Algoritmusok Tervezése

6. Előadás

Algoritmusok 101

Dr. Bécsi Tamás

Mi az algoritmus?

- Lépések sorozata egy feladat elvégzéséhez (legáltalánosabban)
- Informálisan ***algoritmusnak*** nevezünk bármilyen jól definiált számítási eljárást, amely ***bemenetként*** bizonyos értéket vagy értékeket kap és ***kimenetként*** bizonyos értéket vagy értékeket állít elő. Eszerint az algoritmus olyan számítási lépések sorozata, amelyek a bemenetet átalakítják kimenetté.

A jó algoritmus

- Egy algoritmust ***helyesnek*** mondunk, ha minden bemenetre megáll és helyes eredményt ad. Azt mondjuk, hogy a helyes algoritmus megoldja az adott számítási feladatot.
- Egy nem helyes algoritmus esetén előfordulhat, hogy nem minden bemenetre áll meg, vagy bizonyos bemenetekre nem a kívánt választ adja. Várakozásunkkal ellentétben egy nemhelyes algoritmus is lehet hasznos, ha hibázási aránya elfogadható.

Algoritmusok megadása

- Egy algoritmus megadható magyar nyelven, számítógépes programként vagy akár hardver segítségével. Az egyetlen követelmény az, hogy a leírás pontosan adja meg a követendő számítási eljárást.
- Szöveges, informális megadás
- Folyamatábra
- Pszeudokód
- Programkód

Az algoritmustervezés tanulása

- Megismerni az algoritmusok korlátait, és lehetőségeit
- Megismerni a tervezési célokat, azok használatát
- Megismerni létező algoritmusokat
- Megismerni alapvető algoritmustervezési módszereket, amelyek mintát ad más feladatok megoldásához

Helyes algoritmus vs. hatékony algoritmus

- Nagyon sok algoritmizálási feladatnak létezik exakt, matematikailag bizonyított **helyes** megoldása.
- Az algoritmus **hatékonysága** úgy írható le, hogy elfogadható erőforrás felhasználásával szolgáltatja a helyes eredményt. (Idő, Tárhely)
- A helyesség és a hatékonyság sokszor egymással ellentétes igényeket fogalmaznak meg.

„Nehéz” feladatok

- Vannak azonban olyan feladatok, amelyeknek a megoldására nem ismerünk hatékony és helyes algoritmust. (Például az NP teljes feladatok)
- Miért érdekes? Azért, mert bár NP-teljes feladat meg oldására még soha senki nem talált hatékony algoritmust, soha senki nem bizonyította be, hogy ilyen algoritmus nem létezik.
- Ha meg lehet mutatni, hogy a feladat NP-teljes, akkor a tervező arra fordíthatja az idejét, hogy olyan hatékony algoritmust tervezzen, amely jó – bár nem a lehető legjobb – eredményt ad.

Algoritmusok futásideje

- Az algoritmus futásideje függ az N bemenő paramétertől. Azonos feladat különböző N értékek esetén más futásidőt igényelnek. Például:
  - Lineáris keresés esetén az N növekedésével egyenes arányosan nő a futásidő
  - Bináris keresés esetén N növekedésével logaritmikusan nő a futásigény
  - Egyszerű (mondjuk buborék) sorbarendezés esetén N-nel négyzetesen arányosan nő a futásidő

Függvények növekedése

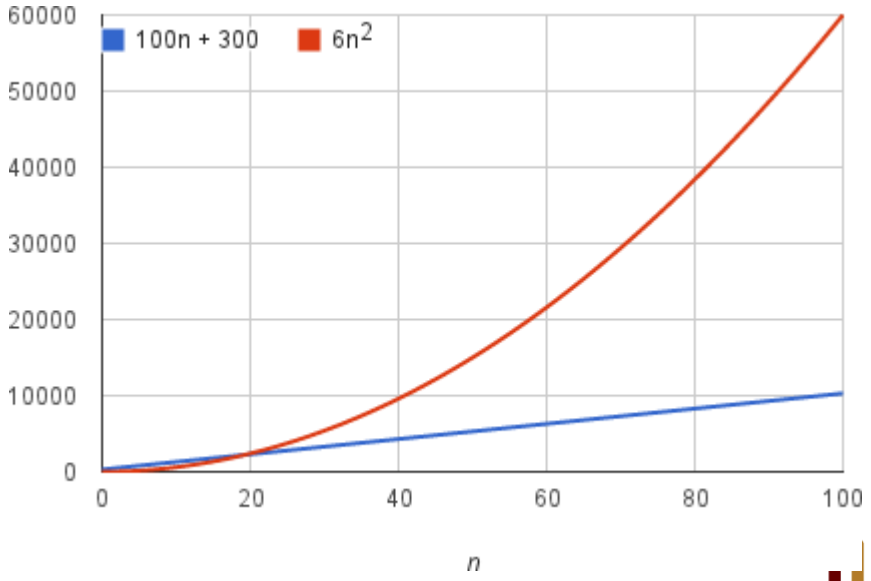
- Ha a bemenet mérete elég nagy, akkor az algoritmus futási idejének csak a nagyságrendje lényeges, ezért az algoritmusnak az ***aszimptotikus*** hatékonyságát vizsgáljuk.
- Ekkor csak azzal foglalkozunk, hogy a bemenet növekedésével miként növekszik az algoritmus futási ideje *határértékben,* ha a bemenet mérete minden határon túl nő. Általában az aszimptotikusan hatékonyabb algoritmus lesz a legjobb választás, kivéve a kis bemenetek esetét.

Aszimptotikus jelölések

- Egy algoritmus aszimptotikus futási idejét leíró jelöléseket olyan függvényekként definiáljuk, melyeknek értelmezési tartománya a természetes számok N = {0, 1, 2, . . .} halmaza.
- Ilyen jelölésekkel kényelmesen leírható a futási idő a legrosszabb esetben, azaz a *T*(N) függvény, amely általában csak egész számú bemenő adattól függ. Mindemellett néha kényelmes az aszimptotikus jelölések többféle *pontatlan* használata.

Példa

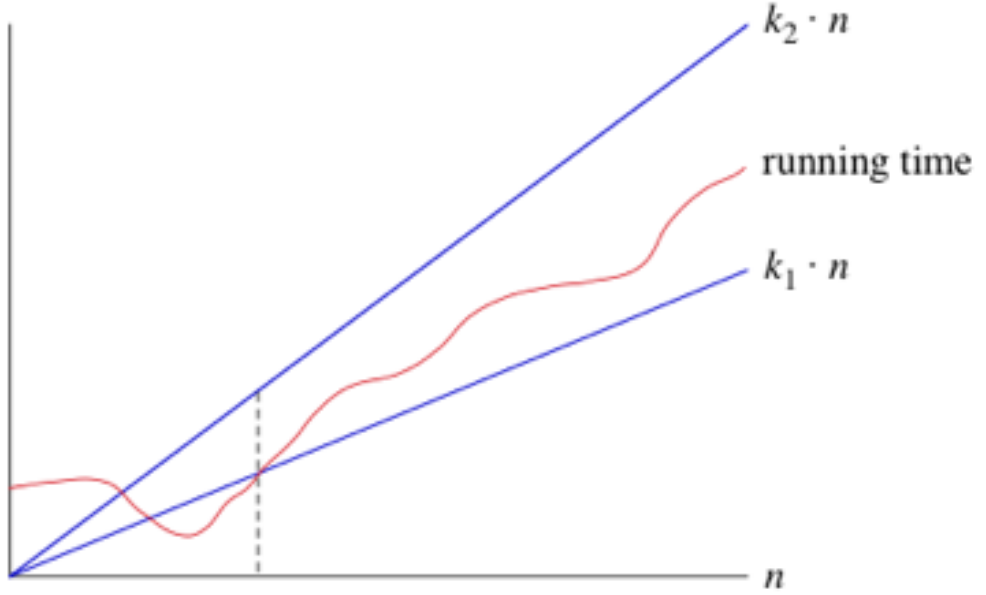
Tételezzük fel, hogy az algoritmusunk futásideje meghatározható, és n darabszám függvényében $T(n) = 6n^2 + 100n + 300$ utasítást igényel. Látható, hogy n>20 esetén a $6n^2$ tag jóval nagyobb, és gyorsabban nő, mint a másik tag. Ekkor az mondjuk, hogy hogy az algoritmus négyzetesen nő. Ez a viselkedés az együtthatóktól független, mindig lesz olyan határszám, ahonnan a négyzetes tag a meghatározó.



Θ jelölés

Definíció: Egy adott g(n) esetén $\Theta(g(n))$-nel jelöljük a függvényeknek azt a halmazát, amelyre:

$\Theta(g(n))$ ={f(n), ha létezik olyan k_1, k_2 és n_0 pozitív állandó, hogy $0 \leq k_1 g(n) \leq f(n) \leq k_2 g(n)$, minden $n > n_0$ esetén}



O jelölés

Definíció: Egy adott g(n) esetén $O(g(n))$-nel jelöljük a függvényeknek azt a halmazát, amelyre:

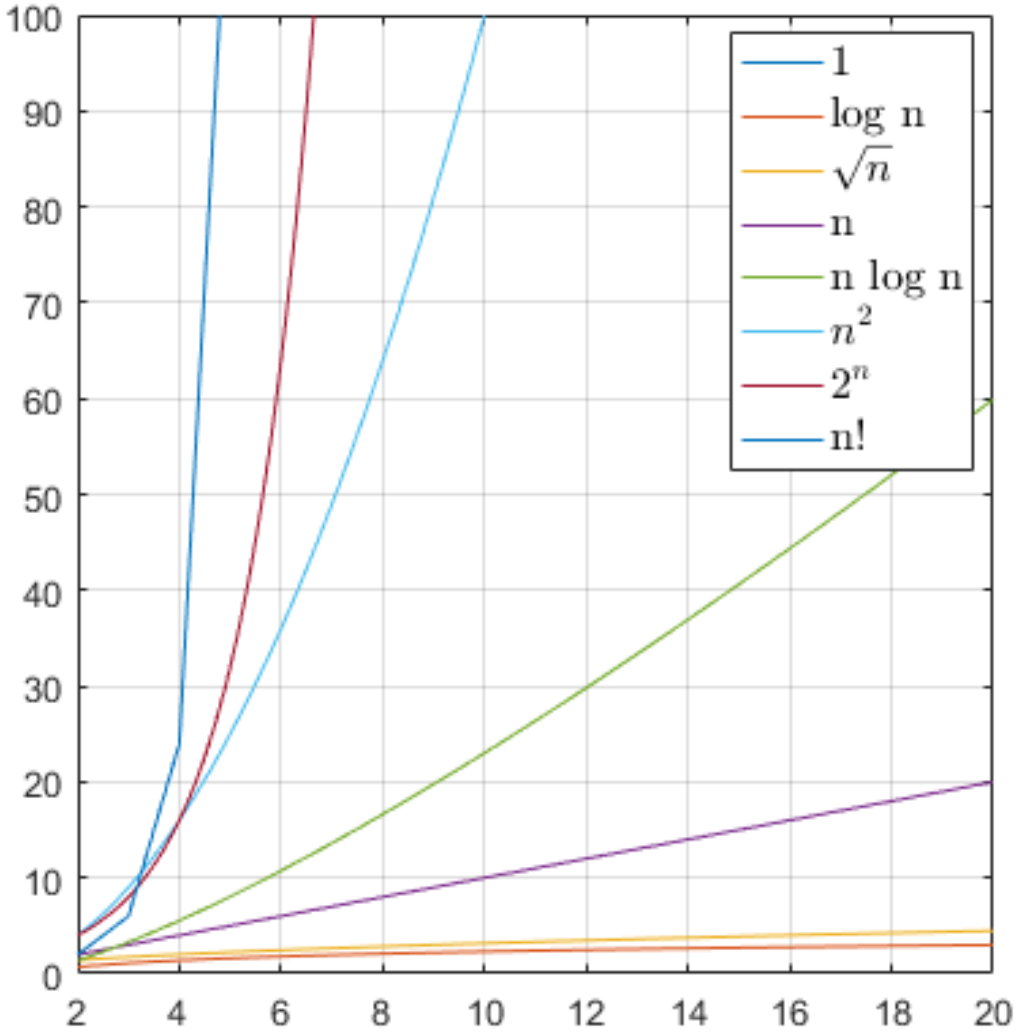
$O(g(n))$ ={f(n), ha létezik olyan k $és$ n_0 pozitív állandó, hogy $0 \leq f(n) \leq kg(n)$, minden $n > n_0$ esetén}

Felső korlát

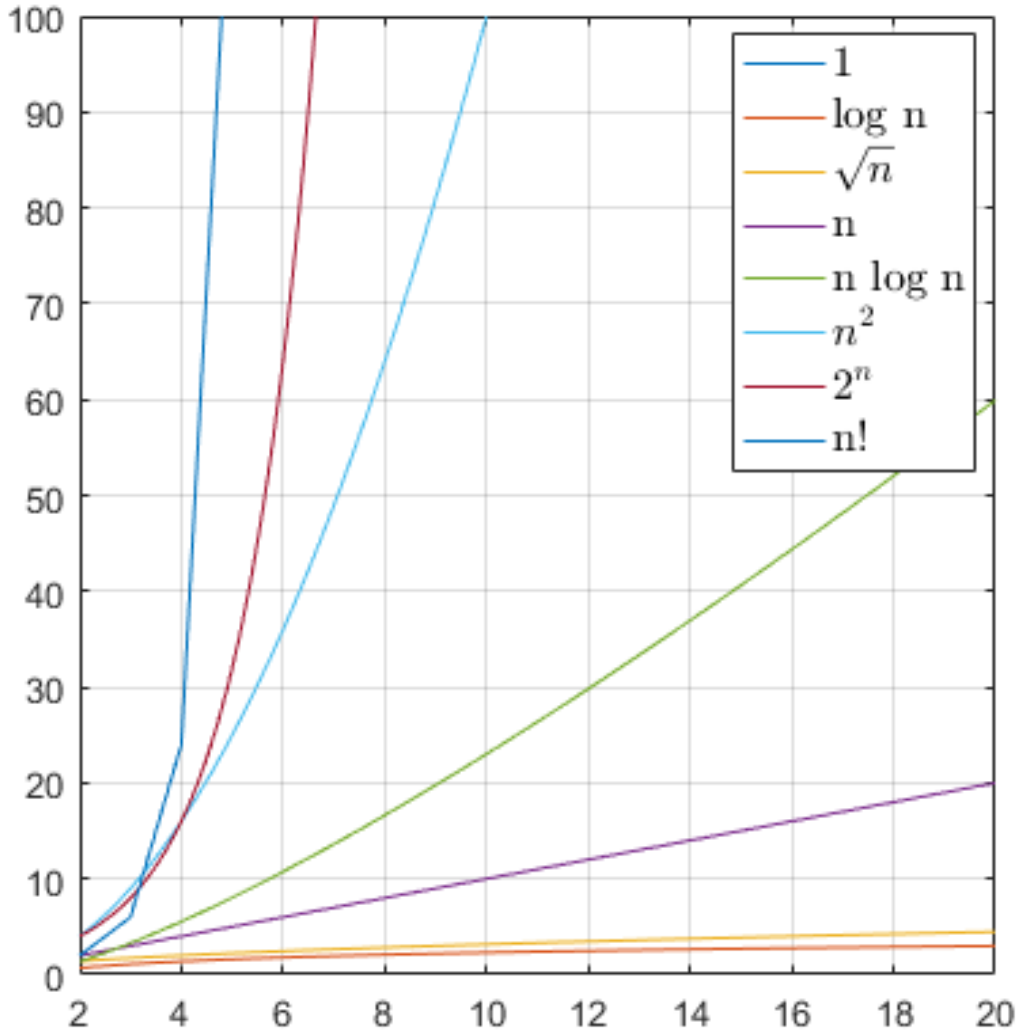
- Értelemszerűen:
  - ha $f(n) = \Theta(g(n))$, akkor $f(n) = O(g(n))$

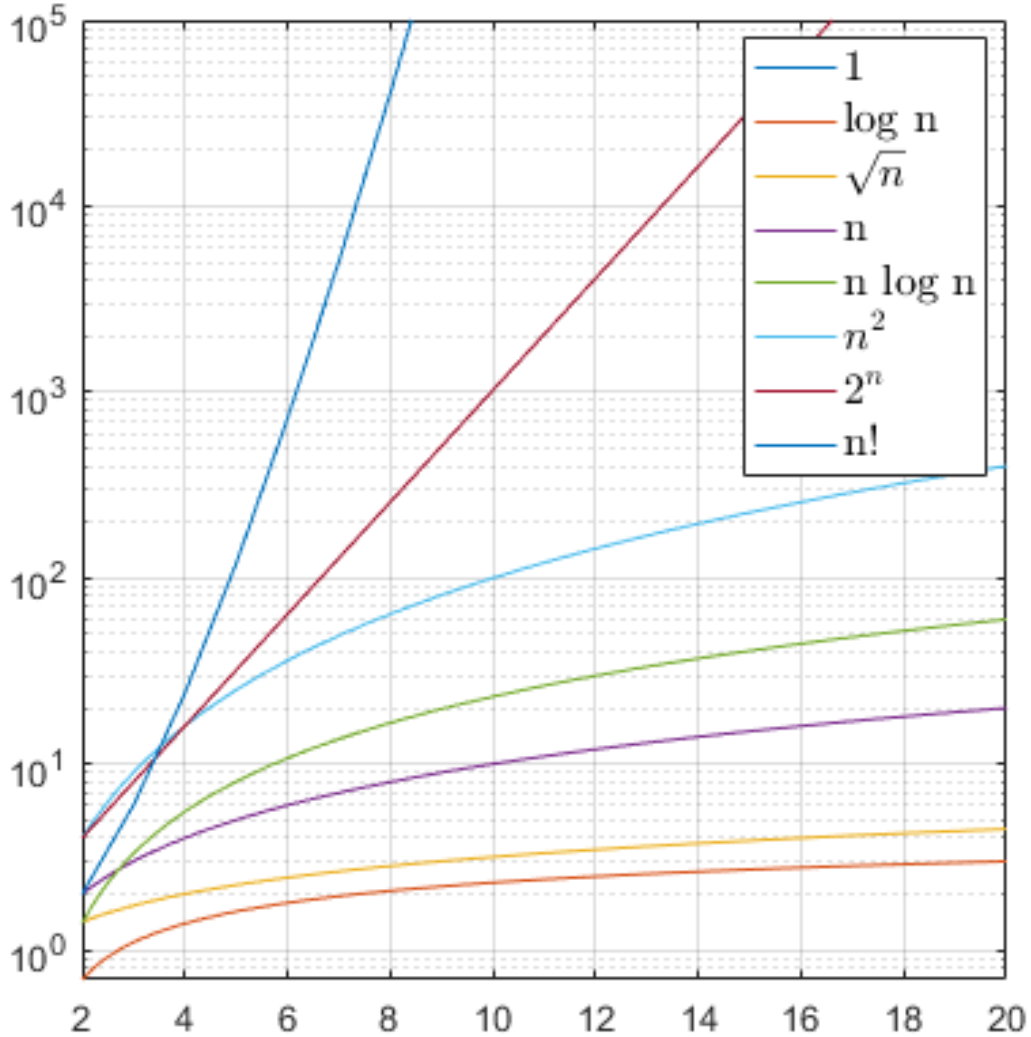
Nagyságrendek, példák

Komp.	Elnevezés	Példa
Θ(1)	konstans	Párosság eldöntése
Θ(lg n)	logaritmikus	bináris keresés
Θ($\sqrt{n}$)	gyökös	példa gyakorlatról
Θ(n)	lineáris	lineáris keresés
Θ(n log n)		mergesort, quicksort, heapsort
Θ(n^2)	négyzetes	sorbarendezés „legrosszabb” esetben
Θ(n^3)	köbös	n*n mátrixszorzás
Θ(n^c)	polinomiális	
Θ(c^n)	exponenciális	TSP exakt megoldás dinamikus programozással
Θ(n!)	faktoriális	TSP exakt megoldás brute force algoritmussal



Nagyságrendek





Tulajdonságok

- $f_1 \in O(g_1)$ és $f_2 \in O(g_2) \Rightarrow f_1 f_2 \in O(g_1 g_2)$
- $fO(g) \subset O(fg)$

- $f_1 \in O(g_1)$ és $f_2 \in O(g_2) \Rightarrow f_1 + f_2 \in O(|g_1| + |g_2|)$
  - ha $f_1 \in O(g)$ és $f_2 \in O(g) \Rightarrow f_1 + f_2 \in O(g)$

- $O(kg) = O(g)$