál nagyobbat találunk, akkor kicseréljük.

```
max ← a[1]
Minden i←2,n-re végezd el
│   Ha a[i] > max akkor
│   │         max ← a[i]
│   ■
■
Kiír max
```

6.) Tömb elemeinek az összege: számítsuk ki a tömb elemeinek az összegét.

```
s ← 0
Minden i←1,n-re végezd el
│       s ← s + a[i]
■
Kiír s
```

7.) Kiválogatás: írd ki a tömb elemei közül azokat, amelyek 5-re vagy nullára végződnek

```
Minden i←1,n-re végezd el
|   Ha a[i] utolsó számjegye 5 vagy 0 akkor
|   |         Kiír a[i]
|   ■
■
```

8.) Darabszám: számoljuk össze mennyi páros érték van a tömb elemei között

```
db ← 0
Minden i←1,n-re végezd el
|   Ha a[i] páros akkor
|   |         db ← db + 1
|   ■
■
Kiír db
```

9.) Lineráis keresés: keressük meg, melyik pozíción fordul elő az **x** a tömbben, vagy írjunk ki egy „Nem szerepel a tömbben” üzenetet

```
Beolvas x
p ← 0
Minden i←1,n-re végezd el
|   Ha a[i]=x akkor
|   |         p ← i
|   ■
■
Ha p=0 akkor Kiír „Nem szerepel a tömbben”
|       különben Kiír p
■
```

10.) Bináris keresés: amennyiben a tömb elemei rendezve vannak (az algoritmusban úgy tekintjük, hogy a tömb elemei növekvő sorrendben vannak) akkor a keresést fel lehet gyorsítani – abban a részben keresünk tovább, ahol érdemes, ahol az értéknek lennie kell

```
Beolvas x
p ← 0
bal ← 1
jobb ← n
Amíg bal≤jobb és p=0 végezd el
| k ← [(bal+jobb)/2]
| Ha a[k]=x akkor p ← k
| |  különben
| |    Ha x<a[k] akkor jobb ← k-1    {a baloldali részben kellene legyen}
| |    |  különben bal ← k+1         {a jobboldali részben kellene legyen}
| |    ■
| ■
■
Ha p=0 akkor Kiír „Nem szerepel a tömbben”
|       különben Kiír p
■
```

11.) Ütköző keresés: a keresett elemet elhelyezem az utolsó elem utáni pozícióba, így a keresésnél nem kell külön az elemszámra figyelni. Amennyiben itt a vége után találom meg, akkor a keresett érték nem szerepelt a tömbben.

```
Beolvas x
a[n+1] ← x
k ← 1
Amíg a[k]≠x végezd el
│ k ← k+1
■
Ha k=n+1 akkor Kiír „Nem szerepel a tömbben”
│       különben Kiír k
■
```

Hasonló a módszer akkor, ha csak a különböző értékeket kell a tömbben tárolni. Ilyenkor az elemszámot akkor kell növelni, ha a számot az utolsó utáni pozícióban – az ütközőben – találtam meg.

Tömb elemeinek a rendezése

(a leírt algoritmusok növekvő sorrendbe rendezik az elemeket)

http://www.sorting-algorithms.com/; http://www.cs.ubc.ca/~harrison/Java/sorting-demo.html

1.) Direkt csere, direkt összehasonlítás módszere – metoda selecţiei directe (egyszerű felcseréléses rendezés)
Összehasonlítunk minden lehetséges párt, ahol az elemek rossz sorrenben vannak, ott elvégezzük a cserét. (növekvő sorrend = ha kisebb pozíción nagyobb érték van, akkor ki kell cserélni)
Ideális módszer abban az esetben, ha csak adott feltételnek megfelelő elemeket kell rendezni, a többieket a helyükön kell hagyni (pl: rendezd a páros elemeket növekvő sorrendbe, a páratlanok maradjanak a helyükön).

```
Minden i←1,n-1 -re végezd el
|  Minden j←i+1,n -re végezd el
|  |  Ha a[i]>a[j] akkor
|  |  |    a[i] ↔ a[j]   {három értékadó utasítással elvégzem a cserét}
|  |  ■
|  ■
■
```

2.) Buborékos rendezés – metoda bulelor
Az első elemtől kezdődően egymás melletti elemeket hasonlítunk össze, ahol az elemek rossz sorrendben vannak, ott elvégezzük a cserét. A végigjárást az első elemtől addig isméletjük, amíg a végigjárásnál egyetlen cserét sem kell elvégezni.

```
Ismételd
|  csere ← hamis
|  Minden i ← 1, n-1 -re végezd el
|  |  Ha a[i]>a[i+1] akkor
|  |  |      a[i] ↔ a[i+1]
|  |  |      csere ← igaz
|  |  ■
|  ■
Ameddig nincs csere
```

3.) Buborékos rendezés javított változat: figyeljük melyik elem került már a helyére, az összehasonlításokkal csak eddig megyünk el.

```
vege ← n
Ismételd
|  csere ← hamis
|  Minden i ← 1, vege-1 -re végezd el
|  |  Ha a[i]>a[i+1] akkor
|  |  |      a[i] ↔ a[i+1]
|  |  |      csere ← igaz
|  |  ■
|  ■
|  vege ← vege - 1
Ameddig nincs csere
```

4.) Számolásos rendezés – sortare prin numărare
Minden i-edik elemre megszámoljuk mennyi nála szigorúan kisebb érték van a tömbben, a rendezett tömbbe ennek megfelelően helyezzük el az értékeket. Szükség van még két tömbre: `K` lesz az a tömb, ahol megszámoljuk, hogy az `i`-edik elemnél mennyi nála kisebb érték van és a `b` tömbbe már a helyükre másoljuk az elemeket

```
Minden i←1,n-re végezd el
|  K[i] ← 0
■
Minden i←1,n-1 -re végezd el
|  Minden j←i+1,n -re végezd el
|  |  Ha a[i]>a[j] akkor
|  |  |     K[i] ← K[i]+1   {találtam egy nála kisebb elemet}
|  |  |  különben
|  |  |     K[j] ← K[j]+1
|  |  ■
|  ■
■
Minden i←1,n-re végezd el
|  b[k[i]+1] ← a[i]
■
```

Más rendezési algoritmusok:
5.) Minimumkiválasztásos rendezés: megkeresem minden pozícióra a legkisebb elemet, a továbbiakban már csak a tőle jobbra levő elemekkel foglalkozom

```
Minden i ← 1, n-1 -re végezd el
|  min ← i   {megjegyzem az i-edik legkisebb elem pozícióját }
|  Minden j ← i+1, n -re végezd el
|  |  Ha a[j]<a[min] akkor
|  |  |     min ← j   {találtam egy nála kisebb elemet, de még nem cserélek}
|  |  ■
|  ■
|  Ha i ≠ min akkor {a legkisebb érték mégsem az i pozícióban van}
|  |  a[i] ↔ a[min]
|  ■
■
```

6.) Beszúrásos rendezés (sortare prin inserare): az új elemet egy már rendezett sorozatba a neki megfelelő pozícióba helyezem el {az egy elemből álló tömb rendezett}

```
Minden i ← 2, n -re végezd el
|  Ha a[i]<a[i-1] akkor {az aktuális érték nincs a helyén}
|  |  ment ← a[i] {félreteszem az értéket}
|  |  j ← i
|  |  Ismételd
|  |  |  j ← j-1
|  |  |  a[j+1] ← a[j] {egy pozícióval feljebb léptetem az elemeket}
|  |  Ameddig j=1 vagy a[j-1]<=ment
|  |  a[j] ← ment
|  ■
■
```

7.) Összefésülés (Merge Sort): két rendezett tömböt összefuttatok egy harmadikba. A harmadik tömmbe összeválogatom az elemeket.
Az első tömb neve a és n eleme van, a második tömb neve b és m eleme van, az eredmény tömb a c lesz k darab elemmel.

```
i ← 1;    j ← 1;   k ← 0
Amíg i ≤ n és  j ≤ m végezd el
|  Ha a[i] < b[j] akkor
|  |      k ← k+1
|  |      c[k] ← a[i]  {az első tömbből másoljuk ki az elemet}
|  |      i ← i+1      {az első tömbben lépünk tovább}
|  |  különben
|  |      k ← k+1
|  |      c[k] ← b[j]  {a második tömbből másoljuk ki az elemet}
|  |      j ← j+1      {a második tömbben lépünk tovább}
|  ■
■

Ha i ≤ n akkor  {az első tömbben maradtak még  elemek, nincs mivel összehasonlítani}
|     Minden j ← i, n -re végezd el
|     |   k ← k+1
|     |   c[k] ← a[j]
|     ■
| különben
|     Minden i ← j, m -re végezd el
|     |   k ← k+1
|     |   c[k] ← b[i]
|     ■
■
```

Kétdimenziós tömbök (mátrixok) C++ nyelvben

1.) Deklarálás: **n** sorból, **m** oszlopból álló mátrix, a mátrix sorait és oszlopait **1**-től indexelem, legtöbb 20 sora és 30 oszlopa lehet

```
int a[21][31];
int i, j, n, m;
```

2.) Mátrix elemeinek a beolvasása

```
cin>>n>>m;
for(i=1;i<=n;i++)
  for(j=1;j<=m;j++)
   {
     cout<<"a["<<i<<","<<j<<"] = ";
     cin>>a[i][j];
   }
```

3.) Mátrix elemeinek a kiírása:

```
for(i=1;i<=n;i++)
{
  for(j=1;j<=m;j++)
     cout<<a[i][j]<<" ";
  cout<<"\n";
}
```