## 1. gyakorlat

### Téma:

Algoritmusok műveletigényének meghatározása, hatékonyság, hatékonyság jellemzése (aszimptotikus korlátok bevezetése).

### 1 feladat:

Polinom helyettesítési értékének kiszámítása. Adott egy n-ed fokú polinom, határozzuk meg egy adott x helyen felvett értékét: an\*xn+an-1\*xn-1+ …. a1\*x+a0(Tfh. nagyon sok polinomunk van, és nagyon sok helyen kell kiszámítani az értékét, ezért készítsünk minél hatékonyabb megoldást.)

A polinom együtthatóit egy nullától indexelt, n+1 méretű tömbben helyezzük el.

*Megállapodás: tömbök jelölése:   
A:T[n] – ez nullától indexelt, n méretű tömb, melynek elemei T típusúak.   
A/1:T[n] – ez egy 1-től indexelt, n méretű tömb, melynek elemei T típusúak.   
Az algoritmusokban a tömb méretére a „length” tulajdonsággal hivatkozhatunk: A.length=tömb elemszáma*

Feladatunkban tehát a polinom együtthatói legyenek az A:R[n+1] tömbben.

A megoldásoknál írjuk fel, hogy az egyes lépések hányszor hajtódnak végre. Vizsgáljuk meg a ciklusiterációk *it(n)*, a szorzások *S(n*) és az összeadások *Ö(n)* számát, a polinom fokszámának függvényében.

Feltehető, hogy n≥0, azaz A.length>0

**Első megoldás, az összegzés tételéből származik:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Polinom(A:R[n+1]; x:R):R** | | | Végrehajtások száma |
| y:=A[0] | | | 1 |
| i=1 to n | | | n+1 |
|  | h:=x | | n |
| j=1 to i-1 | | 1+2+..+n |
|  | h:=h\*x | 0+1+2+..+n-1 |
| y:=y+A[i]\*h | | n |
| return y | | | 1 |

it(n)=((n-1)\*n)/2= (n2-n)/2

S(n)=(n\*(n+1))/2=(n2+n)/2

Ö(n)=n

**Második megoldás, x hatványait rekurzívan számoljuk a h változóban: xi=xi-1\*x, ha i>0, x0=1**

|  |  |  |
| --- | --- | --- |
| **Rekurzív(A:R[n+1]; x:R):R** | | Végrehajtások száma |
| y:=A[0] h:=1 | | 1 |
| i=1 to n | | n+1 |
|  | h:=h\*x | n |
| y:=y+A[i]\*h | n |
| return y | | 1 |

it(n)=n

S(n)=2n

Ö(n)=n

**Harmadik megoldás, a Horner séma:**

y=(…(an\*x+an-1)\*x+an-2)\*x+ …+ a1)\*x + a0

|  |  |  |
| --- | --- | --- |
| **Horner(A:R[n+1]; x:R):R** | | Végrehajtások száma |
| y:=A[n] | | 1 |
| i=n-1 downto 0 | | n+1 |
|  | y:=y\*x+A[i] | n |
| return y | | 1 |

it(n)=n

S(n)=n

Ö(n)=n

Hogyan hasonlíthatók össze a kapott eredmények?

**Aszimptotikus korlátok (Θ,Ο,Ω,ο,ω)**

A továbbiakban legyen *f* olyan függvény, amelyet a természetes számok halmazán értelmezünk és nem-negatív valós értékeket vesz fel (lépésszám, futási idő, tárméret):

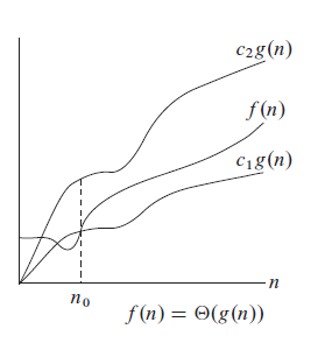
}

Definiáljuk egy adott *g* függvény esetén azon függvények osztályait, amelyek nagyságrendben rendre

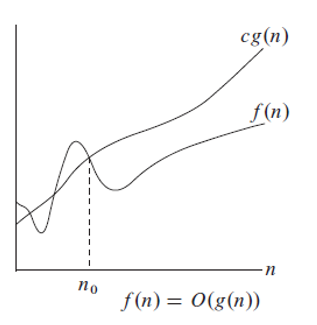
* nem nagyobbak, mint *g*: *O*(*g*) „nagy ordó”,
* kisebbek, mint *g*: *ο*(*g*) ”kis ordó”,
* nem kisebbek, mint *g*: Ω(*g*) „nagy omega”,
* nagyobbak, mint *g*: ω(*g*) „kis omega”,
* illetve g-vel azonos nagyságrendűek: Θ(*g*) „theta”.

**Definíciók:**



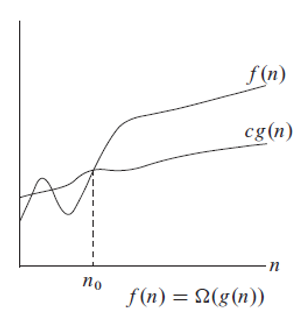
****

**Θ(g) halmazba tartozó függvényeknek a g függvény aszimptotikusan éles korlátja.**



**O(g) halmazba tartozó függvényeknek a g függvény aszimptotikusan felső korlátja.**





**Ω(g) halmazba tartozó függvényeknek a g függvény aszimptotikusan alsó korlátja.**

Jellemezzük három megoldásunkat:

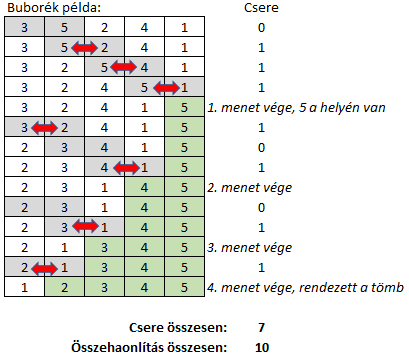
|  |  |  |  |
| --- | --- | --- | --- |
|  | Polinom | Rekurzív | Horner |
| *S(n)* | Θ(*n2*) | Θ(*n*) | Θ(*n*) |
| *Ö(n)* | Θ(*n*) | Θ(*n*) | Θ(*n*) |
| *it(n)* | Θ(*n2*) | Θ(*n*) | Θ(*n*) |

### 2. feladat: buborék rendezés

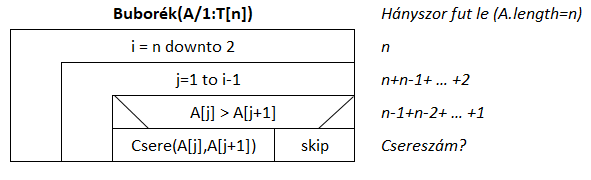
**Rendezési feladat**

* Általánosan megfogalmazva: adott egy rekord sorozat, melyet egy adott kulcs szerint rendezni szeretnénk. A kulcsokon értelmezett egy teljes rendezési reláció. Rendezhetünk növekvőleg, vagy csökkenőleg.
* A rendezési algoritmusok bemutatására elegendő, ha a kulcsokat egész számokkal modellezzük, és alapértelmezésként kulcs szerint növekvő sorrendbe kívánjuk rendezni az adatainkat.
* A rendezendő kulcsok lehetnek egyediek (mind különböző: például hallgatói rekordok Neptun kód szerinti rendezése), vagy lehetnek köztük azonosak (például hallgatói rekordok születési évszám szerinti rendezése).
* Elsőként olyan rendezésekkel foglalkozunk, melyek a kulcsok összehasonlításán alapulnak (összehasonlító rendezések).
* A rendezéseket sokféleképpen osztályozhatjuk, leggyakrabban a futási időt és a tárigényt vizsgáljuk.
* Futási idő szerint vannak **lassú rendezők:**
  + Buborék-, maximum kiválasztó-, beszúró rendezés.
* **gyors rendezők:**
  + Összefésülő rendezés, gyorsrendezés, kupacrendezés.
* Tárigény szerint: lehetnek helyben / nem helyben rendezők.

**Buborék rendezés működése:**



A rendezendő kulcsokat (és a hozzájuk tartozó adatokat) egy A nevű tömbben helyeztük el. A.length = n, a rendezendő kulcsok darabszáma, a tömböt 1-től indexxeljük.



Az összehasonlítások száma

Cserék számát hogyan tudjuk meghatározni?

* Cserék száma a rendezendő adatsorban található inverziók számával egyenlő.   
  Lásd a példában: [3,5,2,4,1]   
  7 inverzió van: 3,2 3,1 5,2 5,4 5,1 2,1 4,1
* mCs(n)=0 (nincs inverzió, azaz növekvően rendezett a bemenet)
* MCs(n)= Ö(n) (minden összehasonlítást csere követ, azaz fordítottan rendezett a tömb)
* ACs(n)=

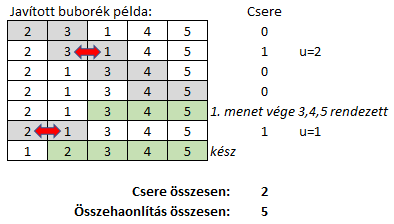
mCs(n)=0, MCs(n)=Θ(n2) azaz Cs(n)=Ο(n2)

Az átlagos futási idő kiszámítása részletesen megtalálható dr Fekete István jegyzetében: <https://people.inf.elte.hu/fekete/algoritmusok_jegyzet/01_fejezet_Muveletigeny.pdf>

Buborék rendezés javítási módszerei:

* figyelhetjük egy logikai változóval, hogy volt-e csere, ha nem volt akkor a külső ciklus álljon le,
* megjegyezhetjük az utolsó csere helyét: ha ez u és u+1 indexen történt, akkor u+1-től már a tömb rendezett, a külső ciklus változót u-ra lehet csökkenteni.

Példa:



Javított Buborék rendezés struktogramja:

|  |  |  |  |
| --- | --- | --- | --- |
| **JavítottBuborék(A/1:T[n])** | | | |
| i:=n | | | |
| i>1 | | | |
|  | u:=-1 | | |
| j=1 to i-1 | | |
|  | A[j] > A[j+1] | |
| csere(A[j], A[j+1]) | skip |
| u:=j |
| i:=u | | |

Adjuk meg a nagyságrendjét az alábbi jellemzőknek:

mÖ(n)∈

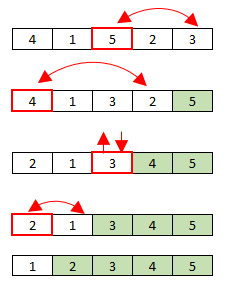
MÖ(n)∈

mÖ(n),MÖ(n) ∈ Ω(?)

mÖ(n),MÖ(n) ∈ Ο(?)

### 3. feladat: maximum kiválasztásos rendezés

Példa:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MaxkivRendezés(A/1:T[n])** | | | | Végrehajtások száma |
| i=n downto 2 | | | |  |
|  | int:=1 | | | n-1 |
|  | j=2 to i | | | n+n-1+..+2 |
|  |  | A[j]>A[ind] | | n-1 + n-2 +.. +1 |
|  |  | ind:=j | skip |  |
|  | csere(A[ind],A[i]) | | | n-1 |

Ö(n) =

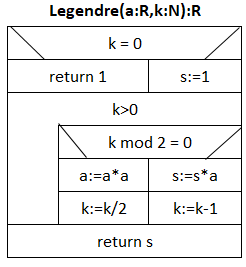
Cs(n) =

### 4. feladat: mit mondhatunk a tanult programozási tételek műveletigényéről? Jellemezzük a tanult aszimptotikus korlátok segítségével!

|  |  |  |
| --- | --- | --- |
|  | mT(n) | MT(n) |
| Összegzés, számlálás | Θ(*n*) | Θ(*n*) |
| Maximum kiválasztás | Θ(*n*) | Θ(*n*) |
| Keresés (eldöntés) | Θ(1) | Θ(*n*) |

### 5. feladat: önállóan készítsük el a minimum kiválasztáson alapuló rendező algoritmust, és töltsük fel a CANVAS-ba az első gyakorlathoz.

### Szorgalmi házi feladatok:

1. Legendre algoritmus műveletigényének meghatározása *k* függvényében:  
   

Útmutatás:

* mutassuk be egy konkrét esetben a működést (például a=3, k=11)
  + 1.iteráció: s=3, k=10, a=3
  + 2.i: s=3, k=5, a=9
  + 3.i: s=27, k=4, a=9
  + 4.i: s=27, k=2, a=81
  + 5.i: s=27, k=1, a=6561
  + 6.i: s=177147, k=0, a=6561
* vizsgáljuk meg k>0 esetben mi a lehető legkedvezőbb végrehajtás szám
  + ha k 2-nek a hatványa akkor a végrehajtási szám
* vizsgáljuk meg k>0 esetben mi a lehető legkedvezőtlenebb végrehajtás szám
  + k-1

1. Adott egy *n* hosszú, egész számokat tartalmazó tömb. Keressük a tömb azon szakaszát, melynek összege a lehető legnagyobb. (Legyen a tömb neve: A, adjuk meg az a két indexet: 1≤ind1≤ind2≤n, melyre a a maximális.). Elemezzük a megoldásunk műveletigényét, készítsünk minél hatékonyabb algoritmust!

Útmutatás: a „Brute-Force” megoldás Θ(n3), könnyen javítható Θ(n2)-re, de van Θ(n)-es megoldás is!