

I. ALAPFOGALMAK

1. ALGORITMUSOK MŰVELETIGÉNYE	2
1.1. A buborék rendezés műveletigénye.....	2
1.2. A Hanoi tornyai probléma megoldásának műveletigénye	4
1.3. Függvények aszimptotikus viselkedése.....	7

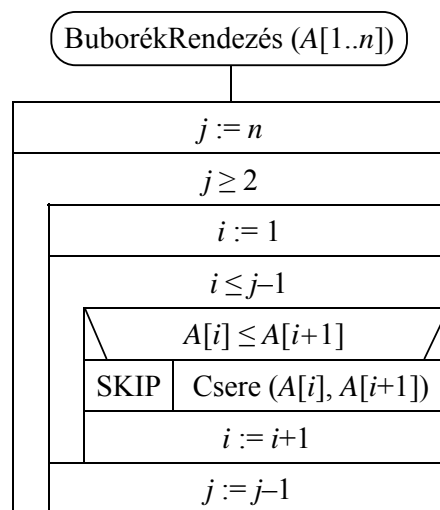
1. ALGORITMUSOK MŰVELETIGÉNYE

Az előadáson ismertetésre kerülő adatszerkezeteket és algoritmusokat mindig jellemezni fogjuk a hatékonyság szempontjából. Az adatszerkezetek egyes ábrázolásairól megállapítjuk a helyfoglalásukat, az algoritmusoknál pedig a műveletigényt becsüljük. Általában megelégszünk mindkét adat nagyságrendben közelítő értékével. A műveletigény számításakor eleve azzal a közelítéssel élünk, hogy csak az algoritmus meghatározó műveleteit vesszük számításba. Egy műveletet meghatározónak (dominánsnak) mondunk, ha a többihez képest jelentős a végrehajtási ideje valamint a végrehajtásainak száma. (Általában kijelölhető egyetlen meghatározó művelet, amelyre a számítást elvégezzük.) A műveletigényt végső soron jól jellemzi a kiszemelt művelet végrehajtásainak száma, hiszen az egyes műveletek végrehajtási ideje gépről-gépre változhat. Így az ilyen módon, több közelítéssel kiszámolt műveletigény (lépésszám) nagyságrendben jól jellemzi az algoritmus tényleges futási idejét.

1.1. A buborékrendezés műveletigénye

Először is fogalmazzuk meg a rendezés általános feladatát! Egy adott $A[1..n]$ tömb elemeit kell (növekvően) rendeznünk, ahol az elemek típusa olyan, hogy közöttük teljes rendezés értelmezhető, pl. egész vagy valós számok, több byte-os szavak, vagy olyan rekordok, amelyeket egy (vagy esetleg több) meghatározott mezőjük alapján kell sorba rendezni. Tehát nem csak a tömb mérete lehet nagy, hanem maguk az elemei is.

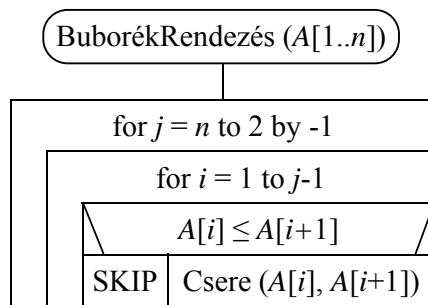
A feladat egyik – közismert és kis hatékonyságú – megoldása a *buborékrendezés*. Az eljárás működésének alapja a szomszédos elemek cseréje, amennyiben az elől álló nagyobb, mint az utána következő. Az első menetben a szomszédos elemek cseréjével "felbuborékolatjuk" a legnagyobb elemet a tömb végére. A következő menetben ugyanezt tesszük az eggyel rövidebb tömbbel, és így tovább. Utoljára még a tömb első két elemét is megcseréljük, ha szükséges. Az $(n-1)$. menet végére elérjük, hogy az elemek nagyság szerint növekvő sorban követik egymást. A rendezés struktogramja az 1.1. ábrán látható.



1. 1. ábra. A buborék rendezés algoritmus

Az egyszerűség kedvéért itt a *Csere* művelet (nyilvánvaló) megvalósítását nem részletezzük.

Az algoritmus műveleteit két kategóriába sorolhatjuk. Az egyikbe tartoznak a két ciklusváltozóra vonatkozó műveletek, a másikba pedig az A tömb szomszédos elemeinek összehasonlítása és cseréje. Nyilvánvaló, hogy az utóbbiak a meghatározó műveletek. Egyrészt végrehajtásuk lényegesen nagyobb időt vesz igénybe, mint a ciklust adminisztráló utasításoké (különösen, ha a tömb elemeinek mérete jelentős), másrészt mindkét utasítás a belső ciklusban van (még ha a csere feltételhez kötött is), így végrehajtásukra minden iterációs lépésben sor kerül(het). Ezért az összehasonlítások, illetve a cserék számát fogjuk számolni. Ha ezt a döntést meghoztuk, érdemes az algoritmusban szereplő ciklusokat tömörebb, áttekinthetőbb formában (for-ciklusként) újra felírni. Ez a változat látható a 1.2. ábrán.



1.2. ábra: A buborék-rendezés (for-ciklusokkal)

A számolást az egyszerűség kedvéért külön-külön végezzük. Nézzük először az összehasonlítások – $\bar{O}(n)$ -nel jelölt – számát. A külső ciklus magja $(n-1)$ -szer hajtódik végre, a belső ciklus ennek megfelelően rendre $n-1, n-2, \dots, 1$ iterációt eredményez. Mivel a két szomszédos elem összehasonlítása a belső ciklusnak egy olyan utasítása, amely mindig végrehajtódik, ezért

$$\bar{O}(n) = (n-1) + (n-2) + \dots + 1 = n(n-1)/2 = n^2/2 - n/2.$$

Az 1.3. pontban precízen bevezetjük majd azokat az aszimptotikus fogalmakat és jelöléseket, amelyekkel a műveletigény nagyságrendjét jellemezzük. Az összehasonlítások számára ennek megfelelően azt mondjuk majd, hogy értéke nagyságrendben n^2 , és ezt így jelöljük:

$$\bar{O}(n) = \Theta(n^2).$$

Megjegyezzük, hogy az összehasonlítások száma a bemenő adatok bármely permutációjára ugyanannyi. Az elemzett műveletek végrehajtási száma a legtöbbször azonban változó az adatok függvényében. Ilyenkor elsősorban a végrehajtások maximális és átlagos számára vagyunk kíváncsiak, de a minimális végrehajtási számot is gyakran meghatározzuk, bár ennek általában nincs jelentősége. Ha $T(n)$ -nel jelöljük a műveletigényt, akkor annak minimális, maximális és átlagos értékét rendre $mT(n)$, $MT(n)$ és $AT(n)$ jelöli.

Vizsgáljuk meg most a cserék – $Cs(n)$ -nel jelölt – számát! Ez a szám már nem állandó, hanem a bemenő adatok függvénye. Nevezetesen igaz az, hogy a cserék száma megegyezik az A tömb elemei között fennálló inverziók számával:

$$Cs(n) = inv(A).$$

Valóban, minden csere pontosan egy inverziót szüntet meg a két szomszédos elem között, újat viszont nem hoz létre. A rendezett tömbben pedig nincs inverzió.

Ha a tömb eleve rendezett, akkor egyetlen cserét sem kell végrehajtani, így a cserék száma a legkedvezőbb esetben nulla, azaz

$$mCs(n) = 0.$$

A legtöbb cserét akkor kell végrehajtani, ha minden szomszédos elempár inverzióban áll, azaz akkor, ha a tömb éppen fordítva, nagyság szerint csökkenő módon rendezett. Ekkor

$$MCs(n) = n(n-1)/2 = \Theta(n^2).$$

A cserék átlagos számának meghatározásához először is feltesszük, hogy a rendezendő számok minden permutációja egyformán valószínű. (Az átlagos műveletigény számításához mindig ismerni kell a bemenő adatok valószínűségi eloszlását.) Az általánosság megszorítása nélkül vehetjük úgy, hogy az 1, 2, ..., n számokat kell rendeznünk, ha elfogadjuk azt a szokásos egyszerűsítést, hogy a rendezendő értékek mind különbözők.

A cserék számának átlagát nyilvánvalóan úgy kapjuk, hogy az 1, 2, ..., n elemek minden permutációjának inverziószámát összeadjuk és osztjuk a permutációk számával:

$$ACs(n) = \frac{1}{n!} \sum_{p \in Perm(n)} inv(p),$$

ahol $Perm(n)$ az n elem összes permutációit jelöli. Az összeg meghatározásánál nehézségbe ütközünk, ha megpróbáljuk azt megmondani, hogy adott n -re hány olyan permutáció van, amelyben i számú inverzió található ($0 \leq i \leq n(n-1)/2$). Ehelyett célszerű párosítani a permutációkat úgy, hogy mindegyikkel párba állítjuk az inverzét, pl. az $p = 1423$ és a $p^R = 3241$ alkot egy ilyen párt. Egy ilyen párban az inverziók száma együtt éppen a lehetséges $\binom{n}{2}$ -t teszi ki, pl. $inv(1423) + inv(3241) = 4 + 2 = 6$. Az állítás igazolására gondoljuk meg, hogy egy permutációban két elem pontosan akkor áll inverzióban, hogy ha az inverz permutációban nincs közöttük inverzió. Írjuk le gondolatban mind az $n!$ permutációt kétszer egymás mellé egy-egy oszlopba, a mondott párosításnak megfelelően. Ekkor minden sorban a két permutáció inverzióinak száma együtt $\binom{n}{2}$, így

$$ACs(n) = \frac{\sum inv(p) + \sum inv(p^R)}{2n!} = \frac{n! \binom{n}{2}}{2n!} = \frac{\binom{n}{2}}{2} = \frac{n(n-1)}{4} = \Theta(n^2).$$

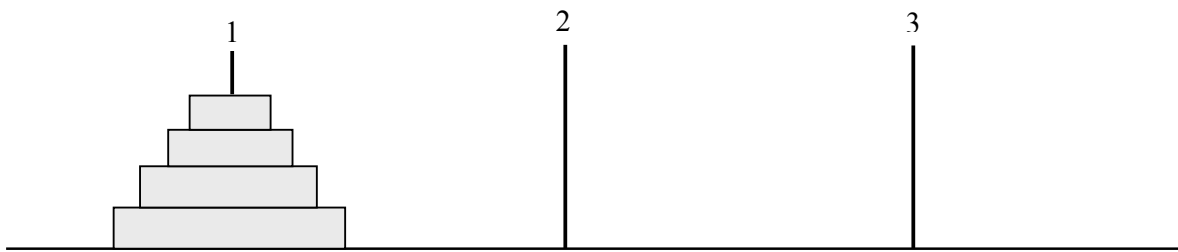
A cserék számának átlaga tehát a legnagyobb érték fele, de nagyságrendben ez így is n^2 -es.

Az elemzés eredménye az, hogy a buborék-rendezés a legrosszabb és az átlagos esetben is – mindkét meghatározó művelete szerint – nagyságrendben n^2 -es algoritmus.

Megjegyzés: A rendezés egy fontos és a gyakorlatban nagyon gyakran felmerülő feladat, ezért a későbbi fejezetek során még visszatérünk rá, több különböző megoldási módszert, algoritmust mutatunk be.

1.2. A Hanoi tornyai probléma megoldásának műveletigénye

A következő algoritmus, amelyet elemzünk, a Hanoi tornyai probléma megoldása. A probléma régről ismert. Adott három rúd és az elsön n darab, felfelé egyre csökkenő méretű korong, úgy, ahogyan a 1.3. ábrán látható $n = 4$ esetére.



1.3. ábra: A Hanoi tornyai probléma kiinduló állapota

A feladat az, hogy a korongokat át kell helyezni az 1-es rúdról a 2-esre a 3-as rúd felhasználásával, oly módon, hogy egyszerre csak egy korongot szabad mozgatni és csak nála nagyobb korongra, vagy üres rúdra lehet áthelyezni.

A feladatnak több különböző megoldása van, köztük olyan (iteratív) heurisztikus algoritmusok, amelyek a mesterséges intelligencia területére tartoznak. Itt a szokásos rekurzív algoritmust ismertetjük.

Számítógépes programok, algoritmusok esetén akkor beszélünk rekurzióról, hogyha az adott eljárás vagy függvény (algoritmus) a működése során „meghívja önmagát”, vagyis a számítás egyik lépéseként önmagát hajtja végre más bemeneti adatokkal, paraméterekkel.

Sok hasznos algoritmus rekurzív szerkezetű, és többnyire az ún. „oszd meg és uralkodj” elv alapján működnek. Ennek a lényege az, hogy a feladatot több részfeladatra bontjuk, amelyek egyenként az eredetihez nagyon hasonlóak, de kisebb méretűek, könnyebben megoldhatók. Ezeket ezután hasonló módon (rekurzívan) további – egyre kisebb és kisebb – részekre osztjuk, majd végül csak a már kellően kicsi részfeladatokat oldjuk meg közvetlen módon. (Ez a lépés gyakran egészen triviális.) Ezután a rekurzív hívási rendszer alapján „visszafelé” haladva minden lépésben összegezzük, összevonjuk az eredményeket, a részfeladatok megoldását. Így az utolsó lépésben megkapjuk az eredeti feladat megoldását.

A módszer elnevezése onnan ered, hogy a rekurzió minden szintjén három alapvető lépést hajt végre:

- felosztja a feladatot több részfeladatra,
- „uralkodik” a részfeladatokon rekurzív módon való megoldásukkal (amennyiben a feladat mérete kellően kicsi, közvetlenül megoldja),
- összevonja a részfeladatok megoldását az eredeti feladat megoldásává.

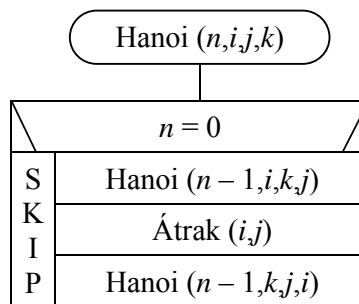
Ennek az általános elvnek megfelelően a Hanoi tornyai probléma rekurzív megoldási módszerének lényege a következő:

1. A felső $n - 1$ korongot helyezzük át az 1-es rúdról a 3-asra a megengedett lépésekkel úgy, hogy a 2-es rúdat vesszük segítségül.
2. Az egyedül maradt alsó korongot tegyük át az 1-es rúdról az üres 2-esre.
3. Vigyük át a 3-as rúdon található $n - 1$ korongot a 2-esre az 1-es rúd igénybe vételével, hasonlóan ahhoz, ahogyan az 1. pontban eljártunk.

Az 1. és 3. lépésben szereplő $n - 1$ korongot hasonló szemlélettel $n - 2$ és 1 korong mozgatására bontjuk, és így tovább.

A rekurzióból való "kijárat" megfogalmazása természetes módon egyetlen korongra adódna: ha $n = 1$, akkor egyszerűen tegyük át a korongot a rendeltetési helyére. Egyszerűbb formájú algoritmust kapunk, és számolni is könnyebb, ha "még lejjebb" adjuk meg a kijáratot a rekurzióból*: ha $n = 0$, akkor nem kell semmit sem tennünk. Erre az ad lehetőséget, hogy így az $n = 1$ eset is kezelhető a fenti három lépéssel: az 1. és a 3. lépés üressé válik, a 2. lépés pedig tartalmazza az elemi áthelyezés műveletét.

Írjuk meg ezután az eljárást! A Hanoi (n, i, j, k) rekurzív eljárás n számú korongot átvisz az i -edik rúdról a j -edikre, a k -adik rúd felhasználásával. Az eljárást "kívülről" a konkrét feladatnak megfelelő paraméterekkel kell meghívni, pl. az ábrán látható esetben így: Hanoi $(4, 1, 2, 3)$. Az eljárás struktogramja a 1.4. ábrán látható. Vegyük észre, hogy a rekurzív eljárás önmagát hívja egészen addig, amíg n nagyobb nullánál. Amikor $n = 1$, akkor is ez történik, de a rekurzív hívások végrehajtása az $n = 0$ ágon már egy-egy üres utasítást eredményez. Az Átrak (i, j) utasítás jelöli azt az elemi műveletet, amellyel egyetlen korongot áteszünk az i -edik rúdról a j -edikre.



1.4.ábra A Hanoi tornyai probléma megoldása

Most kiszámítjuk az n korong átpakolásához szükséges lépésszámot. Az algoritmus rekurzív megfogalmazása szinte kínálja az átrakások számára vonatkozó rekurzív egyenletet:

$$T(n) = \begin{cases} 2T(n-1) + 1, & \text{ha } n \geq 1, \\ 0 & , \text{ ha } n = 0. \end{cases}$$

Ha kiszámítjuk $T(n)$ értékét néhány n -re, akkor azt sejtethetjük, hogy $T(n) = 2^n - 1$. Ezt teljes indukcióval könnyen bebizonyíthatjuk.

Azt kaptuk, hogy a Hanoi tornyai egy exponenciális műveletigényű probléma: $T(n) = \Theta(2^n)$. A legenda szerint Indiában, egy ősi város templomában a szerzetesek ezt a feladatot 64 korongra kezdték el valamikor megoldani azzal, hogy amikor a rakosgatás végére érnek, elkövetkezik a világ vége. Ha minden korongot 1 mp alatt helyeznek át, akkor a 64 korong teljes átpakolása nagyságrendben 600 milliárd évet venne igénybe. Ha 10^{-6} mp-cel számolunk (ami már a számítógépek sebessége), akkor is nem egészen 600 ezer évre lenne szükség a korongok átrendezéséhez.

* Általános szabályként jegyezzük meg azt, hogy a rekurzióból való kijáratot „engedjük minél lejjebb”, azaz a szóban forgó változó minél kisebb értékére próbáljuk azt megvalósítani (pl. fent a kínálkozó $n=1$ helyett $n=0$ -ra). Arra törekszünk tehát, hogy minél kisebb legyen az a részfeladat, amelyet már közvetlenül oldunk meg.

1.3. Függvények aszimptotikus viselkedése

Kiszámítottuk két algoritmus műveletigényét a bemenő adatok számának függvényében. Szeretnénk a jövőben pontos matematikai fogalmakra támaszkodva beszélni az algoritmusok hatékonyságáról. Ebben a pontban ennek megalapozása történik. Először is alkalmasan megválasztjuk az algoritmusok végrehajtási költségét, műveletigényét, ill. lépésszámát jelentő függvények értelmezési tartományát és értékkészletét.

A továbbiakban legyen $f(g, h$ stb.) olyan függvény, amely a természetes számok halmazán van értelmezve és nem-negatív valós értékeket vesz fel:

$$f : \mathbf{N} \rightarrow \mathbf{R}_0^+,$$

ahol $\mathbf{N} = \{0, 1, 2, \dots\}$.

Most definiáljuk egy adott g függvény esetén azon függvények osztályát (= halmazát), amelyek nagyságrendben nem nagyobbak, ill. nem kisebbek, mint g , továbbá g -vel azonos nagyságrendűek.

1.3.1. Definíciók

1. Egy adott g függvény esetén $O(g)$ -vel jelöljük függvényeknek azt a halmazát, amelyre

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

teljesül. Ha $f \in O(g)$, akkor azt mondjuk, hogy g aszimptotikus felső korlátja f -nek. Ebben az esetben szokásos módon inkább az $f = O(g)$ jelölést alkalmazzuk. Valamint használjuk a következő függvényhalmazt is:

$$o(g) = \{f : \text{minden } c > 0 \text{ esetén létezik } n_0 \geq 0, \text{ hogy minden } n \geq n_0 \text{ esetén } f(n) < cg(n)\}$$

2. Hasonlóan, jelölje $\Omega(g)$ azon függvények halmazát, amelyekre

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

áll fenn. Ha $f \in \Omega(g)$ – szokásos jelöléssel $f = \Omega(g)$ –, akkor azt mondjuk, hogy g aszimptotikus alsó korlátja f -nek. Az itt megjelenő másik függvényhalmaz pedig:

$$\omega(g) = \{f : \text{minden } c > 0 \text{ esetén létezik } n_0 \geq 0, \text{ hogy minden } n \geq n_0 \text{ esetén } f(n) > cg(n)\}$$

3. Végül, $\Theta(g)$ jelöli azt a függvény-osztályt, amelyet a

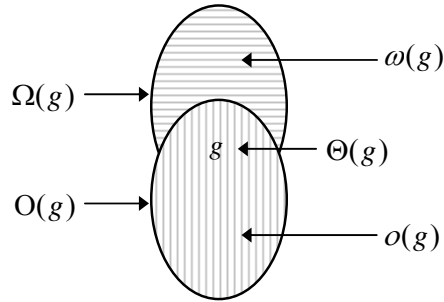
$$\Theta(g) = \{f : \text{létezik } c_1, c_2 > 0 \text{ és } n_0 \geq 0, \text{ hogy minden } n \geq n_0 \text{ esetén } c_1g(n) \leq f(n) \leq c_2g(n)\}$$

összefüggés ír le. Ha $f \in \Theta(g)$ – szokásos jelöléssel $f = \Theta(g)$ –, akkor azt mondjuk, hogy g aszimptotikusan éles korlátja f -nek.

A definíciók alapján teljesül a következő egyenlőség:

$$\Theta(g) = O(g) \cap \Omega(g).$$

Ha egy tetszőleges, rögzített g függvényt tekintünk, akkor a többi függvény általában besorolható a most definiált osztályok némelyikébe. Azonban fontos megjegyeznünk, hogy vannak nem összehasonlítható függvények is, pl. $g(n) = 1$ és $f(n) = n(1 + \sin n)$, ill. $g(n) = n$ és $f(n) = n^{1+\sin n}$ minden n -re. Ezekre ugyanis könnyen belátható, hogy sem $f = O(g)$, sem pedig $f = \Omega(g)$ nem teljesül. A nagyságrendi viszonyokról szemléletes képet ad az 1.5. ábra.



1.5. ábra: A függvényhalmazok egymáshoz való viszonya

Megjegyzés: A függvények aszimptotikus viszonyainak jelen vizsgálatához az szolgál alapul, hogy kellően nagy méretű bemenet esetén egy algoritmus futási idejének (ill. az azt leíró függvénynek) csak a nagyságrendje lényeges. Viszonylag kis méretű bemeneteket leszámítva tehát az aszimptotikusan leghatékonyabb algoritmus lesz a ténylegesen leggyorsabb!

1.3.2. Visszavezetés határértékekre

Valamely adott f és g függvény aszimptotikus viszonyát a $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ határérték – amennyiben létezik – a következőképpen határozza meg:

1. Ha $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} 0, \text{ vagy} \\ c > 0 \end{cases}$, akkor $f = O(g)$. $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$ esetén $f = o(g)$.
2. Ha $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} \infty, \text{ vagy} \\ c > 0 \end{cases}$, akkor $f = \Omega(g)$. $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$ esetén $f = \omega(g)$.
3. Ha $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c > 0$, akkor $f = \Theta(g)$.

Megjegyzés: A $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ határérték általában létezik, de pl. ha $f(n) = \begin{cases} 1, \text{ ha } n \equiv 1 \pmod{2} \\ 2, \text{ ha } n \equiv 0 \pmod{2} \end{cases}$, és $g(n) = 1$ minden n -re, akkor nem létezik a határérték, de a két függvény aszimptotikus viszonya mégis megállapítható, ebben az esetben $f = \Theta(g)$.

1.3.3. L'Hospital szabály alkalmazása

Mikor adott f és g függvényekre $\lim_{n \rightarrow \infty} f(n) = \infty$ és $\lim_{n \rightarrow \infty} g(n) = \infty$, és mindkét függvény valós kiterjesztése differenciálható, akkor gyakran alkalmazzuk a L'Hospital szabályt a $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ határérték meghatározására:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}$$

1.3.4. Tulajdonságok

A $O, o, \Theta, \omega, \Omega$ jelöléseket függvények közötti bináris relációként is felfoghatjuk (pl. $O = \{(f, g) : f = O(g)\}$). Így a relációkra vonatkozó ismert definíciók értelmezhetők rájuk, és beláthatók a következő állítások:

1. $O, o, \Theta, \omega, \Omega$ tranzitívak: f, g, h függvényekre pl.: $f = \omega(g) \wedge g = \omega(h) \Rightarrow f = \omega(h)$.
2. O, Θ, Ω reflexívek: pl. $f = \Omega(f)$.
3. Θ szimmetrikus: $f = \Theta(g) \Leftrightarrow g = \Theta(f)$.
4. O és Ω , valamint o és ω „felcserélten szimmetrikusak: pl. $f = O(g) \Leftrightarrow g = \Omega(f)$.
5. Rögzített h függvény mellett $O(h), o(h), \Theta(h), \omega(h), \Omega(h)$ halmazok zártak az összeadásra és a (pozitív) számmal való szorzásra nézve: pl. $f = \omega(h) \wedge g = \omega(h) \Rightarrow f + g = \omega(h)$; és $f = \omega(h) \wedge c > 0 \Rightarrow cf = \omega(h)$.
6. Összegben a nagyobb függvény határozza meg az aszimptotikát: $\max\{f, g\} = \Theta(f + g)$. (Itt a \max jelölés az aszimptotikusan nagyobb függvényt jelenti.) Ha ezt a szokásos alakot átírjuk az $f + g = \Theta(\max\{f, g\})$ formába, akkor könnyen kiolvasható belőle az, hogy egy szekvencia műveletigényének nagyságrendjét a nagyobb műveletigényű tag határozza meg.

1.3.5. Függvényosztályok

A fenti 1., 2. és 3. tulajdonságok miatt Θ – mint bináris reláció – ekvivalenciareláció a függvények halmazán, tehát osztályokra bontja azt. Az egyes osztályok reprezentálhatók a legegyszerűbb függvényükkel, pl.: $\Theta(1), \Theta(n), \Theta(n^2), \Theta(\sqrt{n}), \dots$

Megjegyzés: A gyakorlatban – amennyiben nem okoz félreértést – az egyszerűség kedvéért a $g, O(g), o(g), \Theta(g), \omega(g), \Omega(g)$ jelölések helyett gyakran a $g(n), O(g(n)), o(g(n)), \Theta(g(n)), \omega(g(n)), \Omega(g(n))$ jelöléseket használjuk. Így pl. a $3n^2 + 7n = \Theta(n^2)$ jelentése: $f = \Theta(h)$, ahol $f(n) = 3n^2 + 7n$ és $h(n) = n^2$ minden n -re.

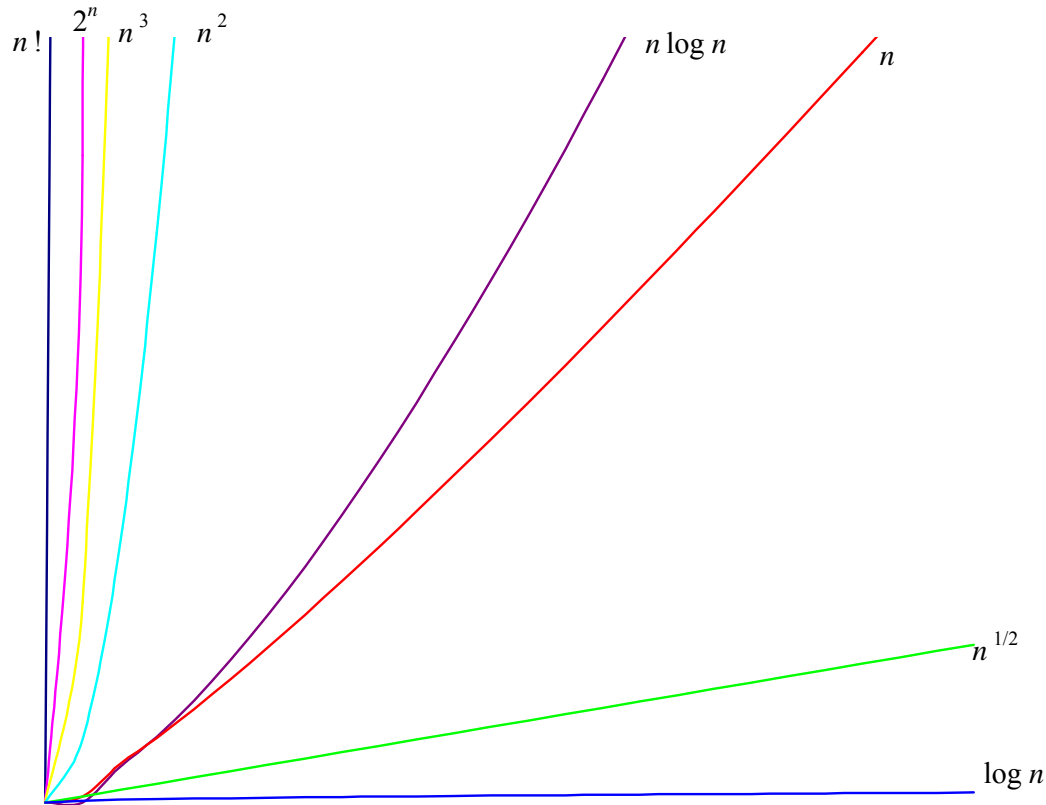
Néhány további – könnyen belátható – tulajdonság:

- Polinom esetén a legnagyobb fokú tag a meghatározó (lásd: fenti 6. tulajdonság):
$$a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0 = \Theta(n^k).$$
- Bármely két (1-nél nagyobb) alapszámú logaritmikus függvény aszimptotikusan egyenértékű: $\log_a n = \Theta(\log_b n)$ ($a > 1$ és $b > 1$). Ezért az alap feltüntetése nem szükséges: $\log_a n = \Theta(\log n)$
- Hatványfüggvények esetén különböző kitevők különböző függvényosztályokat jelölnek ki: $a \geq 0$ és $\varepsilon > 0$ esetén $n^a = O(n^{a+\varepsilon})$ és $n^a \neq \Theta(n^{a+\varepsilon})$, vagyis $n^a = o(n^{a+\varepsilon})$.

1.3.6. Nagyságrendek

Néhány jellegzetes, és a gyakorlatban gyakran előforduló függvény menetét és egymáshoz viszonyított aszimptotikus nagyságrendjét szemlélteti az 1.6. ábra (nem mérhetően, főleg nem az origó közelében). Ezek aszimptotikusan mind különbözőek.

A hatványfüggvények egyik – az ábrán is látható – fontos tulajdonsága, hogy bármely $\alpha > 1$ kitevő esetén n^α az $n \cdot \log n$ és az a^n ($a > 1$) közé, míg ha $0 < \alpha < 1$, akkor n^α a $\log n$ és az n közé esik aszimptotikusan.



1.6. ábra. A gyakran előforduló nagyságrendek

1.3.7. Példák algoritmusok aszimptotikus futási idejére

- Verem vagy sor bármely művelete $\Theta(1)$
- Logaritmikus keresés $\Theta(\log n)$ ($= \lfloor \log_2 n \rfloor + 1$)
- Lineáris keresés $\Theta(n)$
- Kupacrendezés $\Theta(n \log n)$
- Buborékrendezés $\Theta(n^2)$
- Mátrixszorzás $\Theta(n^3)$
- Hanoi tornyai $\Theta(2^n)$

1.3.8. Az aszimptotikus jelölések használata

1. Gyakori hiba, hogy algoritmusok műveletigényének jellemzésekor O -t használnak, de Θ -t értenek alatta. Az így leírt egyenlőség általában igaz, de mást (kevesebbet) fejez ki, mint amit a közlő ért alatta. Pl. a buborék rendezésre igaz, hogy $MT(n) = O(n^2)$, de többet mond az, hogy $MT(n) = \Theta(n^2)$.

2. A $T(n)$ jelölés használata is hibát rejthet. $T(n)$ az összes lehetséges futási időt magában foglalja, amelyek nagyságrendben különbözőek lehetnek. Pl. ahogy korábban láttuk, a buborék rendezésre igaz (Cs helyett most T -vel), hogy: $MT(n) = \Theta(n^2)$ és $AT(n) = \Theta(n^2)$. A legkedvezőbb esetben azonban nincs csere, vagyis $mT(n) = 0$. Ekkor a $T(n) = \Theta(n^2)$ (felületes) kijelentés nem igaz, helyette a $T(n) = O(n^2)$ kijelentést kellene használni, mely az összes esetre helyes lesz. De ez kevesebb információt tartalmaz, mint a fenti két egyenlőség, ezért ajánlatos $MT(n)$ -t és $AT(n)$ -t használni.
3. Néhány tipikus használat:
- $2n^2 + 3n + 1 = \Theta(n^2)$, $3n + 1 = O(n^2)$, de $3n + 1 \neq \Theta(n^2)$
 - $2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$
 - A logaritmikus keresés futási idejére fennáll a következő (közelítő) rekurzív egyenlet:

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1).$$