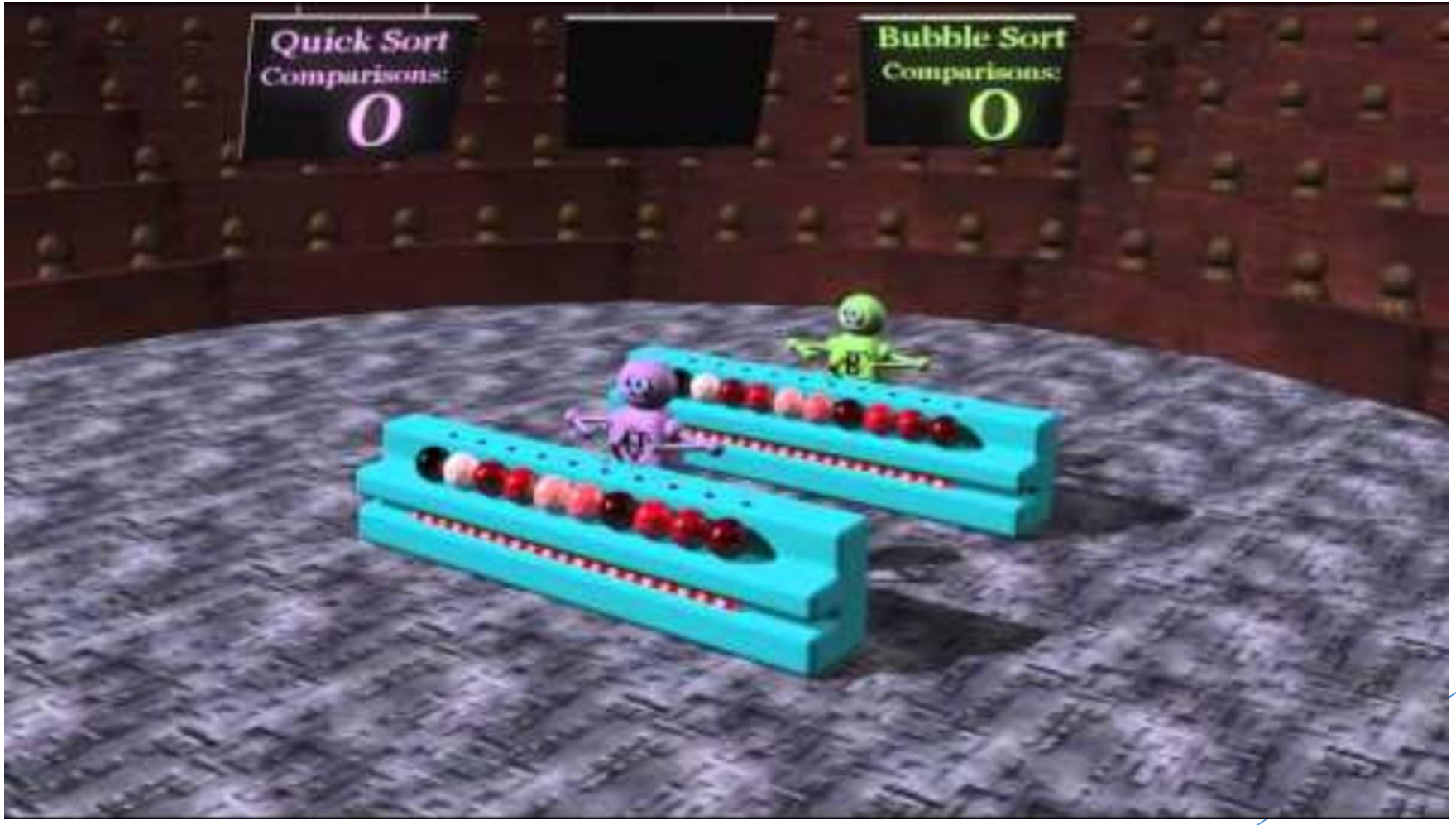
Programozás 2

Gyorsrendezés (quicksort); összefésülő rendezés (mergesort)

Gyorsrendezés (quicksort)

Hatékony és széles körben alkalmazott rendezési algoritmus, amely a **"oszd meg és uralkodj" (divide and conquer)** elv alapján működik.

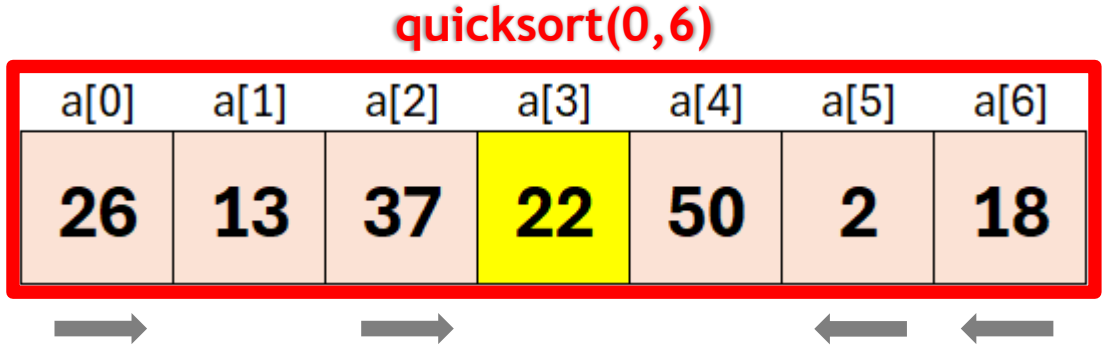


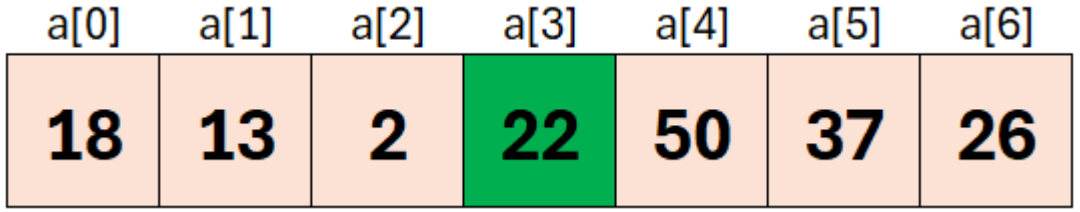
https://www.youtube.com/watch?v=aXXWXz5rF64

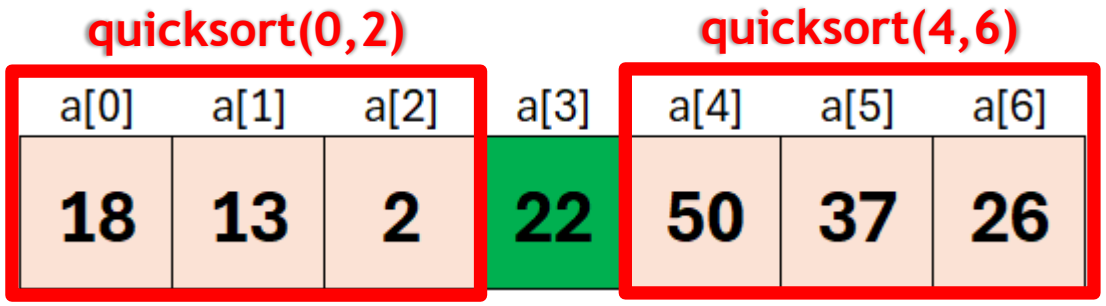
A quicksort rekurzív függvény működése:

- **Kiválasztunk egy vezérelemet (pivot) a tömbből.**
- **A tömbelemeket átrendezzük:**
  - A tömb elejére rakjuk vezérelemnél kisebb vagy egyenlő elemeket.
  - A tömb végére rakjuk a vezérelemnél nagyobb elemeket.

  Ezzel a tömböt felosztottuk két részre.
- **Mindkét résztömbre újra meghívjuk a quicksort rekurzív függvényt (amennyiben a résztömbben több mint egy elem van).**







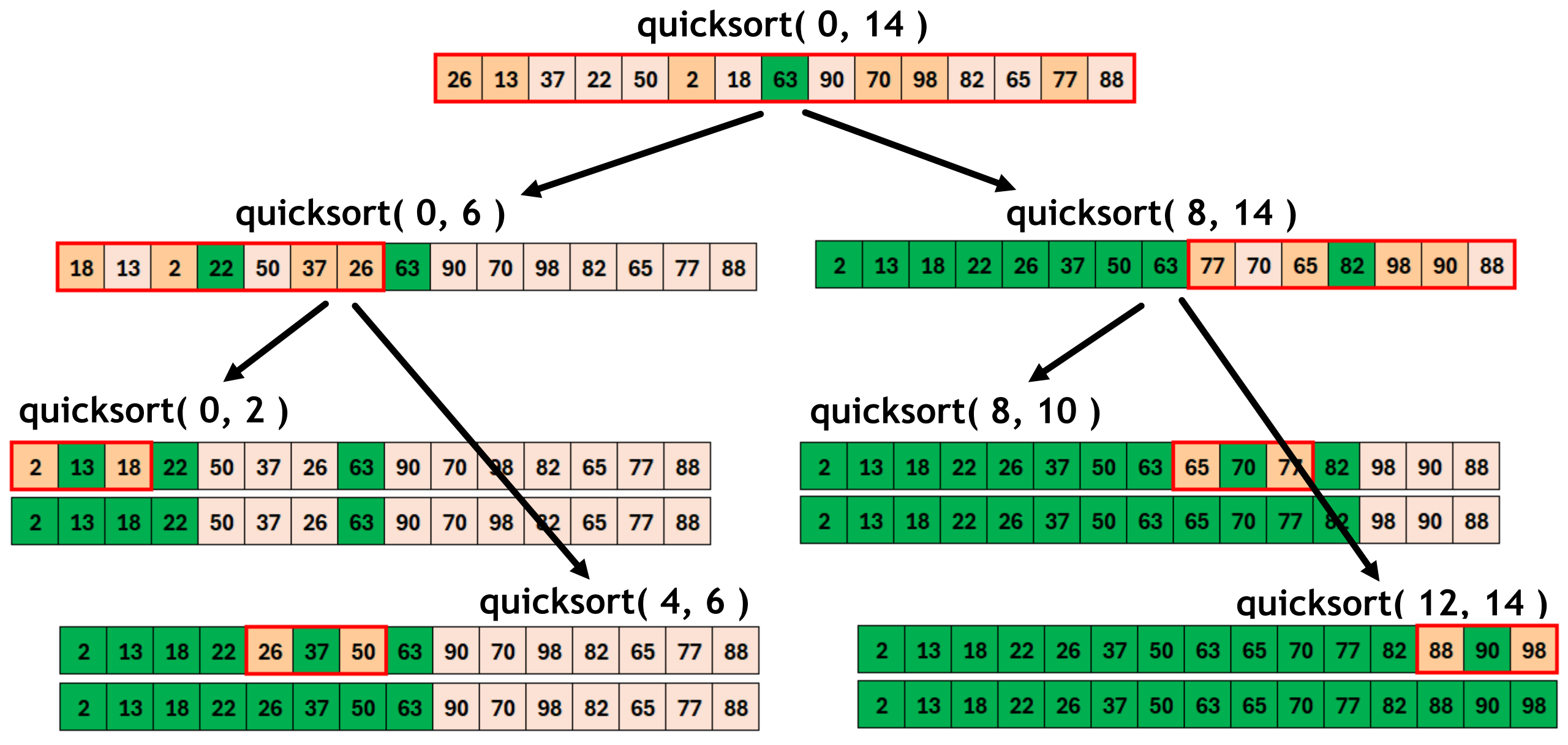
7-elemű tömb rendezése: https://ani.ide.sk/quicksort.html

Rekurzív függvény:

```
void quicksort(int t[], int kez, int veg) {
    int i = kez;
    int j = veg;
    int pivot = t[(i+j)/2];
    while (i<=j) {
        while (t[i]<pivot) { i++; }
        while (t[j]>pivot) { j--; }
        if (i<=j) {
            int tmp=t[i];
            t[i] = t[j];
            t[j] = tmp;
            i++;
            j--;
        }
    }
    if (kez<j) { quicksort(t,kez,j); }
    if (i<veg) { quicksort(t,i,veg); }
}
```

Pl. N=15-re:

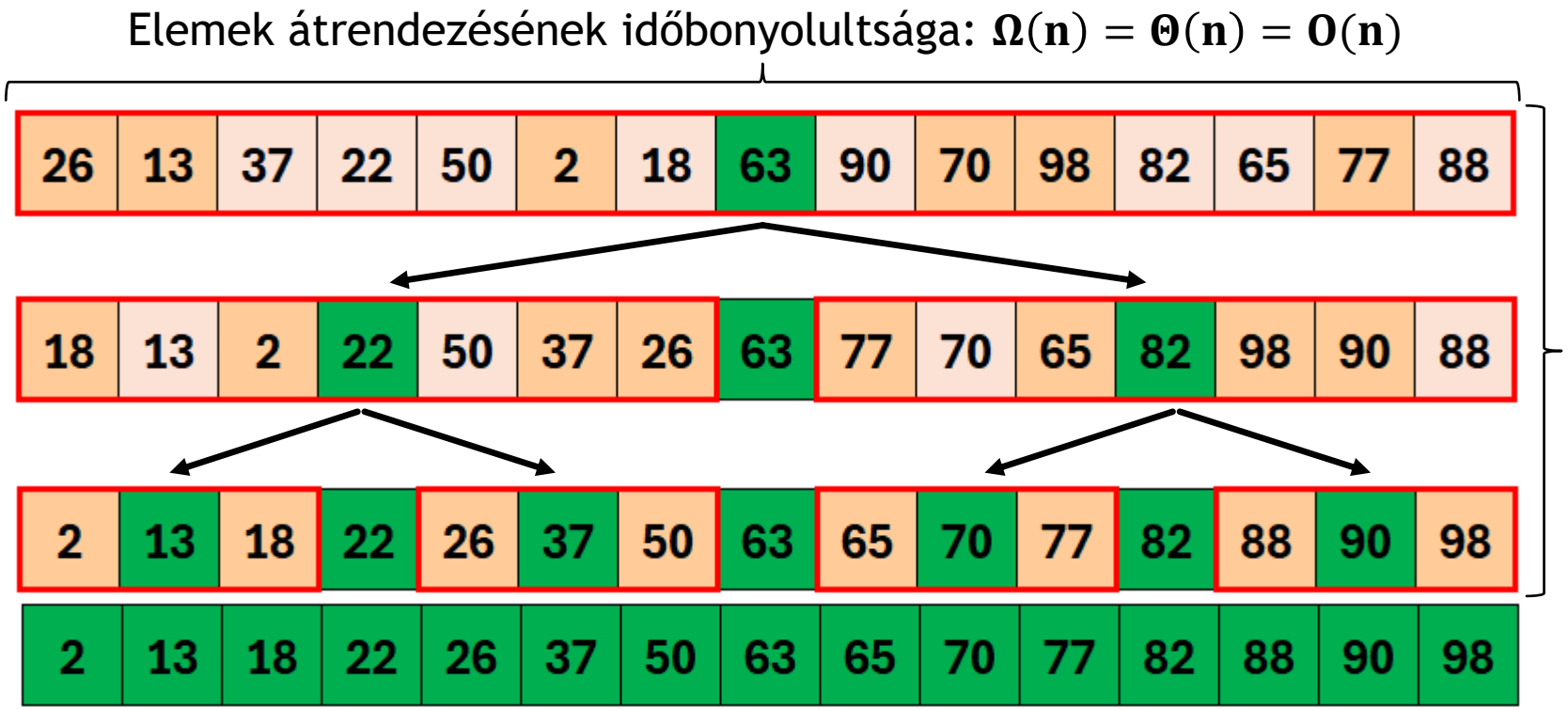
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]
77	98	37	22	50	70	18	63	90	2	13	82	65	26	88



Az előző dia a legjobb esetet szemléltette. Miért ez a legjobb eset?

A kiválasztott vezérelem **mindig két egyforma méretű részre** osztotta az adott tömbrészt.

Mennyi a legjobb eset időbonyolultsága?



A legjobb eset időbonyolultsága: $\Omega(n \log n)$

Mi történik akkor, ha a kiválasztott vezérelem nem egyforma méretű résztömbökre osztja fel az adott tömbrészt?

A rekurzív függvény a nagyobb méretű résztömbön többször hívódik meg.

Az átlagos eset időbonyolultsága: $\boldsymbol{\Theta(n \log n)}$

Mikor következik be a legrosszabb eset?

Ha a vezérelem mindig az adott tömbrész legkisebb vagy legnagyobb eleme.

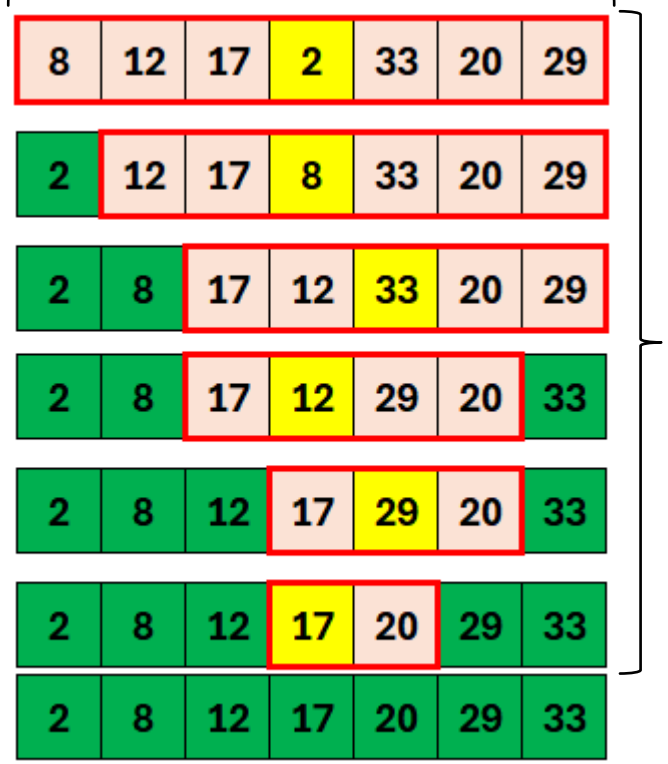
Ekkor a quicksort mindig egyel kisebb elemszámra fog lefutni.

Mennyi a legrosszabb eset időbonyolultsága?

Legrosszabb eset időbonyolultsága: $\boldsymbol{O(n^2)}$

De csakis akkor, ha a tömb elemei kezdetben úgy vannak kialakítva, hogy vezérelemnek mindig a legkisebb vagy legnagyobb elem kerül kiválasztásra!

Elemek átrendezésének időbonyolultsága:
$\boldsymbol{\Omega(n) = \Theta(n) = O(n)}$



Hogyan lehetne módosítani az algoritmust, hogy a legrosszabb eset nagyon alacsony eséllyel következzen be?

Hogyan lehet megoldani azt, hogy a legrosszabb esetre „előkészített” bemeneti tömbön se következzen be a legrosszabb eset?

- **A vezérelemet véletlenszerűen válasszuk ki** (tehát ne mindig a tömbrész közepén levő elemet válasszuk).
- **A rendezés előtt véletlenszerűen keverjük össze a tömbelemeket, pl. Fisher-Yates keveréssel** (ez után már választhatjuk akár a tömbrész közepén levő elemet is vezérelemnek).

```
for (int i=0; i<N-1; i++) {
    int r = rand() % (N-i) + i;  // i..N-1
    int tmp = a[i];
    a[i] = a[r];
    a[r] = tmp;
}
```

Ha a vezérelemet mindig véletlenszerűen választjuk ki, akkor mennyi az esélye a legrosszabb eset bekövetkezésének?

- A legrosszabb eset akkor következik be, ha vezérelemnek minden egyes rekurzív hívásnál a tömbrész legkisebb vagy legnagyobb elemét választjuk ki.
- Minden rekurzív hívásnál eggyel csökken az elemszám (N → N-1 → N-2 → … → 2), ezért annak valószínűsége hogy a legrosszabb eset következik be:

$$P_{rossz\ eset} = \frac{2}{N} \times \frac{2}{N-1} \times \frac{2}{N-2} \times \cdots \times \frac{2}{2} = \frac{2^{N-1}}{N!}$$

N	$P_{rossz\ eset}$
10	1.41×10^{-4}
20	2.15×10^{-13}
50	1.85×10^{-50}
100	6.79×10^{-129}
200	1.02×10^{-315}

A Földön megközelítőleg 7.5×10^{18} homokszem van.

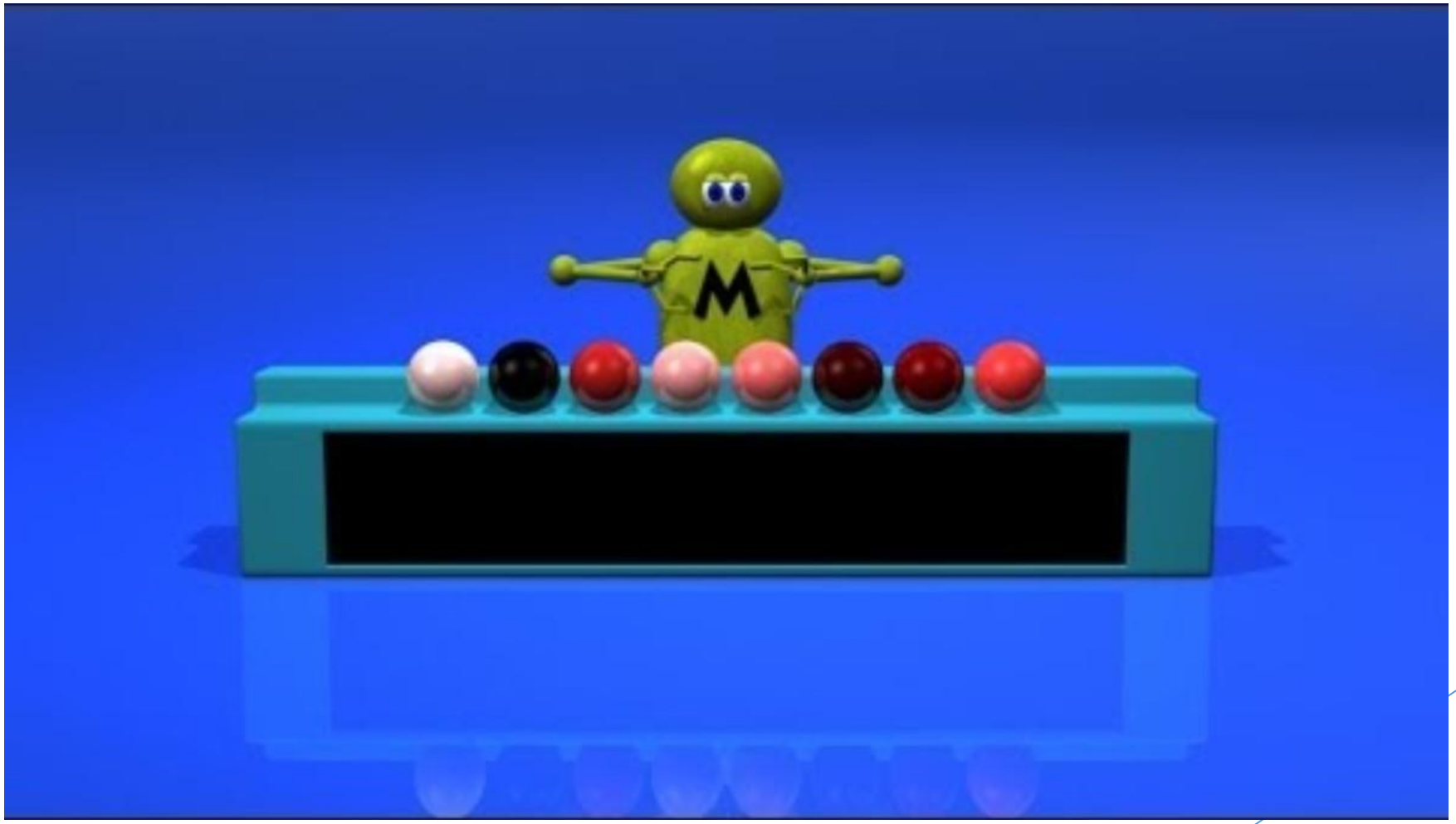
A látható univerzum kb. $10^{78} - 10^{82}$ atomból áll.

Mivel már nagyon kicsi N-re is gyakorlatilag lehetetlen annak a valószínűsége, hogy bekövetkezik a quicksort legrosszabb esete, ezért az algoritmus időbonyolultsága $\mathbf{O(n^2)}$ helyett:

$$\mathbf{O(n \log n)}$$

Összefésülő rendezés (mergesort)

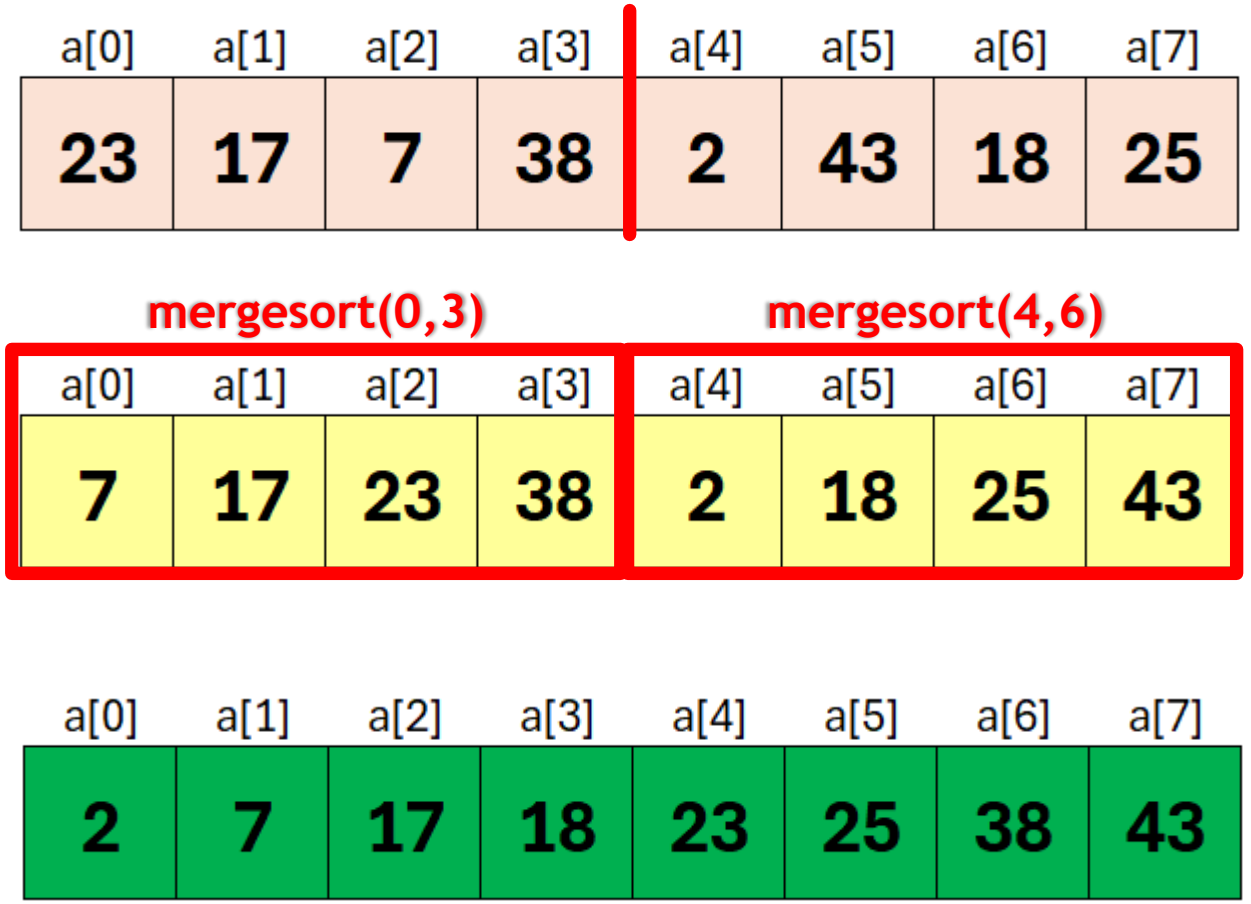
Egy rekurzív rendezési algoritmus, amely garantáltan O(n.log n) időbonyolultságú, és stabil rendezést biztosít.



https://www.youtube.com/watch?v=es2T6KY45cA

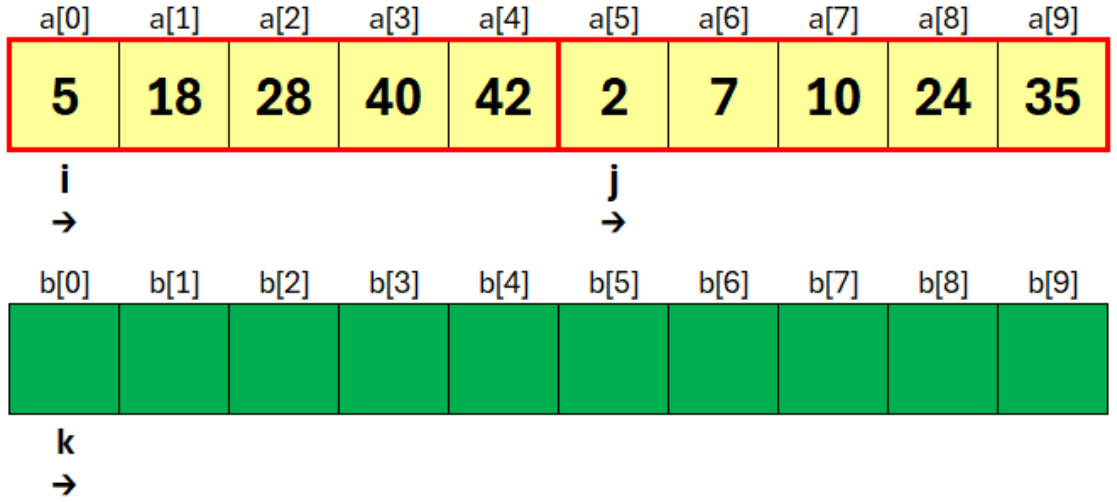
A mergesort rekurzív függvény működése:

- **A tömböt két egyenlő részre bontjuk.**
- **Mindkét résztömböt külön-külön rendezzük a mergesort függvény segítségével.**
- **A két rendezett résztömböt összefésüljük egy rendezett tömbbé.**



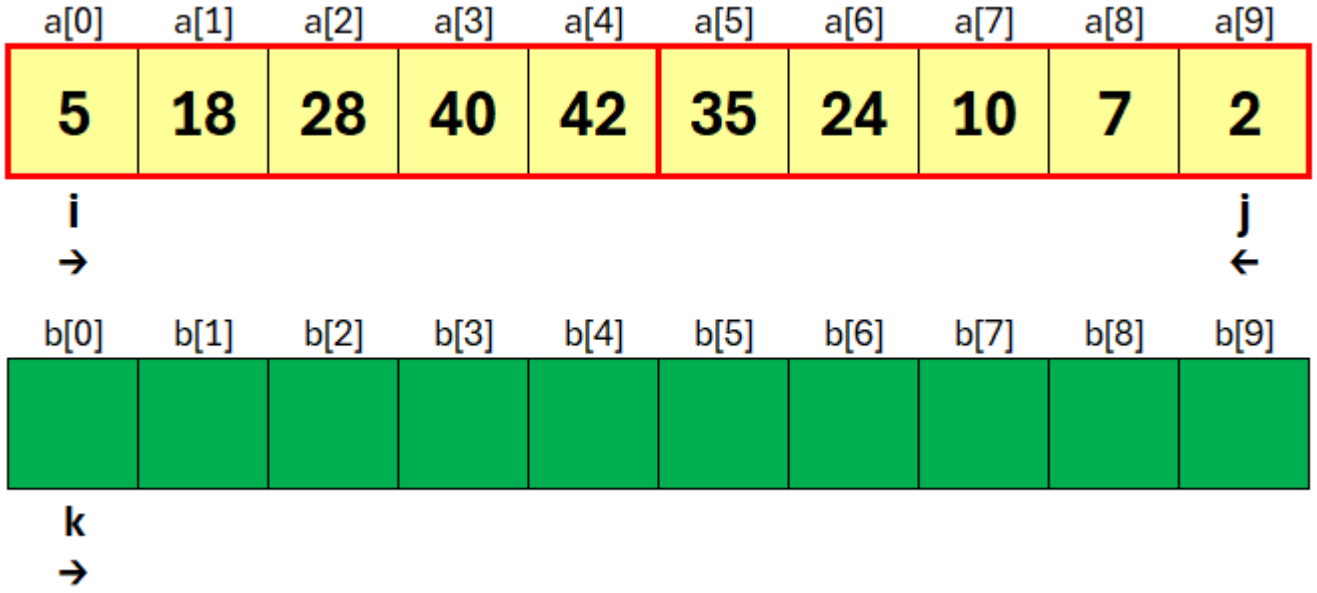
Két rendezett (növekvő-növekvő) tömbrész összefésülése:

```
int i = 0, j = N/2;          // ket "a" tombresz elejenek az indexe
int k = 0;                   // index "b" tombben ahova osszefesuljuk
while (i < N/2 && j < N) {   // amig nem ertunk egyik "a" tombresz vegere sem
    if (a[i] < a[j]) {
        b[k] = a[i];
        i++;
    } else {
        b[k] = a[j];
        j++;
    }
    k++;
}
while (i < N/2) {            // ha maradt meg elem az elso tombreszben
    b[k] = a[i];
    i++;
    k++;
}
while (j < N) {              // ha maradt meg elem a masodik tombreszben
    b[k] = a[j];
    j++;
    k++;
}
```



Két rendezett (növekvő-csökkenő) tömbrész összefésülése:

```
int i = 0, j = N-1;                 // elso tombresz elejenek indexe,
                                    // masik tombresz vegenek indexe
for (int k=0; k<N; k++) {           // atrakjuk az osszes elemet "b" tombbe
    if (a[i]<a[j]) {
        b[k] = a[i];
        i++;
    } else {
        b[k] = a[j];
        j--;
    }
}
```



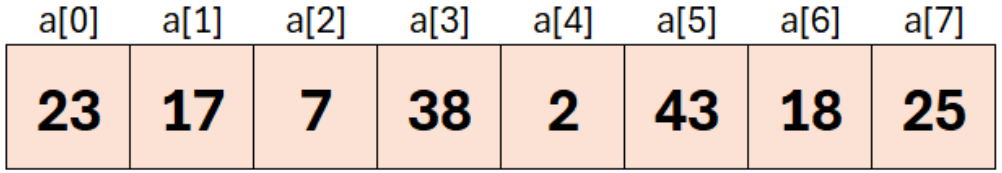
Ha valamelyik tömbrészben elfogynak az elemek, akkor annak a tömbrésznek az indexe a másik tömbrész legnagyobb elemére fog mutatni.

8-elemű tömb rendezése: https://ani.ide.sk/mergesort.html

Rekurzív függvény:

```
void mergesort(int t[], int kez, int veg, int x[]) {
    if (kez<veg) {
        int m = (kez+veg)/2;
        mergesort(t,kez,m,x);
        mergesort(t,m+1,veg,x);
        for (int i=kez; i<=m; i++) { x[i] = t[i]; }
        int j = veg;
        for (int i=m+1; i<=veg; i++) { x[j] = t[i];
                                       j--; }
        int i = kez;
        j = veg;
        for (int k=kez; k<=veg; k++) {
            if (x[i]<x[j]) {  t[k] = x[i];
                              i++; }
            else { t[k] = x[j];
                   j--; }
        }
    }
}
```

Pl. N=8-ra:



mergesort(0, 7)

23 17 7 38 2 43 18 25

2 7 17 18 23 25 38 43

mergesort(0, 3)

23 17 7 38 2 43 18 25

7 17 23 38 2 43 18 25

mergesort(4, 7)

7 17 23 38 2 43 18 25

7 17 23 38 2 18 25 43

mergesort(0, 1)

23 17 7 38 2 43 18 25

17 23 7 38 2 43 18 25

mergesort(2, 3)

17 23 7 38 2 43 18 25

17 23 7 38 2 43 18 25

mergesort(4, 5)

7 17 23 38 2 43 18 25

7 17 23 38 2 43 18 25

mergesort(6, 7)

7 17 23 38 2 43 18 25

7 17 23 38 2 43 18 25

mergesort(0, 0) *visszatérés*

mergesort(1, 1) *visszatérés*

mergesort(2, 2) *visszatérés*

mergesort(3, 3) *visszatérés*

mergesort(4, 4) *visszatérés*

mergesort(5, 5) *visszatérés*

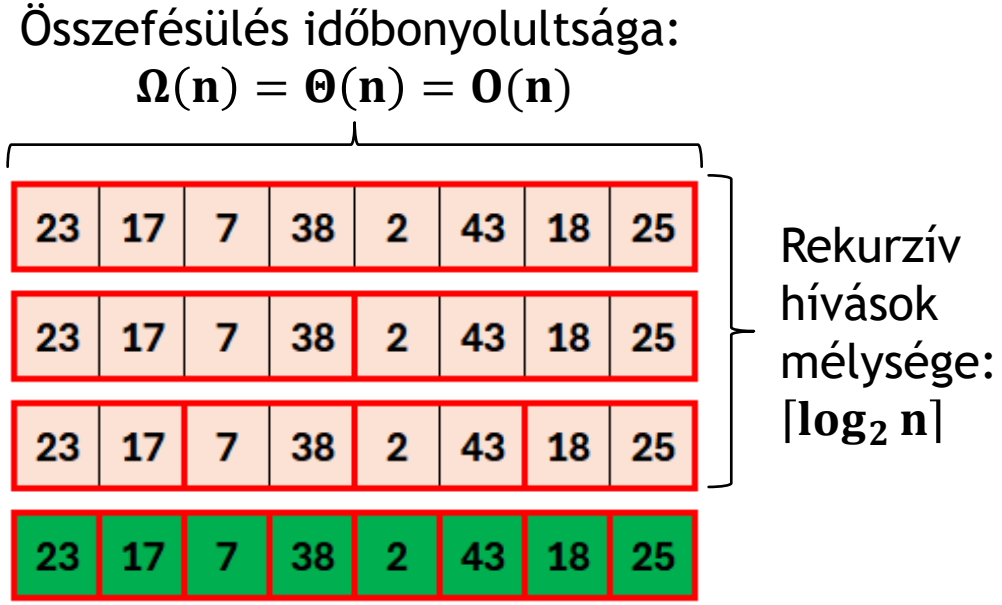
mergesort(6, 6) *visszatérés*

mergesort(7, 7) *visszatérés*

Mi történik akkor, ha a tömb nem pontosan 2^k darab elemet tartalmaz?

Ilyenkor a rendezés közben előfordulhat, hogy az adott tömbrész nem mindig két egyforma részre osztódik (az egyik rész eggyel több elemet tartalmazhat). Ez azonban nincs hatással a rendezés folyamatára.

Mennyi a mergesort algoritmus időbonyolultsága?



A mergesort algoritmus futása nem függ a tömb elemeinek kezdeti értékeitől, ezért minden esetre az időbonyolultság:

$$\Omega(n \log n) = \Theta(n \log n) = O(n \log n)$$

A mergesort memóriaigénye azonban nagyobb, mint a quicksorté, mivel egy segédtömböt használ az összefésüléshez.