Programozás 2

Rendezési algoritmusok: egyszerű cserés rendezés, buborékos rendezés, beszúró rendezés, minimum- és maximumkiválasztásos rendezés; algoritmusok időbonyolultsága

Tantárgy teljesítésének feltételei

1. **Részvétel az előadásokon és a gyakorlatokon!**
   (SJE Tanulmányi Szabályzata: 8. cikk, 3-4.)

2. **Gyakorlati rész teljesítése:**
   - Szemeszter alatti két ZH megírása, melyeken el kell érni legalább 50% értékelést!
   - Két BEADANDÓ program leadása az e-learninges portálon keresztül a megadott határidőig, melyeken el kell érni legalább 50% értékelést!
   - A gyakorlati órákon lehetőség van BONUSZ pontok szerzésére is.

   Nincs ZH javítási lehetőség! Nincs lehetőség a programok leadására a határidők után!

3. **Vizsga:**
   - Írásbeli vizsga, amely elméleti és gyakorlati kérdéseket is tartalmaz. A vizsgán el kell érni legalább 50% értékelést!

   Legfeljebb háromszor lehet jönni vizsgázni!

Gyakorlati rész értékelése:

AIS-ban a folyamatos értékelés alatt mindenki megtekintheti mennyi százalékot ért el ZH1, ZH2, BEADANDO1 és BEADANDO2-ből. Ezekből kiszámítható a gyakorlati értékelés:

ZH = (ZH1 + ZH2) / 2 … minimum 50% szükséges!

BEADANDO = (BEADANDO1 + BEADANDO2) / 2 … minimum 50% szükséges!

GYAKORLAT = (ZH + BEADANDO) / 2 + BONUSZ

Vizsgán elért értékelés:

VIZSGA … minimum 50% szükséges!

JEGY = (GYAKORLAT + VIZSGA) / 2

Tömb (vektor) elemeinek rendezése

Feladat: **Rendezzük az egydimenziós tömb elemeit növekvő (vagy csökkenő) sorrendbe!**

Bemenet:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
2	18	6	40	19	20	26	25	24	7

Kimenet:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
2	6	7	18	19	20	24	25	26	40

Kimenet (csökkenő sorrend):

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
40	26	25	24	20	19	18	7	6	2

Hogyan tudjuk ezt elérni?

Tömbelemek összehasonlításával és cseréivel, pl.:

			a[i]		a[j]				
2	18	6	40	19	20	26	25	24	7

```
if (a[i]>a[j]) {
    int tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
}
```

			a[i]		a[j]				
2	18	6	20	19	40	26	25	24	7

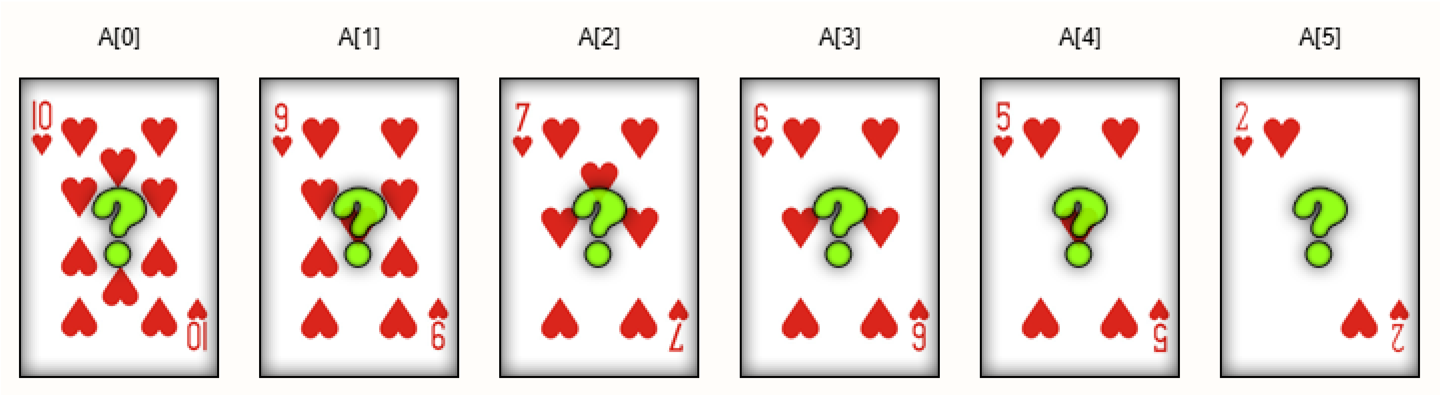
Rendezési algoritmusok

Egyszerűbb rendezési algoritmusok:

- Egyszerű cserés rendezés (Simple exchange sort)
- Buborékos rendezés (Bubblesort)
- Beszúró rendezés (Insertion sort)
- Minimumkiválasztásos rendezés (Minimum selection sort)
- Maximumkiválasztásos rendezés (Maximum selection sort)

Előnyük: Rövidebb, könnyebben megérthetőbb forráskódok.

Hátrányuk: Sok elem (pl. 50 000) rendezésére nem használhatók, mivel lassúk.



https://anim.ide.sk/kartyarendezes.php

Feladat:

Rendezzük a kártyalapokat növekvő sorrendbe!

Egyszerű cserés rendezés (Simple exchange sort)

7-elemű tömb rendezése: https://ani.ide.sk/simplesort.html

<u>Forráskód **n**-elemű **t** tömb rendezésére:</u>

```
for (int i=0; i<n-1; i++) {
    for (int j=i+1; j<n; j++) {
        if (t[i]>t[j]) {
            int tmp = t[i];
            t[i] = t[j];
            t[j] = tmp;
        }
    }
}
```

Buborékos rendezés (Bubblesort)

7-elemű tömb rendezése: https://ani.ide.sk/bubblesort.html

<u>Forráskód **n**-elemű **t** tömb rendezésére:</u>

```
for (int i=n-1; i>0; i--) {
    for (int j=0; j<i; j++) {
        if (t[j]>t[j+1]) {
            int tmp = t[j];
            t[j] = t[j+1];
            t[j+1] = tmp;
        }
    }
}
```

Továbbfejlesztett buborékos rendezés (Improved bubblesort)

7-elemű tömb rendezése: https://ani.ide.sk/bubblesort2.html

Forráskód **n**-elemű **t** tömb rendezésére:

```
int i=n-1;
while (i>0) {
    int cs=-1;
    for (int j=0; j<i; j++) {
        if (t[j]>t[j+1]) {
            int tmp = t[j];
            t[j] = t[j+1];
            t[j+1] = tmp;
            cs=j;
        }
    }
    i=cs;
}
```

Beszúró rendezés (Insertion sort)

7-elemű tömb rendezése: https://ani.ide.sk/insertsort.html

Forráskód **n**-elemű **t** tömb rendezésére:

```
for (int i=1; i<n; i++) {
    int j=i-1;
    while (j>=0 && t[j]>t[j+1]) {
        int tmp = t[j];
        t[j] = t[j+1];
        t[j+1] = tmp;
        j--;
    }
}
```

Továbbfejlesztett beszúró rendezés (Improved insertion sort)

7-elemű tömb rendezése: https://ani.ide.sk/insertsort2.html

Forráskód **n**-elemű **t** tömb rendezésére:

```
for (int i=1; i<n; i++) {
    int j=i-1;
    int tmp = t[i];
    while (j>=0 && t[j]>tmp) {
        t[j+1] = t[j];
        j--;
    }
    t[j+1] = tmp;
}
```

Minimumkiválasztásos rendezés (Minimum selection sort)

7-elemű tömb rendezése: https://ani.ide.sk/minsort.html

Forráskód **n**-elemű **t** tömb rendezésére:

```
for (int i=0; i<n-1; i++) {
    int min = i;
    for (int j=i+1; j<n; j++) {
        if (t[j] < t[min]) {
            min = j;
        }
    }
    int tmp = t[i];
    t[i] = t[min];
    t[min] = tmp;
}
```

Maximumkiválasztásos rendezés (Maximum selection sort)

7-elemű tömb rendezése: https://ani.ide.sk/maxsort.html

Forráskód **n**-elemű **t** tömb rendezésére:

```
for (int i=n-1; i>0; i--) {
    int max = 0;
    for (int j=1; j<=i; j++) {
        if (t[j] > t[max]) {
            max = j;
        }
    }
    int tmp = t[i];
    t[i] = t[max];
    t[max] = tmp;
}
```

Algoritmusok időbonyolultsága

Jelölések:

- **O (nagy O) → Legrosszabb eset (Worst-case):**
  A maximális számú lépés, amit az algoritmus végezhet.
- **Θ (Theta) → Átlagos eset (Average-case):**
  Átlagos számú lépés, amit az algoritmus általában végez.
- **Ω (Omega) → Legjobb eset (Best-case):**
  A legkevesebb lépés, amit az algoritmus végezhet.

Az eddig átvett rendezési algoritmusoknál az időbonyolultság:

- Legrosszabb eset (fordított sorrendben levő adatokon): $O(n^2)$.
- Átlagos eset (véletlenszerű adatokon): $\Theta(n^2)$.
- Legjobb eset (rendezett sorrendben levő adatokon): $\Omega(n^2)$ vagy $\Omega(n)$.

Feladat: Határozzuk meg az egyszerű cserés rendezés időbonyolultságát.

```
for (int i=0; i<n-1; i++) {
    for (int j=i+1; j<n; j++) {
        if (t[i]>t[j]) {
            int tmp = t[i];
            t[i] = t[j];
            t[j] = tmp;
        }
    }
}
```

Az algoritmus két egymásba ágyazott ciklusból áll, ezért az időbonyolultság az ismétlések számán alapul:

- **Külső ciklus:**
  Az **i** értékei **0**-tól **n−2**-ig futnak, ez összesen **n−1** iteráció.
- **Belső ciklus:**
  A **j** értékei **i+1**-től **n−1**-ig futnak, tehát az első iterációnál **n−1**, a másodiknál **n−2**, stb.

Az összehasonlítások száma:

$$\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} n-1-i = \frac{(n-1+1).(n-1)}{2} = \frac{n.(n-1)}{2} = O(n^2)$$

Vajon mennyi lesz a legrosszabb esetben, átlagos esetben és a legjobb esetben az algoritmus időbonyolultsága?

```
for (int i=0; i<n-1; i++) {
    for (int j=i+1; j<n; j++) {
        if (t[i]>t[j]) {
            int tmp = t[i];
            t[i] = t[j];
            t[j] = tmp;
        }
    }
}
```

A két ciklus mindig lefut, függetlenül attól, hogy az elemek kezdetben fordított sorrendben vannak-e, véletlenszerű értékük van-e, vagy már rendezett sorrendben vannak-e.

$$O(n^2) = \Theta(n^2) = \Omega(n^2)$$

Feladat: Az alábbi weboldalon található animáció segítségével próbáljuk ki, hogy a továbbfejlesztett buborékos rendezés hogyan működik kezdetben rendezett tömbön (az oszlopok magasságait állítsuk rendezettre az algoritmus futtatása előtt)! Vajon mennyi ennek az algoritmusnak az időbonyolultsága a legjobb esetben?

```
int i=n-1;
while (i>0) {
    int cs=-1;
    for (int j=0; j<i; j++) {
        if (t[j]>t[j+1]) {
            int tmp = t[j];
            t[j] = t[j+1];
            t[j+1] = tmp;
            cs=j;
        }
    }
    i=cs;
}
```

https://ani.ide.sk/bubblesort2.html

Legrosszabb eset (az elemek kezdetben fordított sorrendben vannak): $O(n^2)$

Átlagos eset (az elemek kezdetben véletlenszerűek):

$$\Theta(n^2)$$

Legjobb eset (az elemek kezdetben már rendezettek):

$$\Omega(n)$$

Időbonyolultságok összehasonlítása: