

Algoritmusok és Adatszerkezetek I.

Bevezetés, algoritmusok elemzése

2018. szeptember 4.



Dr. Farkas Richárd

SzTE TTIK, Számítógépes Algoritmusok és
Mesterséges Intelligencia tanszék

rfarkas@inf.u-szeged.hu



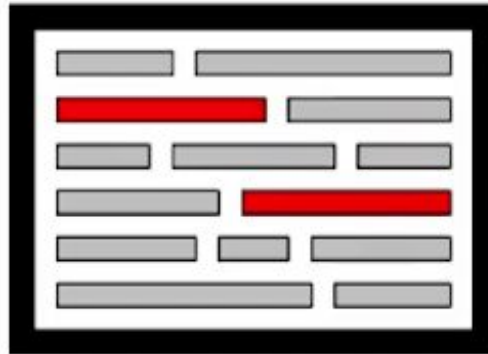
Algoritmusok

Algoritmusnak nevezünk bármilyen ***jól definiált számítási eljárást***, amely ***bemenetként*** bizonyos értéket vagy értékeket kap és ***kimenetként*** bizonyos értéket vagy értékeket állít elő.



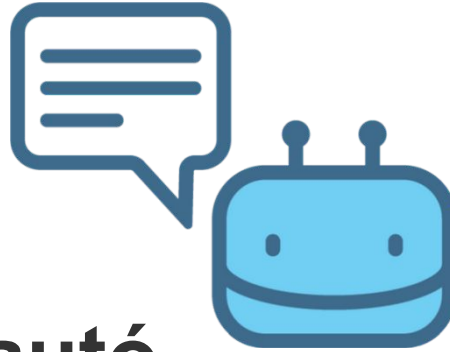
Algoritmus?

- Jeleníts meg egy képet a weblapon
 - túl triviális, nem érdekes itt...
- Egy adott szó szerepel-e egy fájlban
 - Ha sebesség fontos, okos megoldás kell!



Algoritmus?

- Chatrobot
- Önvezető autó

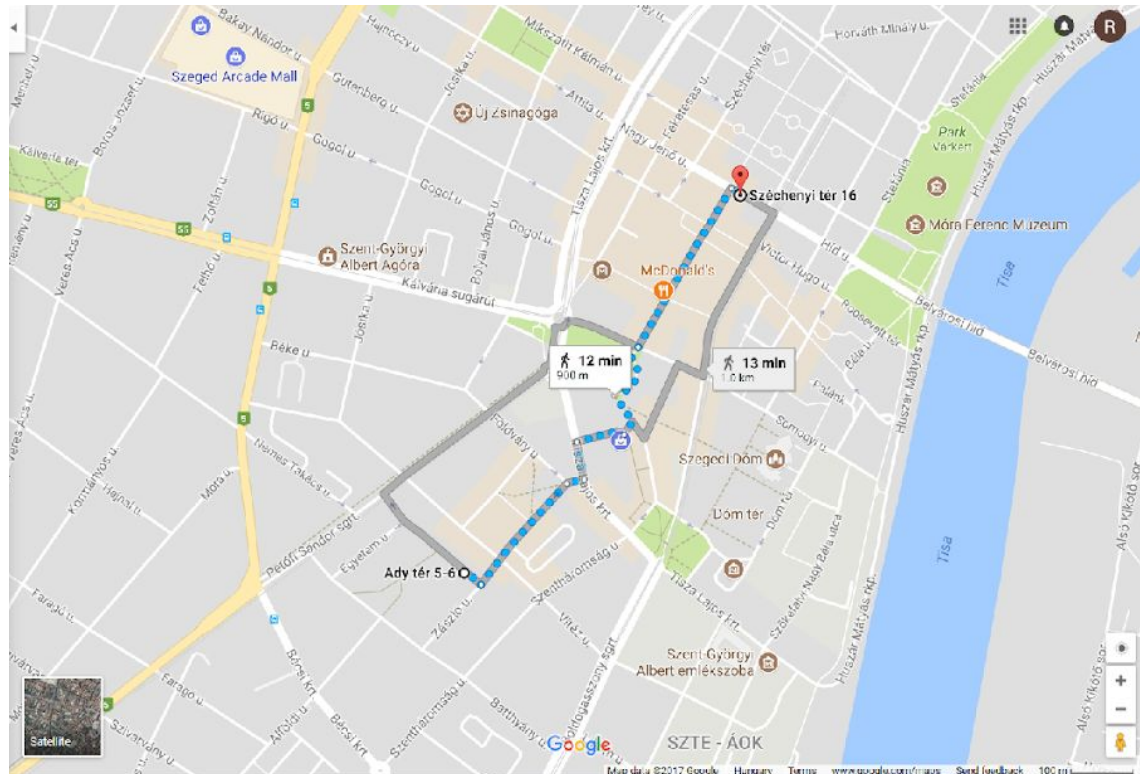


- Nem jól definiált!
- Mesterséges Intelligencia



Algoritmus!

- Legrövidebb út keresése



- Nem triviális a megoldás
- Egyszerű megoldás túl lassú

Adatszerkezetek

Az ***adatszerkezet*** adatok tárolására és szervezésére szolgáló módszer, amely lehetővé teszi a hozzáférést és módosításokat



egfelelő algoritmushoz
egfelelő adatszerkezetet!

Miért tanuljak algoritmusokat?

- Mindenki fogja használni!
- BigData – skálázódás fontos!



Miért tanuljak algoritmusokat?

- **Algoritmikus gondolkodás!**
 - Algoritmus eddig megoldatlan problémára?
 - Megfelelő algoritmusok és adatszerkezetek kiválasztása
 - Gondoljuk végig a helyességet és hatékonyságot!



Miért tanuljak algoritmusokat?

- Nyelvfüggetlen programozói szemlélet

“

You can practice really hard for two years to become a great programmer and you can practice for 10 years to become an excellent programmer. Or you can practice for two years and take an algorithms course and become an excellent programmer

- Absztraktabb gondolkodás

[How to: Work at Google — Example Coding/Engineering Interview](#)



Követelmények

Előadás:

- írásbeli kollokvium
- 7 kérdés, megértés a cél!

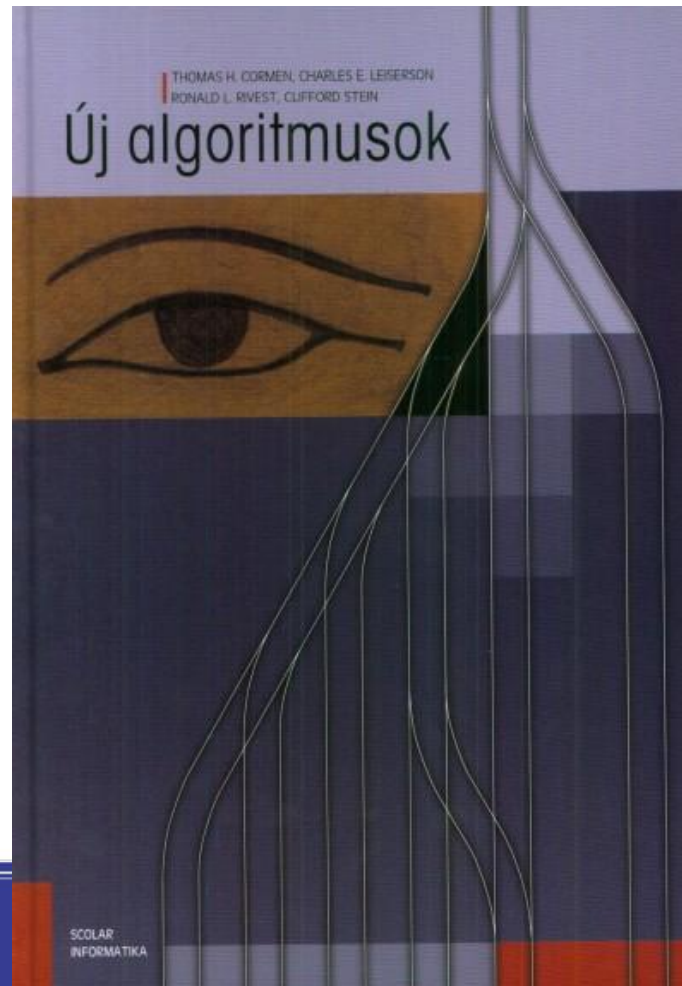
Gyakorlat:

????



Anyagok

<http://www.inf.u-szeged.hu/~rfarkas>



Algoritmusok tervezése

- Értsük meg mélyen a feladatot!
- Nincs általános módszertan algoritmizálásra
- A félév folyamán
 - megismerünk hasznos technikákat
 - látunk számtalan algoritmust különböző problémákraezek mintául szolgálhatnak a jövőben.



Algoritmusok elemzése

- Helyesség
- Hatékonyság:
 - előre megmondjuk, milyen erőforrásokra lesz szüksége az algoritmusnak
 - ***számítási idő***, memória, sávszélesség
- Cél: algoritmusok összehasonlítása



Futási idő

- Milyen hardver?
- CPU? GPU? Felhő?
- ***Futási idő***: egy bizonyos bemenetre a végrehajtott (gépfüggetlen) alpműveletek vagy "lépések" száma
- Feltesszük, hogy egy kód mindegyik sorának végrehajtásához konstans mennyiségű idő szükséges

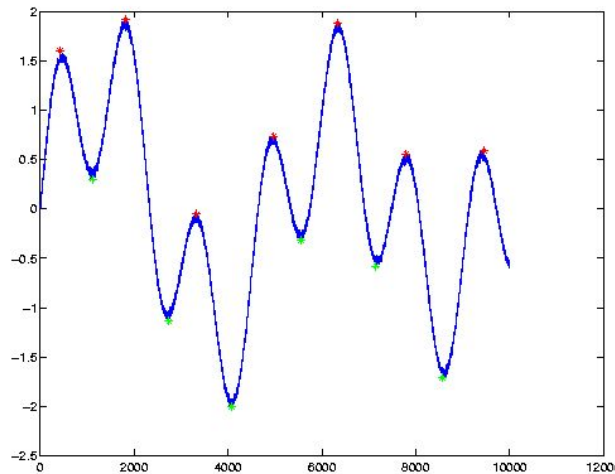


Feladat: csúcs keresés

Bemenet: egész számok tömbje

Találjunk és adjunk vissza **egy** „csúcs”ot ha létezik!

Csúcs: olyan elem a tömbben ami minden szomszédjánál nem kisebb



Forrás: [MIT Introduction to Algorithms, Lecture 1.](#)

Feladat: csúcs keresés

1	3	4	3	5	1	3
---	---	---	---	---	---	---

Csúcs: olyan elem a tömbben ami minden szomszédjánál nem kisebb

Létezik mindig csúcs?

„nem kisebb” helyett „nagyobb” egy másik algoritmust igényelhet!



Egyszerű csúcs kereső alg

- Balról jobbra vizsgáljuk meg minden elemet és ha csúcsot találunk térjünk vissza

1	3	4	3	5	1	3
---	---	---	---	---	---	---

- Pszeudokód:
for $j \leftarrow 1$ **to** $\text{hossz}[A]$
 do if $A[j] \geq A[j-1]$ és $A[j] \geq A[j+1]$
 return j

Helyes?



Egyszerű csúcs kereső alg

```
if  $hossz[A] < 1$ 
  return nil
if  $hossz[A] = 1$ 
  return 1
else if  $A[1] \geq A[2]$ 
  return 1
else if  $A[hossz[A]] \geq A[hossz[A]-1]$ 
  return  $hossz[A]$ 
for  $j \leftarrow 2$  to  $hossz[A]-1$ 
  do if  $A[j] \geq A[j-1]$  és  $A[j] \geq A[j+1]$ 
    return  $j$ 
```



Egyszerű csúcs kereső alg

```
public static Integer find_a_peak(int[] a) throws IllegalArgumentException {  
    //határesetek kezelése:  
    if(a == null)  
        throw new IllegalArgumentException();  
    if(a.length < 1)  
        return null;  
    if(a.length == 1)  
        return 0;  
    if(a[0] >= a[1])  
        return 0;  
    if(a[a.length-1] >= a[a.length-2])  
        return a.length-1;  
  
    //algoritmus:  
    for(int j = 2 ; j < a.length-1; ++j)  
        if((a[j] >= a[j-1]) && (a[j] >= a[j+1]))  
            return j;  
  
    //soha nem érhetünk ide:  
    return null;  
}
```



Algoritmusok (és implementáció) helyességének tesztelése

- **Tesztesetek (unit test)**
- **Először elvárt bemenetekre**

```
find_a_peak(new int[]{1,3,4,3,5,1,3});
```

- **Aztán határesetekre is!**

```
find_a_peak(new int[]{7,3,4,3,5,1,3});
```

```
find_a_peak(new int[]{1,3,4,3,5,1,5});
```

```
find_a_peak(new int[]{1,3});
```

```
find_a_peak(new int[]{1});
```

```
find_a_peak(new int[]{});
```

```
find_a_peak(null);
```



Egyszerű csúcs kereső algoritmus elemzése

költség végrehajtási szám

if $hossz[A] < 1$	c_1	1
return nil	c_2	1
if $hossz[A] = 1$	c_3	1
return 1	c_4	1
else if $A[1] \geq A[2]$	c_5	1
return 1	c_6	1
else if $A[hossz[A]] \geq A[hossz[A]-1]$	c_7	1
return $hossz[A]$	c_8	1
for $j \leftarrow 2$ to $hossz[A]-1$	c_9	$n-2$
do if $A[j] \geq A[j-1]$ és $A[j] \geq A[j+1]$	c_{10}	$n-2$
return j	c_{11}	1



Legjobb, átlagos esetek elemzése

- Bemenet mérete konstans n
- Algoritmus futás idejét (utasítások száma) $T(n)$ -el jelöljük
- Legjobb esetben az első elem csúcs, ekkor $T(n)=2$ minden n -re
- Az átlagos vagy várható futásidőt nagyon nehéz kiszámolni...
(valószínűségi elemzés)



Legrosszabb eset elemzése

- Inkább legyünk pesszimisták!
- Az algoritmus legrosszabb futási ideje bármilyen bemenetre a futási idő felső korlátja.
 - Gyakran előfordul a legrosszabb eset
 - Az átlagos eset gyakran nagyjából ugyanolyan rossz, mint a legrosszabb eset.



Egyszerű csúcs kereső algoritmus elemzése

költség végrehajtási szám

if <i>hossz</i> [A] < 1	c_1	1
return nil	c_2	1
if <i>hossz</i> [A] = 1	c_3	1
return 1	c_4	1
else if A[1] ≥ A[2]	c_5	1
return 1	c_6	1
else if A[<i>hossz</i> [A]] ≥ A[<i>hossz</i> [A]-1]	c_7	1
return <i>hossz</i> [A]	c_8	1
for <i>j</i> ← 2 to <i>hossz</i> [A]-1	c_9	n-2
do if A[<i>j</i>] ≥ A[<i>j</i> -1] és A[<i>j</i>] ≥ A[<i>j</i> +1]	c_{10}	n-2
return <i>j</i>	c_{11}	1

Legrosszabb esetben:

$$T(n) = c_1 + c_3 + c_5 + c_7 + c_9(n-2) + c_{10}(n-2) + c_{11}$$



Egyszerű csúcs kereső algoritmus elemzése

Legrosszabb esetben:

$$T(n) = c_1 + c_3 + c_5 + c_7 + c_9(n-2) + c_{10}(n-2) + c_{11}$$

$$T(n) = (c_9 + c_{10})(n-2) + c_1 + c_3 + c_5 + c_7 + c_{11}$$

- Tényleges futásidők helyett c konstansok
- Nagy n esetén $c_1 + c_3 + c_5 + c_7 + c_{11}$ elhanyagolható
- Hatékonyság szempontjából minket az érdekel, hogy hogyan skálázódik az algoritmus, azaz milyen a ***növekedési sebessége***
- Elég csak a fő tagot figyelembe venni: $(c_9 + c_{10})(n-2)$
- Nagyságrendileg $T(n)$ n lineáris függvénye



Algoritmusok stressz tesztelése

```
public static void running_time(){  
    int size = 100000000;  
    int[] ints = new int[size];  
    ints[size-1] = 0;  
    for(int i=0;i<size-1;++i)  
        ints[i]=i;  
    long time = System.currentTimeMillis();  
    find_a_peak(ints);  
    System.out.println((System.currentTimeMillis()-time) +  
        " msec");  
}
```

stdout 10M: 9 msec
stdout 100M: 78 msec



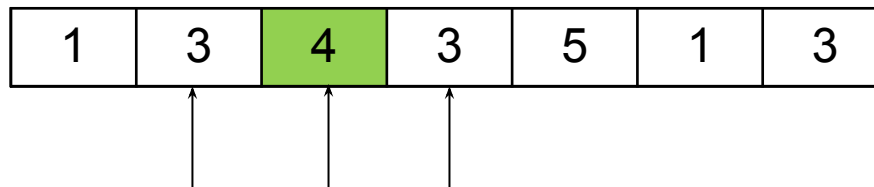
Feladat: csúcs keresés

- Feladat ugyanaz!
- Tudunk hatékonyabb megoldást találni?
- Gondoltam egy számra 1 és 2^{32} közt
- **Oszd meg és uralkodj!**

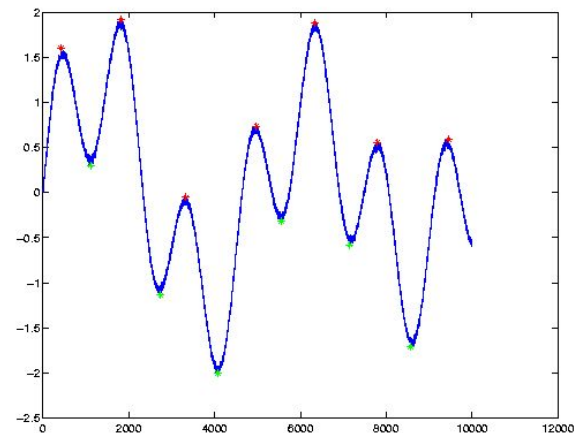


Felező csúcskereső algoritmus

Vizsgáljuk meg a középső elemet. Ha nem csúcs akkor egyik szomszéd nagyobb, vizsgáljuk a bemenet felét a ezen szomszéd irányába!



Helyes?



Felező csúcskereső algoritmus

```
public static Integer find_a_peak(int[] a){
    // határesetek kezelése ugyanaz, mint a lassú verzióban!

    // algoritmus:
    int lo = 1;
    int hi = a.length - 2;
    while (lo <= hi) {
        //kell lennie csucsnek az a[lo..hi]-ban
        int mid = lo + (hi - lo) / 2;
        if (a[mid] < a[mid - 1])
            hi = mid - 1;
        else if (a[mid] < a[mid + 1])
            lo = mid + 1;
        else
            return mid;
    }

    // Soha nem érhetünk ide:
    return null;
}
```



Felező csúcskereső algoritmus futási ideje

$$T(n) = T(n/2) + c$$

$$T(1) = c$$

$$T(16) = T(8) + c = T(4) + c + c = T(2) + c + c + c = T(1) + c + c + c + c = c + c + c + c + c = 5c$$

$$T(n) = (\log_2 n + 1) c$$

Legrosszabb esetben is $T(n)$ n függvényében
logaritmikus növekedési sebességű

Megj: $\log_a b = \log_c b / \log_c a$ miatt a logaritmus alapja nem számít, hiszen az „csak” konstans szorzó. Nem írjuk ki a kurzuson, hanem mindig 2-es alapú logaritmussal számolunk.

stdout 100M: 0 msec



Feladat: 2D csúcs keresés

Bemenet: egész számok két dimenziós tömbje (mátrix)

Találjunk és adjunk vissza **egy** „csúcs”ot ha létezik!

Csúcs: olyan elem a 2D tömbben ami minden szomszédjánál nem kisebb





Feladat: 2D csúcs keresés

Értsük meg a feladatot!

0	0	9	0	0	0	0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	2	0	0	0	0	0
0	3	0	0	0	0	0
0	5	0	0	0	0	0
0	4	7	0	0	0	0

méret: $n \times n$

Mohó hegymászó 2D csúcskereső

Induljunk valahonnan (középről vagy egyik sarokból), minden lépésben lépünk az egyik nagyobb szomszédra. Ha nincs nagyobb szomszéd csúcsot találtunk.



Minden n mérethez kezdőponthoz és lépési startégiához lehet adni olyan bemenetet amire az egész mátrixot be fogja járni (legrosszabb eset) azaz $T(n) n^2$ növekedési sebességű.

Helyes?

2D csúcskeresés 1D csúcskeresésre visszavezetve

Válasszuk a középső oszlopot.
Keressünk 1D csúcsot ebben. A talált
1D csúcs sorában keressünk újra 1D
csúcsot. Ez 2D csúcs lesz.

Legrosszabb esetben is $\log n$
növekedési sebességű, hiszen $\log n$ idő
alatt találunk 1D csúcsot

Helyes?



2D csúcskeresés 1D csúcskeresésre visszavezetve

8	4	1
7	5	3
2	3	1

Egy helyes de nem hatékony
algorithmus ér valamit, nem úgy mint
egy helytelen de hatékony 😊





2D csúcskeresés felezéssel

Válasszuk a középső oszlopot.
Keressünk meg egy maximális elemet
ebben. Ha ennek bal vagy jobb
szomszédja nagyobb, mint az elem
ismételjük meg az eljárást ebben a fél
mátrixban. Ha bal és jobb
szomszédok nem kisebbek 2D
csúcsot találtunk.

Helyes?



2D csúcskeresés felezéssel

0	0	9	0	0	0	0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	2	0	0	0	0	0
0	3	0	0	0	0	0
0	5	0	0	0	0	0
0	4	7	0	0	0	0

Helyes?

2D csúcskeresés felezéssel futási ideje

Ha csak egy oszlop van a maximum elem keresés legrosszabb időben n lépést igényel.

$$T(n, 1) = cn$$

$$T(n, m) = T(n, m/2) + cn$$

$$T(n, m) = \log n \cdot cn$$

Legrosszabb esetben $n \cdot \log n$ növekedési sebességű az algoritmus



Feladat: 2D csúcs keresés

Mohó hegymászó



n^2

Visszavezetés 1D-re



$\log n$

Mátrixfelelés



$n \cdot \log n$



Aszimptotikus hatékonyság

Ha a bemenet mérete elég nagy, akkor az algoritmus futási idejének csak a nagyságrendje lényeges

Theta:

$\Theta(g(n)) = \{f(n) : \text{létezik } c_1, c_2 \text{ és } n_0 \text{ pozitív állandó úgy, hogy}$
 $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ teljesül minden } n \geq n_0 \text{ esetén}\}.$



Theta:

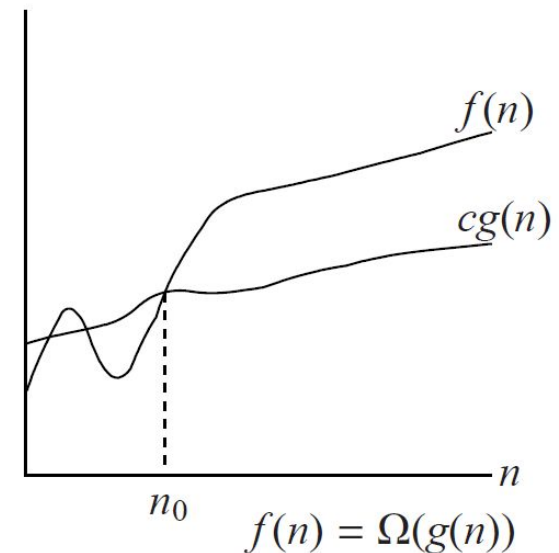
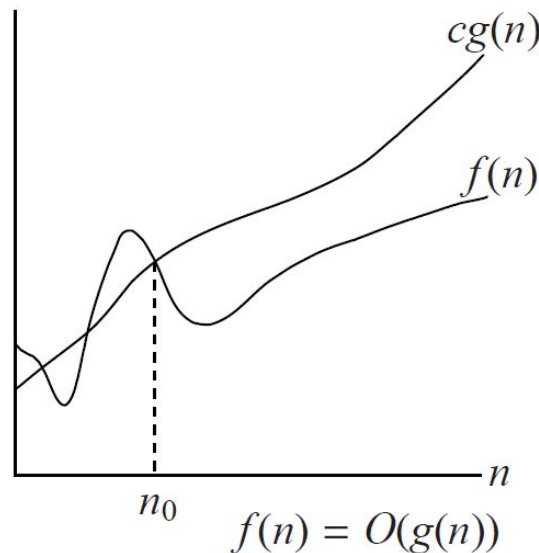
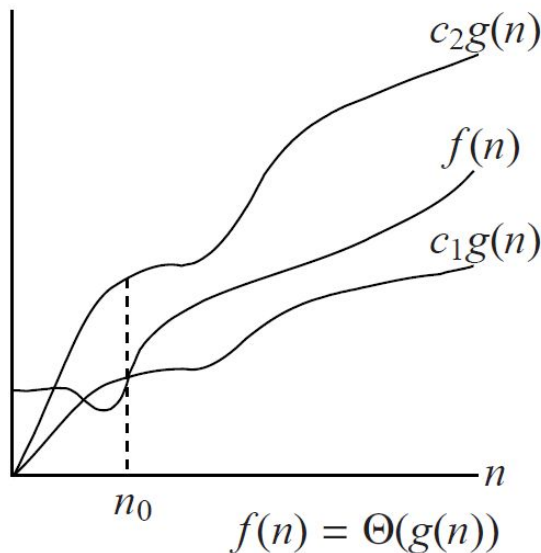
$\Theta(g(n)) = \{f(n) : \text{létezik } c_1, c_2 \text{ és } n_0 \text{ pozitív állandó úgy, hogy}$
 $0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ teljesül minden } n \geq n_0 \text{ esetén}\}.$

Ordó:

$O(g(n)) = \{f(n) : \text{létezik } c \text{ és } n_0 \text{ pozitív állandó úgy, hogy}$
 $0 \leq f(n) \leq cg(n) \text{ teljesül minden } n \geq n_0 \text{ esetén}\}.$

Omega:

$\Omega(g(n)) = \{f(n) : \text{létezik } c \text{ és } n_0 \text{ pozitív konstans úgy, hogy}$
 $0 \leq cg(n) \leq f(n) \text{ teljesül minden } n \geq n_0 \text{ esetén}\}.$



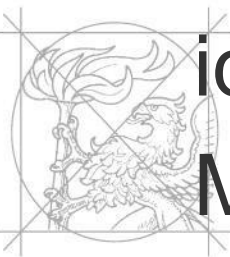
Aszimptotikus felső korlát

Ordó

$O(g(n)) = \{f(n) : \text{létezik } c \text{ és } n_0 \text{ pozitív állandó úgy, hogy}$
 $0 \leq f(n) \leq cg(n) \text{ teljesül minden } n \geq n_0 \text{ esetén}\}.$

Ha nem mondunk mást $O(n)$ azt jelenti,
hogy a vizsgált algoritmus **legrosszabb**
esetben is aszimptotikusan lineáris
időben lefut.

Megj. egy lineáris fgv. is $O(n^2)$ -ben van



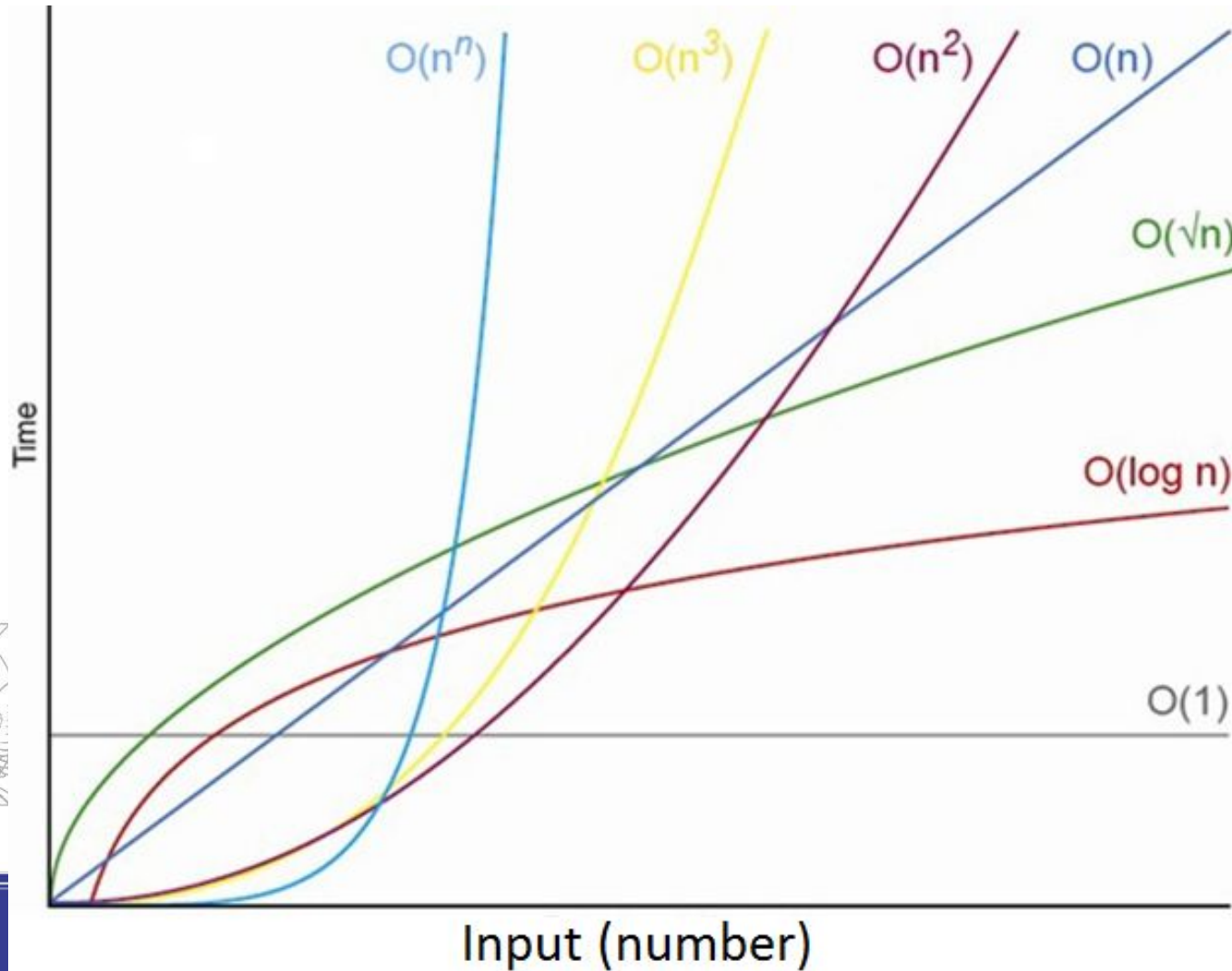
Aszimptotikus felső korlát

$$T(n)=9999n^3 + \sin n + 78n \log n = O(n^3)$$

- Architektúrától független
- Fontos konstans szorzókat elfed!
- Kényelmes használni, de ne feledkezzünk meg róla, hogy ez csak ***aszimptotikus*** korlát!



Tipikus aszimptotikus felső korlátok



Összegzés

- Algoritmusok tervezése
 - Értsük meg a problémát/feladatot
 - Legegyszerűbb megoldást elemezzük
 - Ha kell tervezzünk hatékonyabb algoritmust!
- Algoritmusok elemzése
 - helyesség
 - hatékonyság (skálázódás)
eszköze: legrosszabb eset aszimptotikus felső korlátja

