

# Rendezések (összefésülő, beszűrő, gyors- és hatékonyságuk). Összehasonlító rendezések hatékonysága.

## Rendezések

A rendezési algoritmusoknak két fő típusa van: az összehasonlító és a nem összehasonlító (melyek az elemek összehasonlítása nélkül képesek erre) rendezések. Egy másik szempont a rendezés stabilitása, azaz hogy az azonosnak ítélt elemek közötti sorrendet megőrzi-e.

A rendezések hatékonyságát, összehasonlítását általában a szükséges összehasonlítások, cserék átlagos és maximális száma és az extra tárigény alapján végezzük.

## Összehasonlító rendezések

- A bemenő tömb rendezettségét az elemei közötti összehasonlítási reláción keresztül értelmezzük,
  - Döntési fa: csomópontjai a döntések, levelei az egyes lehetséges bemenő sorok. Rendezés: fábejárás gyökértől levél felé (közben mozgatás)
  - Leveleken: bemenő sorról teljes információ (permutáció)
  - Lépések (összehasonlítások) száma: a fa magassága
  - Tétel:  $n$  elemet rendező döntési fa  $\Omega(n \log n)$  magas (legalább)
  - Biz: t.f.h.  $n$  elemű sort rendező  $h$  magas fa. Levelek (permutációk) száma legalább  $n!$ . F leveleinek száma:  $\leq 2^h \Rightarrow n! \leq 2^h \Rightarrow h \geq \lg(n!)$  Viszont (Stirling formula):  $n! > (n/e)^n \Rightarrow h > \lg(n/e)^n \Rightarrow h > n(\lg(n) - \lg(e)) = \Omega(n \log n)$
  - Következmény: az  $n \log n$  rendezések (kupac- és összefésülő rendezés) aszimptotikusan optimális rendezések
- Egy összehasonlító rendezés maximális összehasonlítás-száma  $n$  elemű input esetén legalább  $\log_2(n!)$

## **Nem összehasonlító rendezések**

Ezek az algoritmusok nem használják az elemek között fennálló rendezési műveletet (ha egyáltalán van ilyen) az elemek összehasonlítására, ezért általában valamely más, speciálisabb követelményt támasztanak, ezért általános esetben nem használjuk őket.

Jelölések:  $n$  az elemek száma,  $k$  a kulcsérték mérete. Az algoritmusok feltételezik hogy a kulcsok egyediek és hogy  $n \ll 2^k$  ( $n$  lényegesen kisebb, mint  $2^k$ ).

## Beszűrő rendezés

Az emberi gondolkodáshoz közel álló algoritmus, amely egyszerre egy elemet visz a helyére.

Stabil rendezés.

Átlagos eset:  $\mathcal{O}(n^2)$   
Legrosszabb eset:  $\mathcal{O}(n^2)$   
Tárigény:  $\mathcal{O}(1)$

### Beszúró rendezés (ábra)

- 5 - 2 6 4 1 3 2 6
- 2 5 - 6 4 1 3 2 6
- 2 5 6 - 4 1 3 2 6
- 2 4 5 6 - 1 3 2 6
- 1 2 4 5 6 - 3 2 6
- 1 2 3 4 5 6 - 2 6
- 1 2 2 3 4 5 6 - 6

Sebessége attól is függ, hogy mennyire rendezett a bemeneti lista...

Ha teljesen, akkor  $T(n) = \Theta(n)$

Egyébként:  $T(n) = \Theta(n^2)$

### Beszúró rendezés hatékonyságvizsgálata

```
for (int j=1; j<vektor.length; j++) {  
    int kulcs = vektor[j];  
    int i=j-1;  
    while (i>0 && vektor[i]>kulcs) {  
        vektor[i+1] = vektor[i];  
        i--;  
    }  
    vektor[i+1] = kulcs;  
}
```

Ciklus időszükséglete (ha a ciklusszám=bemenő hossz):

$$T(n) = (c_1+c_2) + (c_1+c_2*2) + \dots + c_1 + (c_2*(n-1)) = c_1*(n-1) + c_2*(1+2+\dots+(n-1)) = c_1*(n-1) + c_2*n*(n-1)/2 = \Theta(n^2)$$

$$T(n) = c_1*n * T_{\text{ciklusmag}}(n)$$

$$T_{\text{ciklusmag}}(n) = c_2*n/2$$

$$\rightarrow T(n) = c_1*c_2/2*n*n$$

$$\rightarrow T(n) = \Theta(n^2)$$

→ négyzetes algoritmus

### Gyorsrendezés

Az egyik legnépszerűbb rendezési algoritmus, amely átlagos esetben gyorsabb, mint a többi algoritmus, viszont hátránya hogy a legrosszabb esetben lassú, és nem stabil rendezés.

Rekurzív algoritmus, kettéosztja a rendezendő listát egy kiemelt elemnél kisebb és nagyobb elemekre, majd a részekre külön-külön alkalmazza a gyorsrendezést.

Átlagos eset:  $\mathcal{O}(n \log n)$

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

Tárigénye:  $\mathcal{O}(\log n)$

A gyorsrendezés oszd meg és uralkodj elven működik: a rendezendő számok listáját két részre bontja, majd ezeket a részeket rekurzívan, gyorsrendezéssel rendezi. A felbontáshoz kiválaszt egy támpontnak nevezett elemet (más néven pivot, főelem vagy vezérelem), és particionálja a listát: a támpontnál kisebb elemeket eléje, a nagyobbakat mögéje mozgatja. Teljes indukcióval könnyen belátható, hogy ez az algoritmus helyesen működik.

- Alapelve: (oszd meg és uralkodj)

1. felosztjuk az  $A[p..r]$ -t  $A1[p..q]$  és  $A2[q+1..r]$  szakaszokra úgy, hogy  $A1 \leq A2$  (DE!!

NINCSenek rendezve!!)

2. Az  $A1$  és  $A2$  résztömböket rendezzük

3. Nincs szükség összefésülésre

- Az algoritmus sarokpontja: az 1. felosztó lépés. Ennek megvalósítása többféle is lehet

```
public void gyorsrendez(int also, int felso) {  
    if (also < felso) {  
        int kozep = feloszt(also, felso)
```

### **Gyorsrendezés felosztó algoritmus**

→ A  $tomb(also, felso)$  résztömböt helycserék útján felosztjuk úgy, hogy  $tomb(also, kozep) < tomb(kozep+1, felso)$

→ A tömb első eleme az őr, a két résztömböt elválasztó érték.

→ A két szélétől indulva növesztjük a kikötéseknek megfelelő résztömböket addig, amíg összeérnek

### **Hatékonyság: Legrosszabb felosztás**

→ Legrosszabb eset, ha  $1 \dots n-1$  tömbökre osztunk fel

→ Rekurziós fa:

→ Felosztás:  $\Theta(n)$ , mintezt  $n$ -szer kell megtenni:

→  $T(n) = T(n-1) + \Theta(n)$

→  $T(n) = \sum_{i=1}^n \Theta(i) = \Theta(\sum_{i=1}^n i) = \Theta(n^2)$

→ Maximálisan kiegyensúlyozatlan a felosztás

### **Hatékonyság: Kiegyensúlyozott felosztás**

Hatékonyság:  $\Theta(n \cdot \lg(n))$  hasonlóan pl. a kupacrendezéshez

### **Mitől függ a hatékonyság?**

- Mikor lesz  $1 \dots n-1$  felosztás?  $1 \mid 4, 6, 7, 3, 2 \square$  az első a legkisebb/legnagyobb elem  $6, 2, 5, 4, 3, 1 \rightarrow$  csere  $6-1 \square 1, 2, 5, 4, 3 \mid 6$
- Pl. a már sorbarendeztet lista: extrém rossz
- Javaslat1: véletlen keverés  $O(n)$  időben
- Javaslat2: véletlen felosztás:
  - Örkefejezés a lista véletlen eleme legyen
  - Az első elemet egy véletlen elemmel kicseréljük
- Veremmelység: a közölt rekurzív algoritmusra:  $\Theta(\lg(n))$
- A rekurzió azonban kiküszöbölhető!!

**Összefésülő rendezés:** alapötlete, hogy eleve rendezett elemsorozatok aránylag egyszerűen vonhatók össze rendezett sorozattá. Az algoritmus lényege, hogy az elemeket két csoportba osztjuk, a csoportokat rendezzük (akár összefésülő rendezéssel), majd a kapott részeket összefésüljük.

### **Összefésülő rendezés hatékonyságvizsgálata**

• Idő/sorozathossz függvény  $T(n)$

1.  $T(1) = c_1$  egyenlő sorozat rendezése állandó

2.  $T_{\text{felosztás}}(n) = c_2$  sorozat felosztása állandó

3.  $T_{\text{turalkodás}}(n) = 2 \cdot T(n/2)$  2\* félsorozat ideje

4.  $T_{\text{összefésülés}}(n) = c_3 \cdot n$

$T(1) = c_1$ .

$T(n) = c_2 + 2T(n/2) + c_3 \cdot n$  rekurzív egyenlet

**$T(n) = \Theta(n \cdot \lg(n))$  bizonyítás nélkül**

## Buborékrende­zés

A buborékrende­zés egy egyszerű algoritmus, amelyet főleg az oktatásban használunk, mivel nem igazán hatékony. Elve, hogy egy "buborékkal" haladva a tömbben több menetben előlről hátra a buborékban szereplő két elemet felcseréljük, ha azok rossz sorrendben vannak.

Stabil rende­zés.

Átlagos eset:  $\mathcal{O}(n^2)$

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

Tárigénye:  $\mathcal{O}(1)$

(O: „ordó”)

## Kupacrende­zés

A kupacrende­zés a használt adatszerke­zetről kapta a nevét, a [kupacról](#). Működése során felépíti a kupacot, majd egyesével kiemeli a gyökérelemet, ami a kupac definíciója miatt a legnagyobb/legkisebb elem lesz.

Stabil rende­zés.

Átlagos eset:  $\mathcal{O}(n \log n)$

Átlagos eset:  $\mathcal{O}(n \log n)$

Tárigénye:  $\mathcal{O}(1)$