Rendezés maximumkiválasztással

Igen gyakori feladat, hogy egy vektor elemeit növekvő (vagy csökkenő) sorba kell rendeznünk. Eddigi ismereteink alapján erre ugyan felhasználhatjuk a *ListBox* vezérlőt, amely a benne elhelyezett elemeket a *Sorted* tulajdonságának *True*-ra állításával automatikusan rendezi, de érdemesebb inkább önállóan elkészíteni egy rendező eljárást, mivel azt más nyelvek esetében is ki tudjuk használni.

Feladat: Adott az *n* elemű *v* vektor. Rendezzük növekvő sorba az elemeit!

A megoldás gondolatmenete: Válasszuk ki az *n* elemből a legnagyobbat, és cseréljük meg az utolsóval; majd válasszuk ki az első *n–1*-ből a legnagyobbat, és cseréljük meg az utolsó előttivel; majd az első *n–2*-ből válasszuk ki a legnagyobbat, és cseréljük meg az *n-2*-dikkel, stb.

Látható, hogy az első lépésben helyére kerül a legnagyobb elem, ezért a továbbiakban már nem vizsgáljuk. Ezt követően helyére kerül az utolsó előtti, s ezt a továbbiakban nem vizsgáljuk, stb.

Az eljárás lényege tehát egy külső ciklus, melyben a ciklusváltozó „hátulról halad előre", s a ciklusmagban egy maximumkiválasztás, valamint egy csere szerepel:

```
Eljárás Rendezés maximumkiválasztással
   Ciklus j=n-től 2-ig
         MaxKer (az első j elemben)
         Csere (v(j), maxh)
   Ciklus vége
Eljárás vége
```

Mivel mind a *MaxKer*, mind a *Csere* algoritmusa ismert, ezért ezeket újra beírni az eljárásba fölösleges.

Rendezés egyszerű cserével (buborékmódszer)

A buborékmódszer talán a legközismeretebb rendezési eljárás, bár a módszer viszonylag lassú, az eljárás könnyen megvalósítható.

Ha a vektort növekvő módon szeretnénk rendezni, eljárhatunk a következő módon is. Hasonlítsuk össze a vektor első és második elemét, s ha az első nagyobb, cseréljük meg őket. Ezt követően hasonlítsuk össze a vektor második és harmadik elemét, s ha a második nagyobb, cseréljük meg őket; és így tovább, amíg végig nem érünk a vektoron. Látható, hogy ekkor a vektor legnagyobb eleme biztosan a helyére kerül, még akkor is, ha ő volt az első.

Sajnos ez a többi elemre nem feltétlenül teljesül, tehát ahhoz, hogy a második legnagyobb elem a helyére kerüljön, ismét el kell indulnunk a vektor elejéről, és az első elemet össze kell hasonlítani a másodikkal, a másodikat a harmadikkal, …, azonban most elég elmenni az utolsó előtti elemig, hiszen az utolsó már a helyére került.

A következő körben már a harmadik legnagyobb elemet tesszük a helyére, és így tovább; a ciklust tehát lényegében annyiszor kell lefuttatnunk, ahány eleme van a vektorunknak.

```
Eljárás Buborékrendezés
   Ciklus i = n-től 2-ig, -1-esével
         Ciklus j = 1-tő i-1-ig
               Ha v(j)>v(j+1) akkor Csere (v(j), v(j+1))
         Ciklus vége
   Ciklus vége
Eljárás vége
```

Feladatok

1. Eladási adatok
A 12 elemű *v* vektorban egy bolt havi bevételeit tároljuk. A következő adatokat szeretnénk meghatározni: éves összbevétel, átlagos havi bevétel, volt-e olyan hónap, amikor nem volt bevétel. Készítsük el a megfelelő algoritmusokat!

2. Eldöntés és kiválasztás
Hogyan lehetne a lineáris keresés algoritmusát egyszerűsíteni, ...
a) ha csak azt szeretnénk meghatározni, hogy van-e a *v* vektornak adott tulajdonságú eleme (eldöntés)?
b) ha tudjuk, hogy van a *v* vektornak adott tulajdonságú eleme és az első ilyen indexét szeretnénk meghatározni (kiválasztás)?

3. Kiválogatás
Készítsünk algoritmust, amely a *v* vektor összes adott tulajdonságú elemét átmásolja az *u* vektorba!

4. Rekurzív rendezés
Készítsük el a buborékmódszer rekurzív változatát!

5. Logaritmikus keresés
A *v* vektorban lévő elemek nagyság szerint növekvő sorba vannak rendezve. Keressük meg, hogy szerepel-e benne egy adott elem! (Segítség: A korábban megírt számkitaláló esetében hogyan tudtunk olyan eljárást készíteni, ami a lehető legkevesebb lépésben kitalálja a gép által gondolt számot?)

6. Feltételes maximumkeresés
A *v* vektor egész számokat tartalmaz. Keressük meg a páros számok közül a legnagyobbat! Készítsük el a megoldás leírását többféle módon is!