Egyszerű programozási tételek

Sorozatszámítás tétele

Például az X tömbben kövek súlyát tároljuk. Ha ki kellene számolni az összsúlyt, akkor az S = f(S, X(i)) helyére S = S + X(i) kell írni. Az f0 tartalmazza a kezdőértéket. Összeadásnál az f0 értéke 0, míg szorzásnál 1 (hiszen ha valamit nullával szorzunk, akkor az eredmény nulla lesz.)

```
függvény
        f: elemtip, elemtip → elemtip
változók
        N: egész                        [a tömb számossága]
        X: tömb(1..N: elemtip)          [maga a tömb, amely elemtip típusú elemeket tartalmaz]
        f0: elemtip                     [kezdőérték]
        S: elemtip                      [eredmény]

Sorozatszámítás(N, X, S)
        S = f0
        ciklus i = 1-től N-ig
            S = f(S, X(i))
        ciklus vége
eljárás vége
```

Eldöntés tétele

Nem tudjuk, hogy egy adott T tulajdonságú elem létezik vagy sem a tömbben. Az itt leírt algoritmus pontosan azt adja vissza eredményül a Van változóban. A Van értéke igaz, ha a hasonlításoknál nem jutunk az N + 1-dik elemhez.

```
függvény
        T:    elemtip → logikai

változók
        N:    egész
        X:    tömb(1..N: elemtip)
        Van: logikai                    [értéke igaz, ha van ilyen elem]

Eldöntés(N, X, Van)
        i = 1
        ciklus amíg (i ≤ N) és nem T(X(i))
            i = i + 1
        ciklus vége
        Van = (i ≤ N)
eljárás vége
```

Kiválasztás tétele

Biztosra tudjuk, hogy az X tömb tartalmazza a T tulajdonságú elemet, csak azt nem tudjuk, hogy hányadik az. Ez az algoritmus megkeresi nekünk, és eredményül az elem sorszámával tér vissza.

függvény
    T: elemtip → logikai

változók
    N: egész
    X: tömb(1..N: elemtip)
    Sorsz: egész

specifikáció
    Á: $N \in N_0$, $Sorsz \in N$ [N, Sorsz eleme a pozitív egész számok halmazának]
    EF: $\exists i(1 \le i \le n)$: $T(X_i)$ [Létezik olyan i, amelyre igaz az, hogy $T(X_i)$]
    UF: $1 \le Sorsz \le N$ és $T(X_{Sorsz})$ [A Sorsz 1 és N közötti szám. Amelyre igaz $T(X_{Sorsz})$]

```
Kiválasztás(N, X, Sorsz)
    i = i
    ciklus amíg (i ≤ N) és nem T(X(i))
        i = i + 1
    ciklus vége
    Sorsz = i
eljárás vége
```

Lineáris keresés tétele

Nem tudjuk, hogy az X tömbnek van-e T tulajdonságú elem, de ha van akkor a lenti algoritmus eredményül a megtalált elem sorszámát is megadja.

függvény
    T: elemtip → logikai

változók
    N: egész
    X: tömb(1..N: elemtip)
    Van logikai
    Sorsz egész

specifikáció
    Á: $N \in N_0$, $X \in H^N$, Van $\in$ |L, $Sorsz \in N$ [X, N elemű halmaz, Van eleme a logikai halmaznak]
    EF: nincs
    UF: $Van \equiv (\exists i(1 \le i \le N)$: T $(X_i))$, $Van \Rightarrow$ ($i \le Sorsz \le N$ és $T(X_{Sorsz})$) [≡ azonos, ∃ létezik, ⇒ ha Van]

```
Keresés(N, X, Van, Sorsz)
    i = 1
    ciklus amíg (i ≤ N) és nem T(X(i))
        i = i + 1
    ciklus vége
    Van = (i ≤ N)
    ha Van akkor Sorsz = i
eljárás vége
```

Megszámolás tétele

Az algoritmus X tömb T tulajdonságú elemeit számolja meg.

```
függvény
    T:   elemtip → logikai

változók
    N:   egész
    X:   tömb(1..N: elemtip)
    DB:  egész

Megszámolás(N, X, DB)
    DB = 0
    ciklus i = 1-től N-ig
        ha T(X(i)) akkor DB = DB + 1
    ciklus vége
eljárás vége
```

Maximumkiválasztás tétele

Ezzel az algoritmussal egy tetszőleges X tömb elemei közül a legnagyobb értékűt tudjuk kiválasztani. Abban az esetben, ha nem csak egy legnagyobb elem van az X tömbben, ez az algoritmus az első ilyen elem sorszámával tér vissza. Fontos megjegyezni, hogy a a Max változó értéke nem az X tömbben található legnagyobb elem, hanem annak a sorszáma. Az algoritmus felírható úgy is, hogy a Max ne a legnagyobb értékű elem sorszámát tartalmazza, hanem a legnagyobb értéket. Az sorszámos megoldás a jobb.

```
változók
    N:    egész
    X:    tömb(1..N: elemtip)
    Max:  egész                    [a legnagyobb értékű elem indexe]

Maximumkiválasztás(N, X, Max)
    Max = 1
    ciklus i = 2-től N-ig
        ha X(Max) < X(i) akkor Max = i
    ciklus vége
eljárás vége
```

Minimumkiválasztás tétele

Ugyanaz vonatkozik rá mint a maximumkiválasztásra, csak itt a legkisebb elemet keressük.

```
változók
    N:   egész
    X:   tömb(1..N: elemtip)
    Min: egész                     [a legkisebb értékű elem indexe]

Minimumkiválasztás(N, X, Min)
    Min = 1
    ciklus i = 2-től N-ig
        ha X(Min) > X(i) akkor Min = i
    ciklus vége
eljárás vége
```

Összetett programozási tételek

Másolás tétele

Egy tömb elemeit egyesével átadjuk egy függvénynek. A függvény által visszaadott értékét egy másik tömb azonos pozíciójába tároljuk el, tehát Y(i) = f(X(i)). A két tömbnek nem kell azonos típusúnak lennie. Remek példa lehet erre, hogyha az egyik tömbben különböző áruk nettó értékét tárolom, majd a másik tömbbe ezen áruk áfás árát „másolom”. Ebben az esetben a Y(i) = f(X(i)) sor helyett az Y(i) = X(i) * 1,25 sort kell írni.

```
függvény
        f:  H_elemtip → G_elemtip

változók
        N: egész
        X: tömb(1..N: H_elemtip)        [bementi értékeket tartalmazó tömb]
        Y: tömb(1..N: G_elemtip)        [kimeneti értékeket tartalmazó tömb]

Másolás (N, X, Y)
        ciklus i = 1-től N-ig
            Y(i) = f(X(i))
        ciklus vége
eljárás vége
```

Kiválogatás tétele

Ezzel az algoritmussal egy tetszőleges tömb (X) elemei közül T tulajdonságúakat kiválogathatjuk egy másik (Y) tömbbe. Fenti okból az Y tömb számossága, és típusa ugyan az mint X tömbbé. Ennek az algoritmusnak van egy módosított változata, amelyben az Y tömb nem az X tömb T tulajdonságú elemeinek értékét, hanem azon elemek sorszámát tartalmazza. Ebben az esetben az Y tömb nem elemtip típusú elemeket tárol, hanem egész számokat, illetve Y(DB) = X(i) sort kell kicserélni Y(DB) = i sorra. (Ha pl. Az X tömbből ki akarjuk válogatni a 10-nél nagyobb értékűeket, akkor a T(X(i)) helyett X(i)>10 feltételt kell írni.)

```
függvény
        T:   elemtip → logikai

változók
        N:   egész
        X:   tömb(1..N: elemtip)        [bemeneti tömb]
        DB:  egész
        Y:   tömb(1..N: elemtip)        [kiválogatott elemeket tartalmazó tömb]

Kiválogatás(N, X, DB, Y)
        DB = 0
        ciklus i = 1-től N-ig
            ha T(X(i)) akkor
                DB = DB + 1
                Y(DB) = X(i)
            elágazás vége
        ciklus vége
eljárás vége
```

Metszetképzés tétele

Ezzel az algoritmussal két tömb (X, és Y) elemei közül kiválogatjuk azokat az elemeket, amelyeket mindkét tömb tartalmaz (az eredménytömb neve Z). A metszet számossága maximálisan csak az egyik bemeneti tömb számosságával lehet egyenlő, az hogy melyik hány elemű az lényegtelen. Mindkét bemeneti többre jellemző, hogy nincs azonos elemük (az X tömb csak egymástól eltérő elemeket tartalmaz, ugyanez vonatkozik az Y tömbre is).

```
változók
    N:  egész                   [X tömb számossága]
    X:  tömb(1..N: elemtip)     [egyik bemeneti tömb]
    M:  egész                   [Y tömb számossága]
    Y:  tömb(1..M: elemtip)     [másik bemeneti tömb]
    DB: egész                   [ez tárolja a metszet tömb (Z) számosságát]
    Z:  tömb(1..N: elemtip)     [a kimeneti tömb, ez tartalmazza az eredményt]
Metszet(N, X, M, Y, DB, Z)
    DB = 0
    ciklus i = 1-től N-ig
        j = 1
        ciklus amíg (j ≤ M) és (X(i) ≠ Y(j))
            j = j + 1
        ciklus vége
        ha j ≤ M akkor
            DB = DB + 1
            Z(DB) = X(i)
        elágazás vége
    ciklus vége
eljárás vége
```

Unióképzés tétele

Két tömb elemeit tartalmazza, de úgy, hogy azokból az elemekből, amelyek mindkét tömbben megtalál-hatók csak egy példányt tárolunk. Ebből az okból kifolyólag az eredmény tömb (Z) számossága a két bemeneti tömb számosságának összegével kell hogy egyenlő legyen hiszen a előfordulhat, hogy a két tömbnek nincs azonos értékű eleme. Mindkét bemeneti többre jellemző, hogy nincs azonos elemük (az X tömb csak egymástól eltérő elemeket tartalmaz, ugyanez vonatkozik az Y tömbre is).

```
változók
        N:  egész
        X:  tömb(1..N: elemtip)          [az egyik „halmaz”]
        M:  egész
        Y:  tömb(1..M: elemtip)          [a másik „halmaz”]
        DB: egész                        [az Z „halmaz”-ban tárolt elemek száma]
        Z:  tömb(1..N + M: elemtip)      [az eredmény „halmaz”]

Unióképzés(N, X, M, Y, DB, Z)
        ciklus i = 1-től N-ig
            Z(i) = X(i)
        ciklus vége
        DB = N
        ciklus j = 1-től M-ig
            i = 1
            ciklus amíg (i ≤ N) és (X(i) ≠ Y(j))
                i = i + 1
            ciklus vége
            ha i > N akkor
                DB = DB + 1
                Z(DB) = Y(j)
            elágazás vége
        ciklus vége
eljárás vége
```

Összefuttatás tétele

Ahhoz, hogy az algoritmus működjön egyik fontos előfeltétel az, hogy X és Y tömbök nagyság szerint növekvően rendezett elemeket tartalmazzanak (Ha csökkenően vannak rendezve az elemek a tömbben, akkor az “X(i) < Y(j) esetén ...” sorban a < jel helyett > jelet kell írni.). A két bemeneti tömbnek lehetnek azonos elemei. A Z tömb számossága itt is a két bementi tömb számosságának az összegével egyenlő. Azonos elemekből csak egyet tárolunk el.

```
változók
        N:  egész
        X:  tömb(1..N: elemtip)          [az egyik „halmaz”]
        M:  egész
        Y:  tömb(1..M: elemtip)          [a másik „halmaz”]
        DB: egész                        [az Z „halmaz”-ban tárolt elemek száma]
        Z:  tömb(1..N + M: elemtip)      [az eredmény „halmaz”]
Összefuttatás(N, X, M, Y, DB, Z)
        i = 1
        j = 1
        DB = 0
        ciklus amíg (i ≤ N) és (j ≤ M)
            DB = DB + 1
            elágazás
                X(i) < Y(j) esetén Z(DB) = X(i); i = i + 1
                X(i) = Y(j) esetén Z(DB) = X(i); i = i + 1; j = j + 1
                egyéb esetben Z(DB) = Y(j); j = j + 1
            elágazás vége
        ciklus vége
        ciklus amíg i ≤ N
            DB = DB + 1
            Z(DB) = X(i)
            i = i + 1
        ciklus vége
        ciklus amíg j ≤ M
            DB = DB + 1
            Z(DB) = Y(j)
            j = j + 1
        ciklus vége
eljárás vége
```

Összefésülés tétele

Az összefuttatás tételéhez hasonlóan az algoritmus működésének egyik előfeltétele az, hogy a két bemeneti tömb rendezett legyen, de itt nem lehet a két tömbnek (X és Y) azonos értékű eleme. A tömböket N + 1, illetve M + 1 számossággal vesszük fel. Az X tömb N + 1, illetve az Y tömb M + 1 elemei azonosak, értékük $+\infty$ A $+\infty$ egy végtelen nagy (azaz a programtól függ) érték, olyan értéket jelent, amelyet egyik tömb sem tartalmazza, illetve mindkét tömbben a legnagyobb értéket képviseli.

```
változók
        N:   egész
        X:   tömb(1..N + 1: elemtip)
        M:   egész
        Y:   tömb(1..M + 1: elemtip)
        DB:  egész
        Z:   tömb(1..N + M: elemtip)
Összefésülés(N, X, M, Y, DB, Z)
        i = 1
        j = 1
        DB = 0
        X(N + 1) = +∞
        Y(M + 1) = +∞
        ciklus amíg (i ≤ N + 1) vagy (j ≤ M + 1)
            DB = DB + 1
            ha X(i) < Y(j) akkor
                Z(DB) = X(i)
                i = i + 1
            különben
                Z(DB) = Y(j)
                j = j + 1
            elágazás vége
        ciklus vége
eljárás vége
```

Bináris keresés

Adatok gyors megkeresésére találtak ki ezt az algoritmust. Ahhoz, hogy ez az algoritmus megfelelő eredménnyel szolgáljon a bemeneti tömbnek (X) nagyság szerint növekvően rendezettnek kell hogy legyen. Ha van azonos elem, akkor az első olyat fogja megtalálni. Ha megtaláltuk a keresett elemet, akkor annak sorszámát a Sorsz fogja tartalmazni. Az algoritmus felezéssel működik, azaz a tömböt mindig két részre bontja. Az elso és az utolso változók tárolják az X tömb azon tartományát, amelyben tovább keresünk. Ha a keresett elem kisebb mint a középső elem, akkor ez a tartomány [elso..kozepso – 1], míg ha nagyobb akkor [kozepso + 1..utolso] lesz. Ha az X tömb kozepso indexű elem értéke azonos a keresett értékkel, akkor megtaláltuk azt. Ha az utolso ≤ elso, akkor nincs ilyen értékű elem a tömbben (Ábra 1).

```
változó
        N:          egész
        X:          tömb(1..N: elemtip)
        Keresett:   elemtip                 [a keresett elem]
        Van:        logikai
        Sorsz:      egész

Bináris_keresés(N, X, Keresett, Van, Sorsz)
        elso = 1
        utolso = N
        Van = hamis
        ciklus amíg nem Van és (elso ≤ utolso)
            kozepso = (elso + utolso) div 2
            elágazás
                X(kozepso) = Keresett esetén van = igaz
                X(kozepso) < Keresett esetén utolso = kozepso - 1
                egyéb esetben elso = kozepso + 1
            elágazás vége
        ciklus vége
        ha Van akkor Sorsz = kozepso
eljárás vége
```

Rendezések

A programok készítésekor az egyik leggyakrabban előforduló feladat az adatok valamilyen szempont szerinti rendezése. Sok algoritmus készült már ezen feladat ellátására, valamelyik hatékonyabb, valamelyik kevesebb memóriát emészt el. Ezen tételek úgy vannak megírva, hogy a bementi tömböt rendezi, és ugyanabba a tömbbe teszi vissza a rendezett sorozatot. Vannak olyan esetek, amikor szükségünk van az eredeti sorrendre is, ekkor rendezés előtt egy másik tömbbe kell másolni az X tömb elemit.

Egyszerű cserés rendezés

A legegyszerűbb rendezésre használt algoritmus, működés elvéből adódóan a leglassabb, de kevés adat rendezésére jó (Ábra 1). Alapgondolat: Hasonlítsuk össze az első elemet a sorozat összes többi mögötte levő elemével, s ha valamelyik kisebb nála, akkor cseréljük meg azzal. Így elérjük, hogy a sorozat első helyére a legkisebb elem kerül. Folytassuk ugyanezen elven a sorozat második elemével, utoljára az utolsó előttivel.

```
változók
        N: egész
        X: tömb(1..N: elemtip)

Egyszerű_cserés_rendezés(N, X)
        ciklus i = 1-től N – 1-ig
            ciklus j = i + 1-től N-ig
                ha X(i) > X(j) akkor
                    seged = X(i)
                    X(i) = X(j)
                    X(j) = seged
                elágazás vége
            ciklus vége
        ciklus vége
eljárás vége
```

Helyfoglalás elemszámban: $N+1$

Hasonlítások száma: $\frac{N*(N-1)}{2}$

Mozgatások száma: $0 - \frac{3*N*(N-1)}{2}$

A	D	F	B	E	C	G	i = 1, j = 2, X(i) < X(j)
A	D	F	B	E	C	G	i = 1, j = 3, X(i) < X(j)
A	D	F	B	E	C	G	i = 1, j = 4, X(i) < X(j)
A	D	F	B	E	C	G	i = 1, j = 5, X(i) < X(j)
A	D	F	B	E	C	G	i = 1, j = 6, X(i) < X(j)
A	D	F	B	E	C	G	i = 1, j = 7, X(i) < X(j)
A	D	F	B	E	C	G	i = 2, j = 3, X(i) < X(j)
A	**D**	F	**B**	E	C	G	i = 2, j = 4, X(i) > X(j)
A	B	F	D	E	C	G	i = 2, j = 5, X(i) < X(j)
A	B	F	D	E	C	G	i = 2, j = 6, X(i) < X(j)
A	B	F	D	E	C	G	i = 2, j = 7, X(i) < X(j)
A	B	**F**	**D**	E	C	G	i = 3, j = 4, X(i) > X(j)
A	B	D	F	E	C	G	i = 3, j = 5, X(i) < X(j)
A	B	**D**	F	E	**C**	G	i = 3, j = 6, X(i) > X(j)
A	B	C	F	E	D	G	i = 3, j = 7, X(i) < X(j)
A	B	C	**F**	**E**	D	G	i = 4, j = 5, X(i) > X(j)
A	B	C	**E**	F	**D**	G	i = 4, j = 6, X(i) > X(j)
A	B	C	D	F	E	G	i = 4, j = 7, X(i) < X(j)
A	B	C	D	**F**	**E**	G	i = 5, j = 6, X(i) > X(j)
A	B	C	D	E	F	G	i = 5, j = 7, X(i) < X(j)
A	B	C	D	E	F	G	i = 6, j = 7, X(i) < X(j)

Ábra 1

A csere lépésről lépésre (i = 2, j = 4):

S = X(i)	A	**D**	F	B	E	C	G	**D**
X(i) = X(j)	A	**B**	F	**B**	E	C	G	D
X(j) = S	A	B	F	**D**	E	C	G	**D**

Ábra 2

Minimumkiválasztásos rendezés

Ez az algoritmus mindig megkeresi az X tömb i.-edik helyére az i.-edik elem utáni elemek közül a legkisebb értékű elemet. Ha megtalálta, akkor azt az X tömb i.-edik helyére eltárolja, és az X tömb következő helyére keresi a legkisebbet (Ábra 3).

```
változók
        N: egész
        X: tömb(1..N: elemtip)

Minimumkiválasztásos_rendezés(N, X)
        ciklus i = 1-től N–1-ig
            min = i
            ciklus j = i+1-től N-ig
                ha X(min) > X(j) akkor min = j
            ciklus vége
            seged = X(i)
            X(i) = X(min)
            X(min) = seged
        ciklus vége
eljárás vége
```

Helyfoglalás elemszámban: $N+1$

Hasonlítások száma: $\frac{N*(N-1)}{2}$

Mozgatások száma: $\frac{N*(N-1)}{2}$

A	D	F	B	E	C	G	i = 1, j = 2, min = 1
A	D	F	B	E	C	G	i = 1, j = 3, min = 1
A	D	F	B	E	C	G	i = 1, j = 4, min = 1
A	D	F	B	E	C	G	i = 1, j = 5, min = 1
A	D	F	B	E	C	G	i = 1, j = 6, min = 1
A	D	F	B	E	C	G	i = 1, j = 7, min = 1
A	D	F	B	E	C	G	i = 2, j = 3, min = 2
A	D	F	B	E	C	G	i = 2, j = 4, min = 4
A	D	F	B	E	C	G	i = 2, j = 5, min = 4
A	D	F	B	E	C	G	i = 2, j = 6, min = 4
A	**D**	F	**B**	E	C	G	i = 2, j = 7, min = 4
A	B	F	D	E	C	G	i = 3, j = 4, min = 4
A	B	F	D	E	C	G	i = 3, j = 5, min = 4
A	B	F	D	E	C	G	i = 3, j = 6, min = 6
A	B	**F**	D	E	**C**	G	i = 3, j = 7, min = 6
A	B	C	D	E	F	G	i = 4, j = 5, min = 4
A	B	C	D	E	F	G	i = 4, j = 6, min = 4
A	B	C	D	E	F	G	i = 4, j = 7, min = 4
A	B	C	D	E	F	G	i = 5, j = 6, min = 5
A	B	C	D	E	F	G	i = 5, j = 7, min = 5
A	B	C	D	E	F	G	i = 6, j = 7, min = 6

Ábra 3

Buborékos rendezés

Alapgondolata: Hasonlítsuk egymással a szomszédos elemeket, ha a sorrendjük nem jó, akkor cseréljük meg őket. Egy ciklusléptés lefutása alatt a legnagyobb elem biztosan a sorozat végére kerül, és a nagyobb értékű elemek hátrafelé, a kisebbek előre mozdulnak el.

```
változók
        N: egész
        X: tömb(1..N: elemtip)

Buborékos_rendezés(N, X)
        ciklus i = N-től 2-ig -1-es lépésközzel
            ciklus j = 1-től i-1-ig
                ha X(j) > X(j + 1) akkor
                   seged = X(j)
                   X(j) = X(j + 1)
                   X(j + 1) = seged
                elágazás vége
            ciklus vége
        ciklus vége
eljárás vége
```

Helyfoglalás elemszámban: $N+1$

Hasonlítások száma: $\frac{N*(N-1)}{2}$

Mozgatások száma: $0 - \frac{3*N*(N-1)}{2}$

A	D	F	B	E	C	G	i = 7, j = 1, X(j) < X(j + 1)
A	D	F	B	E	C	G	i = 7, j = 2, X(j) < X(j + 1)
A	D	**F**	**B**	E	C	G	i = 7, j = 3, X(j) > X(j + 1)
A	D	B	**F**	**E**	C	G	i = 7, j = 4, X(j) > X(j + 1)
A	D	B	E	**F**	**C**	G	i = 7, j = 5, X(j) > X(j + 1)
A	D	B	E	C	F	G	i = 7, j = 6, X(j) < X(j + 1)
A	D	B	E	C	F	G	i = 6, j = 1, X(j) < X(j + 1)
A	**D**	**B**	E	C	F	G	i = 6, j = 2, X(j) > X(j + 1)
A	B	D	E	C	F	G	i = 6, j = 3, X(j) < X(j + 1)
A	B	D	**E**	**C**	F	G	i = 6, j = 4, X(j) > X(j + 1)
A	B	D	C	E	F	G	i = 6, j = 5, X(j) < X(j + 1)
A	B	D	C	E	F	G	i = 5, j = 1, X(j) < X(j + 1)
A	B	D	C	E	F	G	i = 5, j = 2, X(j) < X(j + 1)
A	B	**D**	**C**	E	F	G	i = 5, j = 3, X(j) > X(j + 1)
A	B	C	D	E	F	G	i = 5, j = 4, X(j) < X(j + 1)
A	B	C	D	E	F	G	i = 4, j = 1, X(j) < X(j + 1)
A	B	C	D	E	F	G	i = 4, j = 2, X(j) < X(j + 1)
A	B	C	D	E	F	G	i = 4, j = 3, X(j) < X(j + 1)
A	B	C	D	E	F	G	i = 3, j = 1, X(j) < X(j + 1)
A	B	C	D	E	F	G	i = 3, j = 2, X(j) < X(j + 1)
A	B	C	D	E	F	G	i = 2, j = 1, X(j) < X(j + 1)

Ábra 4

Javított buborékos rendezés

Figyeljük minden menetben a legutolsó csere helyét, és a következő menetben csak addig rendezzünk.

```
változók
        N:  egész
        X:  tömb(1..N: elemtip)

Javított_buborékos_rendezés(N, X)
        i = N
        ciklus amíg i ≥ 2
            Cs = 0
            ciklus j = 1-től i-1-ig
                ha X(j) > X(j + 1) akkor
                    seged = X(j)
                    X(j) = X(j + 1)
                    X(j + 1) = seged
                    Cs = j
                elágazás vége
            ciklus vége
            i = Cs
        ciklus vége
eljárás vége
```

A	D	F	B	E	C	G	i = 7, j = 1, X(j) < X(j + 1), Cs = 0
A	D	F	B	E	C	G	i = 7, j = 2, X(j) < X(j + 1), Cs = 0
A	D	**F**	**B**	E	C	G	i = 7, j = 3, X(j) > X(j + 1), Cs = 3
A	D	B	**F**	**E**	C	G	i = 7, j = 4, X(j) > X(j + 1), Cs = 4
A	D	B	E	**F**	**C**	G	i = 7, j = 5, X(j) > X(j + 1), Cs = 5
A	D	B	E	C	F	G	i = 7, j = 6, X(j) < X(j + 1), Cs = 5
A	D	B	E	C	F	G	i = 5, j = 1, X(j) < X(j + 1), Cs = 0
A	**D**	**B**	E	C	F	G	i = 5, j = 2, X(j) > X(j + 1), Cs = 2
A	B	D	E	C	F	G	i = 5, j = 3, X(j) < X(j + 1), Cs = 2
A	B	D	**E**	**C**	F	G	i = 5, j = 4, X(j) > X(j + 1), Cs = 4
A	B	D	C	E	F	G	i = 4, j = 1, X(j) < X(j + 1), Cs = 0
A	B	D	C	E	F	G	i = 4, j = 2, X(j) < X(j + 1), Cs = 0
A	B	**D**	**C**	E	F	G	i = 4, j = 3, X(j) > X(j + 1), Cs = 3
A	B	C	D	E	F	G	i = 3, j = 1, X(j) < X(j + 1), Cs = 0
A	B	C	D	E	F	G	i = 3, j = 2, X(j) < X(j + 1), Cs = 0

Ábra 5

Helyfoglalás elemszámban: $N+1$

Hasonlítások száma: $N-1$ - $\frac{N*(N-1)}{2}$

Mozgatások száma: 0 - $\frac{3*N*(N-1)}{2}$

Beillesztéses rendezés

Alapgondolata: Egyetlen elem mindig rendezett, a következő elemet vagy elé, vagy mögé kell beilleszteni.

```
változók
        N: egész
        X: tömb(1..N: elemtip)

Beillesztéses_rendezés(N, X)
        ciklus i = 2-től N-ig
            j = i – 1
            ciklus amíg (j > 0) és (X(j) > X(j + 1))
                seged = X(j)
                X(j) = X(j + 1)
                X(j + 1) = seged
                j = j – 1
            ciklus vége
        ciklus vége
eljárás vége
```

A	D	F	B	E	C	G	i = 2, j = 1, X(j) < X(j + 1)
A	D	F	B	E	C	G	i = 3, j = 2, X(j) < X(j + 1)
A	D	**F**	**B**	E	C	G	i = 4, j = 3, X(j) > X(j + 1)
A	**D**	**B**	F	E	C	G	i = 4, j = 2, X(j) > X(j + 1)
A	B	D	F	E	C	G	i = 4, j = 1, X(j) < X(j + 1)
A	B	D	**F**	**E**	C	G	i = 5, j = 4, X(j) > X(j + 1)
A	B	D	E	F	C	G	i = 5, j = 3, X(j) < X(j + 1)
A	B	D	E	**F**	**C**	G	i = 6, j = 5, X(j) > X(j + 1)
A	B	D	**E**	**C**	F	G	i = 6, j = 4, X(j) > X(j + 1)
A	B	**D**	**C**	E	F	G	i = 6, j = 3, X(j) > X(j + 1)
A	B	C	D	E	F	G	i = 6, j = 2, X(j) < X(j + 1)
A	B	C	D	E	F	G	i = 7, j = 6, X(j) < X(j + 1)

Ábra 6

Helyfoglalás elemszámban: $N+1$

Hasonlítások száma: $N-1$ - $\frac{N*(N-1)}{2}$

Mozgatások száma: 0 - $\frac{3*N*(N-1)}{2}$

Javított beillesztés rendezés

Alapgondolata: Az előző módszernél a beillesztendő elemet sokszor kell mozgatni. Ezt a mozgatást le lehet csökkenteni egyre, ha a beillesztendőt nem tesszük azonnal be a sorozatba, hanem a többieket tologatjuk hátra és a beillesztendőt csak a végén tesszük a helyére.

```
változók
        N:  egész
        X:  tömb(1..N: elemtip)

Javított_beillesztéses_rendezés(N, X)
        ciklus i = 2-től N-ig
            j = i – 1
            Y = X(i)
            ciklus amíg (j > 0) és (X(j) > Y)
                X(j + 1) = X(j)
                j = j – 1
            ciklus vége
            X(j + 1) = Y
        ciklus vége
eljárás vége
```

A	D	F	B	E	C	G	i = 2, j = 1, Y = “D”, X(j) < Y
A	D	F	B	E	C	G	i = 3, j = 2, Y = “F”, X(j) < Y
A	D		**F**	E	C	G	i = 4, j = 3, Y = “B”, X(j) > Y
A		**D**	F	E	C	G	i = 4, j = 2, Y = “B”, X(j) > Y
A	**B**	D	F	E	C	G	i = 4, j = 1, Y = “B”, X(j) < Y
A	B	D		**F**	C	G	i = 5, j = 4, Y = “E”, X(j) > Y
A	B	D	**E**	F	C	G	i = 5, j = 3, Y = “E”, X(j) < Y
A	B	D	E		**F**	G	i = 6, j = 5, Y = “C”, X(j) > Y
A	B	D		**E**	F	G	i = 6, j = 4, Y = “C”, X(j) > Y
A	B		**D**	E	F	G	i = 6, j = 3, Y = “C”, X(j) > Y
A	B	**C**	D	E	F	G	i = 6, j = 2, Y = “C”, X(j) < Y
A	B	C	D	E	F	G	i = 7, j = 6, Y = “G”, X(j) < Y

Ábra 7

Helyfoglalás elemszámban: $N+1$

Hasonlítások száma: $N-1$ - $\frac{N*(N-1)}{2}$

Mozgatások száma: $2*(N-1)+\frac{N*(N-1)}{2}$

QuickSort

Az adatok rendezésének nagyon gyors módja. Az algoritmus rekurzív hívásokkal dolgozik. A rendezendő résztömböt két részre osztja, a középső elemnél kisebb elemek balra, míg a nagyobb elemek jobbra teszi, majd az így létrejött tömb – ha szükséges – mind jobb, mind bal résztömbjével újra elvégzi a fenti műveletet.

```
változók
        N:  egész
        X:  tömb(1..N: elemtip)
Sort(X, e, v)
        változók
                Y:  elemtip
        k = (e + v) div 2
        Y = X(k)
        i = e
        j = v
        ciklus
            ciklus amíg X(i) < Y
                i = i + 1
            ciklus vége
            ciklus amíg Y < X(j)
                j = j – 1
            ciklus vége
            ha i ≤ j akkor
                seged = X(i)
                X(i) = X(j)
                X(j) = seged
                i = i + 1
                j = j – 1
            elágazás vége
        mígnem i > j
        ha i < v akkor Sort(X, i, v)
        ha e < j akkor Sort(X, e, j)
eljárás vége
QuickSort(N, X)
        e = 1
        v = N
        Sort(X, e, v)
eljárás vége
```

A	D	F	B	E	C	G	e = 1, v = 7, Y = “B”, i = 1, j = 7
A	**D**	F	**B**	E	C	G	e = 1, v = 7, Y = “B”, i = 2, j = 4
A	B	F	D	E	C	G	e = 1, v = 7, Y = “B”, i = 3, j = 2
		F	D	E	C	G	e = 3, v = 7, Y = “E”, i = 3, j = 7
		F	D	E	**C**	G	e = 3, v = 7, Y = “E”, i = 3, j = 6
		C	D	**E**	F	G	e = 3, v = 7, Y = “E”, i = 5, j = 5
		C	D	E	F	G	e = 3, v = 7, Y = “E”, i = 6, j = 4
					F	G	e = 6, v = 7, Y = “F”, i = 6, j = 7
					F	G	e = 6, v = 7, Y = “F”, i = 6, j = 6
					F	G	e = 6, v = 7, Y = “F”, i = 7, j = 5
		C	D				e = 3, v = 4, Y = “C”, i = 3, j = 4
		C	D				e = 3, v = 4, Y = “C”, i = 3, j = 3
		C	D				e = 3, v = 4, Y = “C”, i = 4, j = 2
A	B						e = 1, v = 2, Y = “A”, i = 1, j = 2
A	B						e = 1, v = 2, Y = “A”, i = 1, j = 1
A	B						e = 1, v = 2, Y = “A”, i = 2, j = 0

Ábra 8

Leszámláló rendezés

Csak egész számok rendezésére alkalmas algoritmusról van szó, nagy hátránya a magas memória szükséglet. Előnye a gyorsasága.

```
változók
        N: egész                        [a rendezendő és a rendezett elemeinek számossága]
        X: tömb(1..N: egész)            [rendezendő tömb]
        Y: tömb(1..N: égész)            [rendezett tömb]
        S: tömb(1..Max(X): egész)       [rendezéshez használt segéd, a Max(X) a rendezendő tömbben
                                         tárolt elemek közül a legnagyobb (a típushoz tartozó várható
                                         legnagyobb érték)]

Leszámláló_rendezés(N, X, Y)
        k = 1
        ciklus i = 2-től N-ig
            ha X(i) > X(k) akkor k = i
        ciklus vége
        k = X(k)
        ciklus i = 1-től k-ig
            S(i) = 0
        ciklus vége
        ciklus i = 1-től N-ig
            S(X(i)) = S(X(i)) + 1
        ciklus vége
        ciklus i = 2-től k-ig
            S(i) = S(i) + S(i - 1)
        ciklus vége
        ciklus i = N-től 1-ig -1-esével
            Y(S(X(i))) = X(i)
            S(X(i)) = S(X(i)) – 1
        ciklus vége
eljárás vége
```