ADATSZERKEZETEK ÉS ALGORITMUSOK

Gyorsrendező II.
Kupacrendező

Gyorsrendezés – másik felosztási algoritmus

- Többféle algoritmus létezik a felosztásra
  - Nézzünk meg még egyet – pszeudokódban
- FELOSZT(A,p,r)

```
x ← A[r]
i ← p - 1
for j ← p to r-1
   do if A[j] ≤ x then
      i ← i+1
      A[i] ↔ A[j] csere
A[i+1] ↔ A[r] csere
return i+1
```

Gyorsrendezés – másik felosztási algoritmus

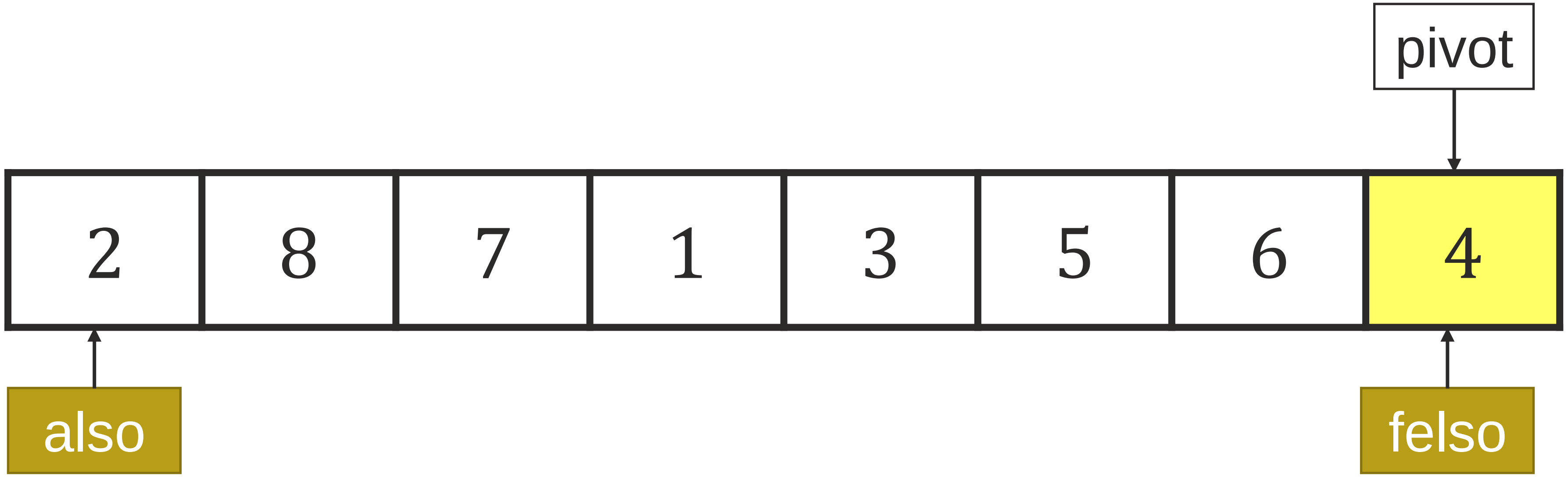
- Többféle algoritmus létezik a felosztásra
  - Nézzünk meg még egyet – pszeudokódban
- FELOSZT(A,also,felso)

```
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
     i←i+1
     Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```

Quicksort – Megosztás

- Bármelyik elem jó strázsának, válasszuk a felsőt

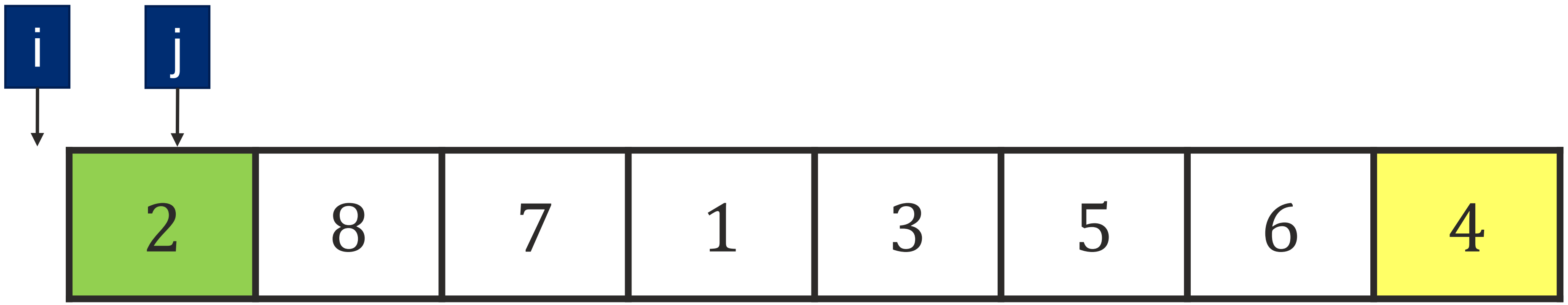
```
FELOSZT(A,also,felso)
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
    i←i+1
    Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```



Quicksort – Megosztás

- Induljunk el az i és j indexszel!

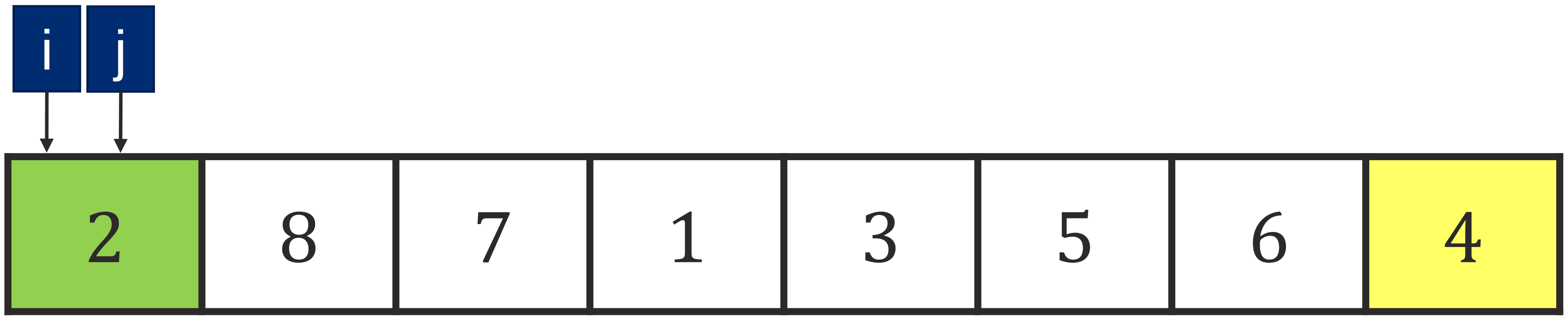
```
FELOSZT(A,also,felso)
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
    i←i+1
    Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```



Quicksort – Megosztás

- A j pozíción találunk a strázsánál kisebb elemet
- Növeljük az i-t
  - És cseréljük saját magával

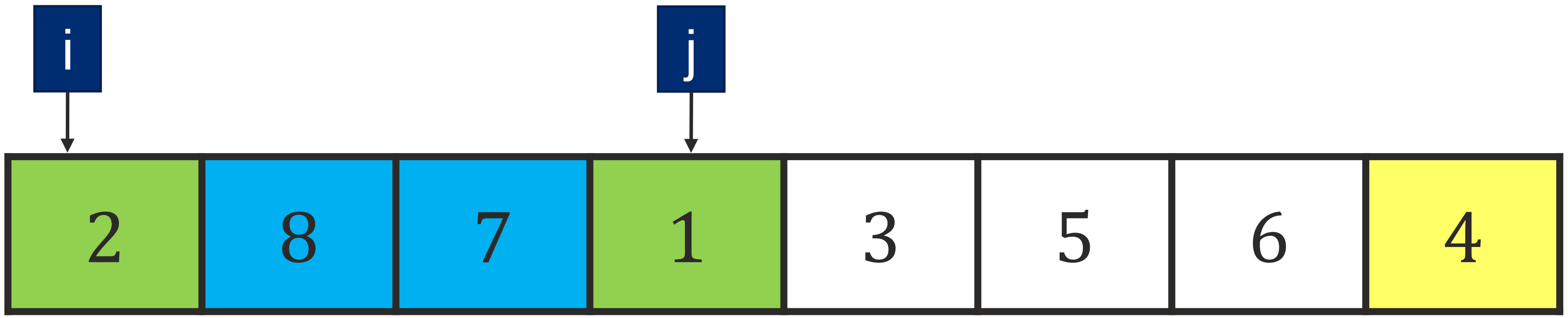
```
FELOSZT(A,also,felso)
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
    i←i+1
    Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```



Quicksort – Megosztás

- A j pozíción találunk a strázsánál kisebb elemet
- Növeljük az i-t

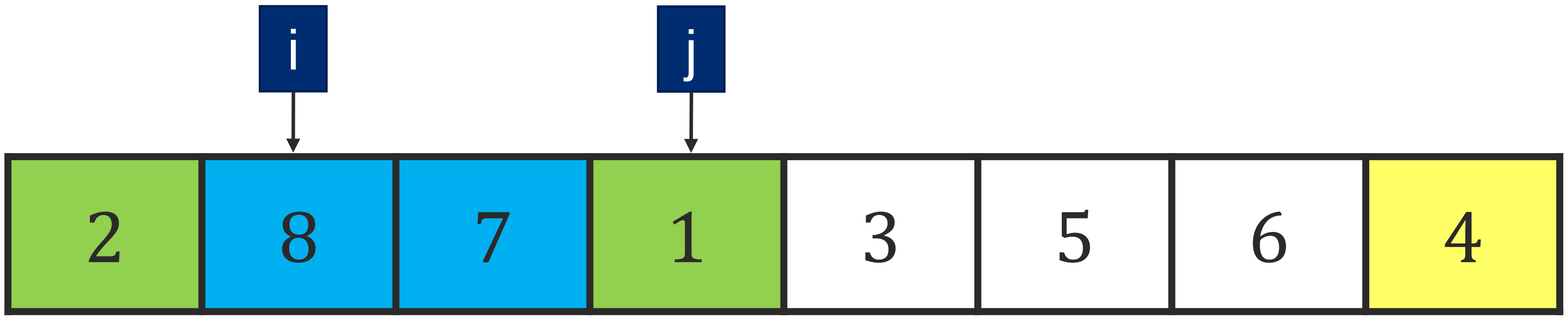
```
FELOSZT(A,also,felso)
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
    i←i+1
    Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```



Quicksort – Megosztás

- És megcseréljük az i és a j elemet

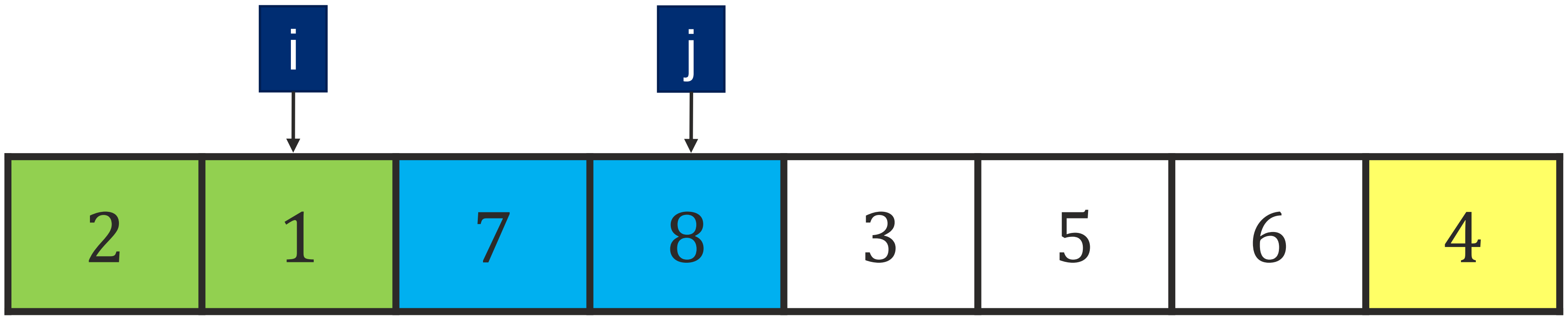
```
FELOSZT(A,also,felso)
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
    i←i+1
    Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```



Quicksort – Megosztás

- Majd folytatjuk a j növelését

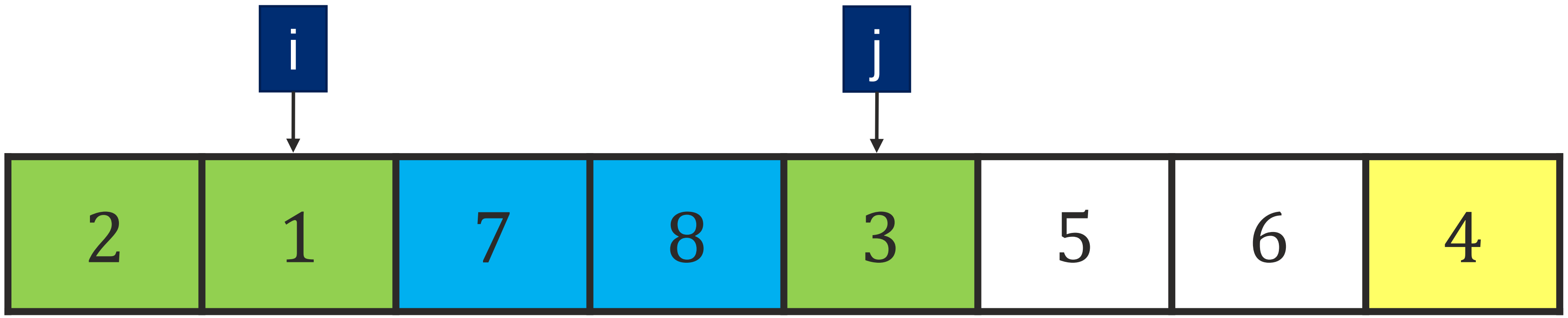
```
FELOSZT(A,also,felso)
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
    i←i+1
    Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```



Quicksort – Megosztás

- Újabb találat

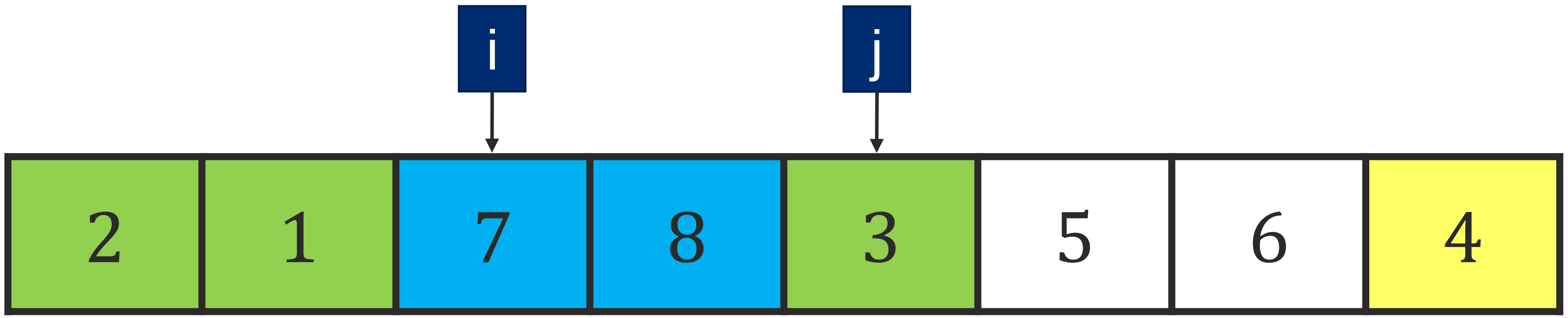
```
FELOSZT(A,also,felso)
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
    i←i+1
    Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```



Quicksort – Megosztás

- Újabb csere

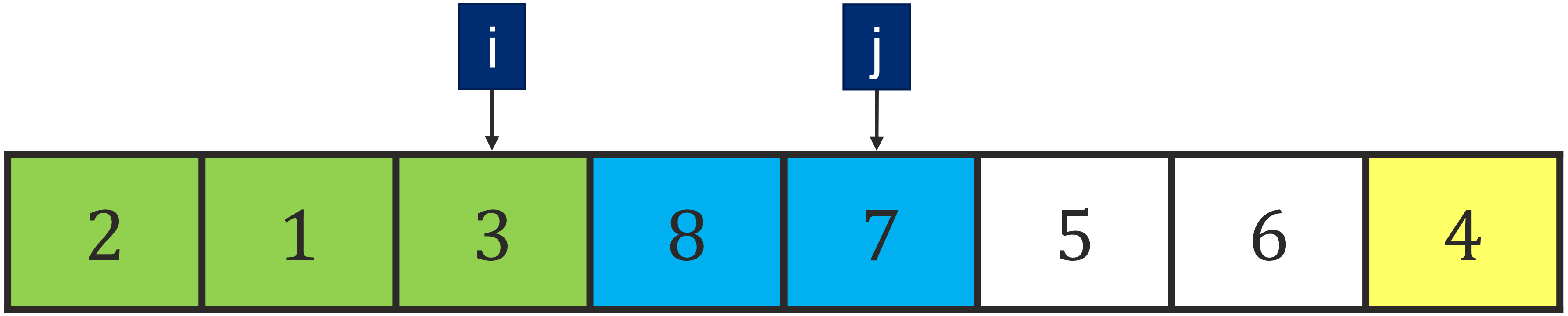
```
FELOSZT(A,also,felso)
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
    i←i+1
    Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```



Quicksort – Megosztás

- …

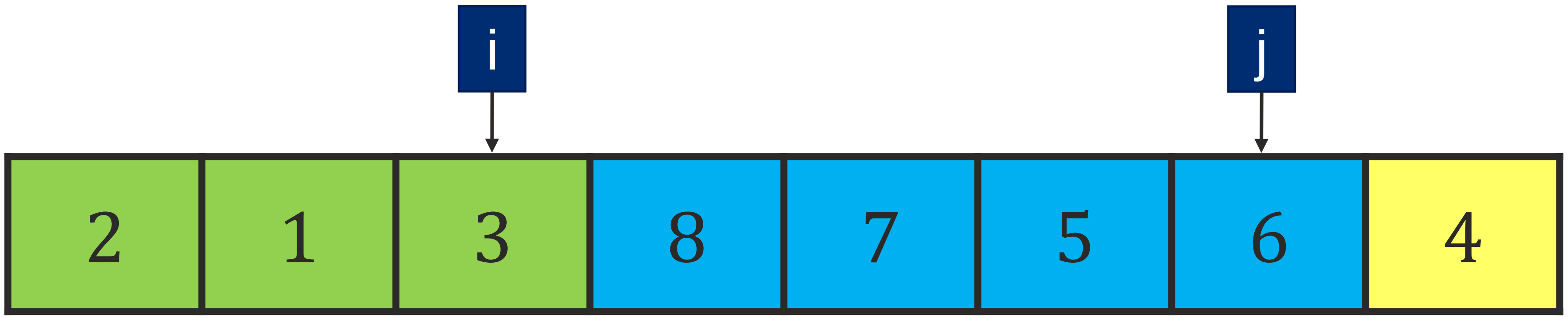
```
FELOSZT(A,also,felso)
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
    i←i+1
    Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```



Quicksort – Megosztás

- Végezetül a strázsa elemet a helyére tesszük

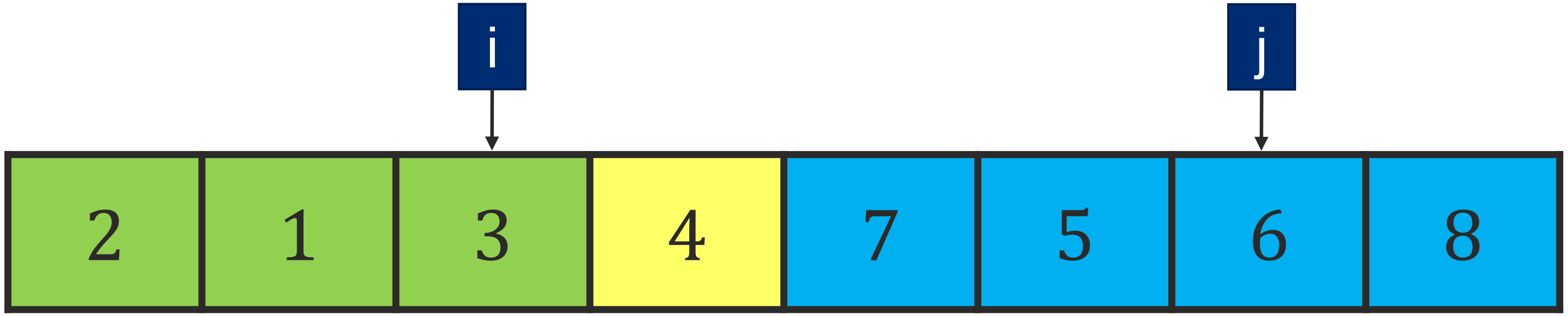
```
FELOSZT(A,also,felso)
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
    i←i+1
    Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```



Quicksort – Megosztás

- …

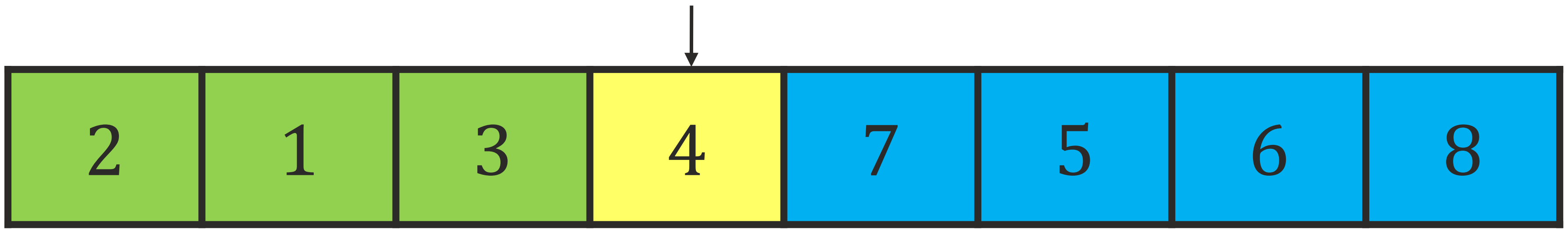
```
FELOSZT(A,also,felso)
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
    i←i+1
    Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```



Quicksort – Megosztás

- Legvégül visszatérünk a strázsa pozíciójával

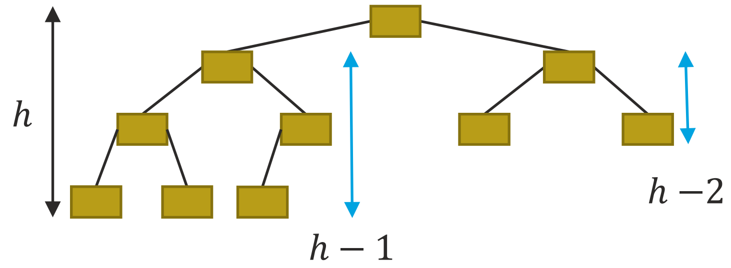
```
FELOSZT(A,also,felso)
str←A[felso]
i←also-1
for j←also to felso-1
  do if A[j]≤str then
    i←i+1
    Csere(A[i],A[j])
Csere(A[i+1],A[felso])
return i+1
```

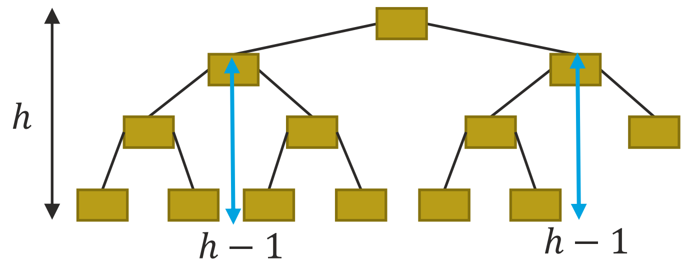


Kupacrendezés

Teljes fák – majdnem teljes fák

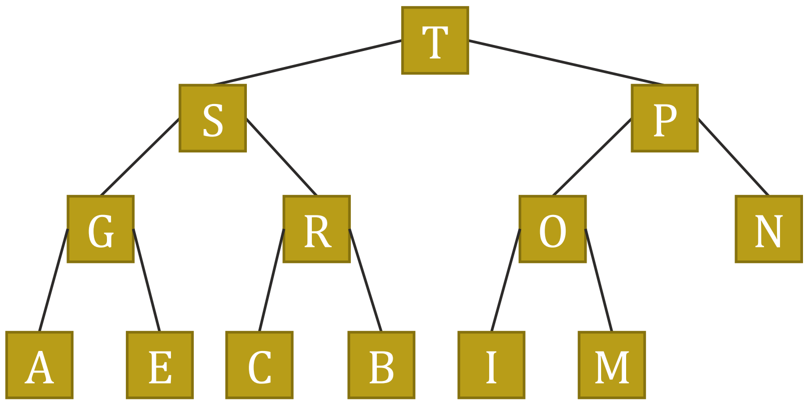
- Egy bináris fa teljes, ha
  - a magassága h, és
  - $2^h - 1$ csomópontja van
- Egy h magasságú bináris fa majdnem teljes, ha
  - Üres, vagy
  - A magassága h, és a bal részfája $h-1$ magas és majdnem teljes és jobb részfája $h-2$ magas és teljes, vagy
  - A magassága h, és a bal részfája $h-1$ magas és teljes és jobb részfája $h-1$ magas és majdnem teljes





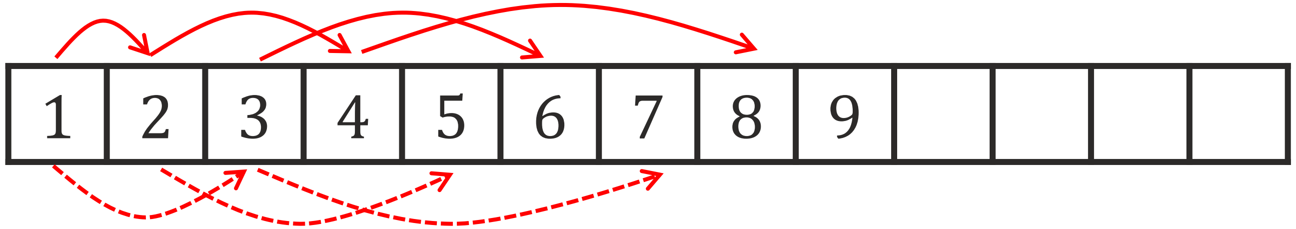
Kupac

- Egy majdnem teljes (bináris) fa heap tulajdonságú, ha
  - Üres, vagy
  - A gyökérben lévő kulcs nagyobb, mint mindkét gyerekében és mindkét részfája is heap tulajdonságú



Kupac – hatékonyság

- A beszúró és törlő műveletek egy h magasságú fában
  - $h \leq \log_2 n$
  - Vagyis $\mathcal{O}(\log n)$ az időigénye



Kupacrendezés

- Használjuk a kupacokat rendezésre!
  - Szúrd be az elemeket egy kupacba!
  - Amíg a sor ki nem ürül, vedd ki a kupacból a maximális elemet, és tedd az eredmény (rendezett) sorba!

Kupacrendezés

- Az *s* sorban lévő elemeket rendezzük a *k* kupac segítségével!

k.empty
¬s.isempty
k.insert(s.out)
¬k.isempty
s.in(k.delmax)

Kupacrendezés

- A kupacok hatékony rendezőt adnak
  - Szúrjunk be minden elemet a kupacba: n elem, ezekhez kell $\mathcal{O}(\log_2 n)$
    - összesen: $\mathcal{O}(n \log_2 n)$
  - sorrendben távolítsuk el az elemeket: n elem, ezekhez kell $\mathcal{O}(\log_2 n)$
    - összesen: $\mathcal{O}(n \log_2 n)$
- Összesen $\mathcal{O}(n \log_2 n)$
  - ignorálva a 2-es konstanst, stb.

Heapsort vs. Quicksort

- Használjuk inkább a Heapsort-ot?
- A Quicksort általában gyorsabb
  - kevesebb összehasonlítás és csere
  - néhány empirikus adat

	Quick		Heap		Beszúró	
	Összehasonlítás	Csere	Összehasonlítás	Csere	Összehasonlítás	Csere
100	712	148	2842	581	2596	899
200	1682	328	9736	9736	10307	3503
500	5102	919	53113	4042	62746	21083

Quicksort ⇔ Heap Sort

- Quicksort
  - általában gyorsabb
  - néha $O(n^2)$
    - jobb pivot választás csökkenti a valószínűségét
    - ha átlagosan jó végrehajtási időt akarunk, használjuk ezt
    - üzleti alkalmazások, információs rendszerek
- Heap Sort
  - általában lassúbb
  - Garantált $O(n \log n)$ … lehet rá építeni, ezt a tervezésben felhasználni!
  - használjuk például real-time rendszerekhez
    - Az idő egy megszorítás

Összefésüléses rendezés

Következő téma