```
program Pelda28;
uses crt;
const n=20;
var a: array[1..n] of integer;
    i,x: integer;
begin
clrscr;
 randomize;
 for i:=1 to n do a[i]:=random(90)+10;
 {a tömb elemeinek kiírása...}
 for i:=1 to n do write(a[i]:3);
 writeln;
 {a tömb tükrözése...}
 for i:=1 to n div 2 do begin
                        x:=a[i];
                        a[i]:=a[n-i+1];
                        a[n-i+1]:=x;
                        end;
 {a tömb elemeinek kiírása...}
 for i:=1 to n do write(a[i]:3);
 readln;
end.
```

11.4 Tömbök rendezése

Egy N elemű sorozatot nagyság szerint sorba kell rendezni. Nagyon sok rendezési algoritmus létezik. Az alábbiakban ezek közül kettőt fogunk megvizsgálni. Közös vonásuk az lesz, hogy az eredmény, a rendezett sorozat, helyben keletkezik, így az eredeti sorrend elvész. Az a rendezési algoritmus jó, amelynek kicsi a tárigénye és nagyon gyorsan végrehajtja a rendezést, és persze egyszerű, könnyen megérthető a működése.

Tökéletes, minden igényt kielégítő rendezési eljárás nem létezik. Az, hogy mikor melyik rendező algoritmust használjuk, sok összetevő függvénye. Tisztázni kell egy-egy konkrét eljárás kiválasztásakor, hogy az algoritmus elsősorban gyors legyen vagy inkább helytakarékos. Az alábbiakban tárgyalt algoritmusokbanaz N=5 elemű A() vektort fogjuk növekvő sorrendbe rendezni.

	1.	2.	3.	4.	5.
A()	**5**	**3**	**9**	**1**	**7**

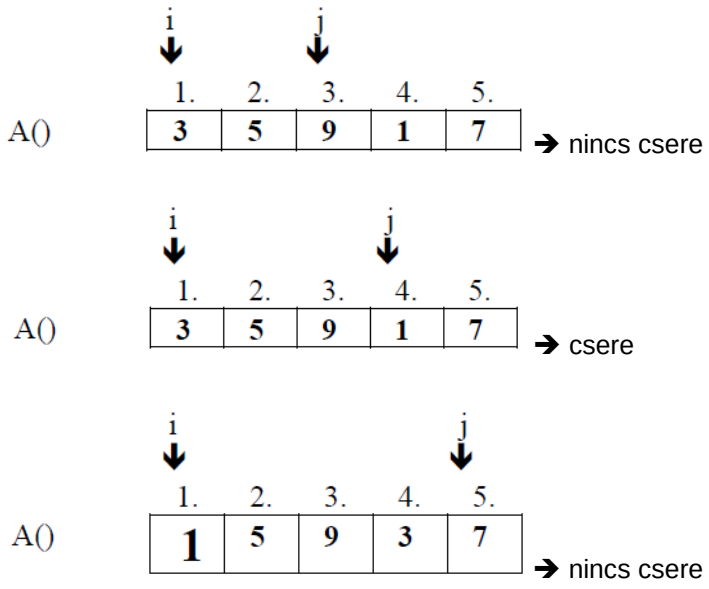
rendezve

	1.	2.	3.	4.	5.
A()	**1**	**3**	**5**	**7**	**9**

11.4.1 Egyszerű cserés rendezés

Hasonlítsuk össze a() tömb első elemét a sorozat összes többi mögötte lévő elemével, s ha valamelyik kisebb nála, akkor cseréljük meg azzal! Ezzel elérhetjük, hogy a sorozat első helyére a legkisebb elem kerül. Folytassuk ugyanezen elven a sorozat második elemével, utoljára pedig az utolsó előttivel.

	i ↓	j ↓				
	1.	2.	3.	4.	5.	
A()	**5**	**3**	**9**	**1**	**7**	➔ csere



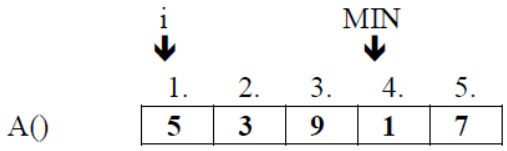
A **j** -vel végigértünk a sorozaton, és az 1. helyre megtaláltuk a sorozat legkisebb elemét. Most **a(1)** biztosan a legkisebb elem. Az **i** -t növeljük eggyel és a 2. helyre keressük meg, a **j** segítségével, **a(2)**, **a(3)**, **a(4)**, **a(5)** közül a legkisebbet. Ez persze biztos nem lesz kisebb **a(1)** - nél, különben ő került volna az 1. helyre. A fentiekben leírt gondolatmenetet követjük, amíg **i** -vel végig nem érünk a sorozaton, azaz az **n-1.** helyre meg nem találtuk **a(n-1)** és **a(n)** közül a kisebbet. Az **n.** helyre pedig nyilvánvalóan a sorozat legnagyobb eleme marad.

```
program cserrend;
uses crt;
const n=5;
var a:array[1..n] of integer;
    i,j,x:integer;
begin
 clrscr;
 randomize;
 i:=0;
 j:=0;
 x:=0;
   write('5 db 1 es 9 kozotti szam generalasahoz nyomj ENTER-t!');
   readln;
   for i:=1 to n do
    begin
     a[i]:=random(10)+1;
     write(a[i],', ')
    end;
   for i:=1 to n-1 do
    for j:=i+1 to n do
     if a[i]>a[j] then
      begin
       x:=a[i];
       a[i]:=a[j];
       a[j]:=x;
      end;
   writeln;
   for i:=1 to n do write(a[i],', ');
 readln;
end.
```

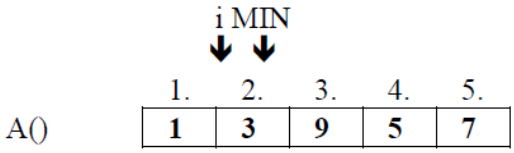
11.4.2 Minimumkiválasztásos rendezés

Az előző módszer hátránya a sok felesleges csere. Célszerűbb lenne az aktuális, **i.** , elemet a mögötte lévők közül egyedül a legkisebbel felcserélni. Ez a felismerés vezet a módszer javításához, a minimumkiválasztásos rendezéshez.

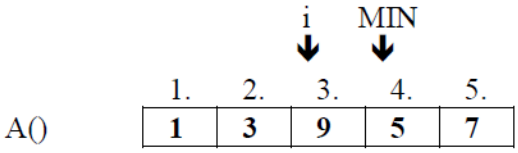
Az **a()** tömb 1. helyére keressük ki az **a(1), a(2), a(3), a(4), a(5)** elemek közül a legkisebbet (minimumkiválasztás)! Ha megtaláltuk a legkisebb elemet, akkor cseréljük ki az **a(1)** - gyel!



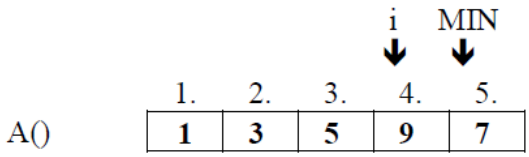
Ekkor az **a(1)** biztosan a vektor legkisebb eleme. Növeljük az **i** -t 1 -gyel, így az **a()** tömb 2. helyére keressük ki **a(2), a(3), a(4), a(5)** közül a legkisebbet!



Most **a(i)** -t cseréljük fel a(min) -nel! (Ez a csere tulajdonképpen felesleges, hiszen most **a(2)** - t cseréljük fel **a(2)** -vel! növeljük **i** -t 1 -gyel!



Most **a(i)** -t cseréljük fel **a(min)** -nel! majd növeljük **i** -t 1 -gyel!



Most **a(i)** -t cseréljük fel **a(min)** -nel! az **i** -vel elértük **n-1** -t, ezután az **a()** biztosan rendezett!

A módszer hátránya, hogy bizonyos esetekben (ha i=min) egy elemet önmagával cserélünk fel. Ez nyilvánvalóan felesleges csere.

```
program minrend;
uses crt;
const n=5;
var a:array[1..n] of integer;
    i,j,x,min:integer;
begin
 clrscr;
 randomize;
 i:=0;
 j:=0;
 x:=0;
   write('5 db 1 es 9 kozotti szam generalasahoz nyomj ENTER-t!');
   readln;
   for i:=1 to n do
    begin
     a[i]:=random(10)+1;
     write(a[i],', ')
    end;
    for i:=1 to n do
     begin
      min:=i;
      for j:=i+1 to n do
       if a[min]>a[j] then min:=j;
      x:=a[i];
      a[i]:=a[min];
      a[min]:=x;
     end;
     writeln;
   for i:=1 to n do write(a[i],', ');
  readln;
end.
```

11.5 Két rendezett tömb összefésülése

A következő példában legyen két tömbünk - **a** és **b**, melyek közül az elsőnek van **na**, a másodikak **nb** darab eleme. Mindkét tömbünk rendezett, növekvő sorrendben (ehhez a program elején a tömbök deklarálásánál megadjuk az tömbök elemeinek értékét). Készítsünk egy olyan programot, amely az **a** és **b** tömböt összefésüli egy új - **c** tömbbe, tehát veszi az **a** és **b** tömb elemeit és azokat sorban egymás után átrakja a **c** tömbbe úgy, hogy a c tömb is rendezett legyen.

Nézzük a feladat tömbjeit ábrán szemléltetve (az **a** és **b** tömbök a program elején adottak, melyek elemeiből a program hozza létre a **c** tömböt):

a[1]	a[2]	a[3]	a[4]	a[5]
8	10	12	17	21

b[1]	b[2]	b[3]	b[4]	b[5]
9	15	19	28	57

c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]	c[10]
8	9	10	12	15	17	19	21	28	57

A programban fogunk használni további két változót - **ai** és **bi**, melyek fogják jelölni, hogy az **a** ill. **b** tömb éppen melyik eleménél járunk.

Az **ai**, **bi** változókat a program elején beállítjuk 1-re, majd a programban (a **c** tömb létrehozására szolgáló ciklusban) mindig az **a** tömb **ai**-dik eleme vagy a **b** tömb **bi**-dik eleme közöl a kisebbet tesszük át a **c** tömbbe (és növeljük az **ai** vagy **bi** értékét egyel attól föggően melyik tömbből raktuk át az elemet a **c** tömbbe). A c tömbhöz is bevezetünk egy **ci** változót, mely azt fogja jelölni, hogy éppen hol járunk a c tömbben (kezdetben ennek az értéke is 1 lesz, majd minen egyes átrakásnál növeljük egyel).

A fent leírt, elemek átrakására szolgáló algoritmust addig fogjuk ismételni, amíg nem érünk az **a** vagy a **b** tömb végére (**ai** vagy **bi** nem éri el a tömb végét, tehát amíg nem lesz több, mint az adott tömb elemeinek száma - **an**, **bn**). Ehhez egy repeat..until ciklust használunk. Ez után már csak a másik tömbből (amelyiknél még nem értünk a tömb végére) átrakjuk az összes megmaradt elemet a **c** tömbbe (**ci**-től kezdve a végéig).

Végül ellenőrzésképpen kiírjuk a képernyőre az eredeti **a**, **b** és az általunk létrehozott **c** tömböt. Programunk így néz ki:

```
program Pelda29;
uses crt;
const an=5;
      bn=5;
      cn=an+bn;
var a: array[1..an] of integer = (8,10,12,17,21);
    b: array[1..bn] of integer = (9,15,19,28,57);
    c: array[1..cn] of integer;
    i,ai,bi,ci: integer;
begin
clrscr;
 ai:=1;
 bi:=1;
 ci:=1;
 repeat
  if a[ai]<b[bi] then begin
                      c[ci]:=a[ai];
                      inc(ai);
                      end
                 else begin
                      c[ci]:=b[bi];
                      inc(bi);
                      end;
  inc(ci);
 until (ai>an) or (bi>bn);
 if (ai>an) then begin
                 for i:=bi to bn do c[ci+(i-bi)]:=b[i];
                 end
            else begin
                 for i:=ai to an do c[ci+(i-ai)]:=a[i];
                 end;
 write('A tomb: ');
 for i:=1 to an do write(a[i],', ');
 writeln;
 write('B tomb: ');
 for i:=1 to bn do write(b[i],', ');
 writeln;
 write('C tomb: ');
 for i:=1 to cn do write(c[i],', ');
 readln;
end.
```

A programban gondolkozzunk el többek között a `for i:=bi to bn do c[ci+(i-bi)]:=b[i];` soron.

Itt valójában egy ciklus segítségével, amelyben az **i** ciklusváltozó értéke **bi**-től **bn**-ig megy átrakjuk a maradék elemeket a **b** tömbből a **c** tömbbe. Ha jobban megfigyeljük, akkor a `c[ci+(i-bi)]` kifejezésben az `(i-bi)` értéke először 0 (hiszen az **i** kezdeti értéke **bi**), majd 1, 2, ... Tehát a **b** tömb maradék elemeit (`b[bi]`, `b[bi+1]`, `b[bi+2]`, ...) a `c[ci]`, `c[ci+1]`, `c[ci+2]`, ... elemekbe rakjuk át, ami a valódi célunk volt.

A `for i:=ai to an do c[ci+(i-ai)]:=a[i];` sor hasonlóan működik, csak itt az **a** tömb maradék elemét rakjuk át a **c** tömbbe.