A függvények növekedése, az algoritmus idő-komplexitása

Bevezetés

Tegyük fel, hogy egy számítógépes program n számú egészből álló listát növekvő listává rendez. A program gyakorlati hatékonyságának megítélése szempontjából lényeges, hogy mennyi idő alatt végzi el a számítógép ezt a feladatot. Tegyük föl, hogy egy analízis az mutatja, hogy a rendezésre fordítandó idő n számú -bizonyos nagyságot meg nem haladó- egész szám esetén kevesebb mint f(n) mikrosec (a másodperc milliomod része), ahol
f(n) = 100 n log(n) + 25 n + 9. Ha a program hatékonyságát elemezni akarjuk, akkor tisztában kell lennünk azzal, hogy milyen gyorsan nő az n növekedésével összevetve az f függvény. A következőkben a függvény növekedésének becslésére mutatunk be néhány fontos módszert.Jelöléseket vezetünk be a függvények növekedési mértékének analizálására.

A nagy O-szimbólum (Landau-szimbólum)

Definíció: Legyen f és g N->R vagy R->R. Azt mondjuk,hogy $f(x)$ $O(g(x))$, ha léteznek olyan C és k állandók, amelyekre

$$|f(x)| \le C\,|g(x)|,$$

ha $k < x$.

1. Példa: Mutassuk meg, hogy $f(x) = x^2 + 2x + 1$ $O(x^2)$.
Megoldás: Azt kell megmutatnunk, hogy minden elég nagy x-re:

$$|x^2 + 2x + 1| \le C\,x^2$$

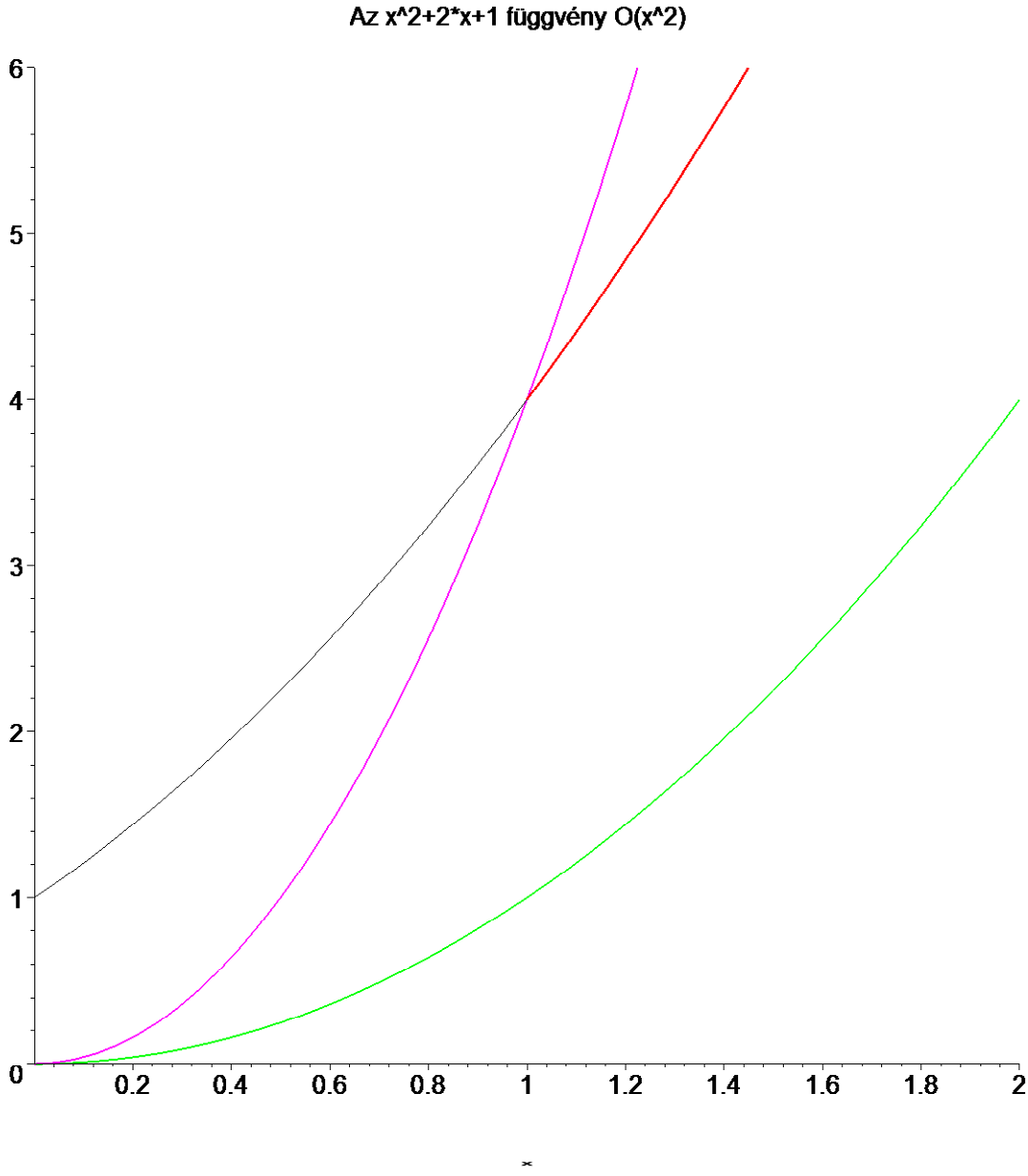
Mivel $1 < x$ esetén:

$$0 \le x^2 + 2x + 1 \quad \text{és} \quad x^2 + 2x + 1 \le x^2 + 2x^2 + x^2$$

$$x^2 + 2x + 1 < 4x^2, \text{ ha } 1 < x.$$

Így $C = 4$ ill. $k = 1$ megfelel. Szemléltessük is mindezt:

```
> restart:
> with(plots):
> abra1:=plot(x^2+2*x+1,x=0..1,color=black,thickness=1):
> abra2:=plot(x^2+2*x+1,x=1..2,color=red,thickness=3):
> abra3:=plot([4*x^2,x^2],x=0..2,color=[magenta,green],thicknes
  s=2):
> display({abra1,abra2,abra3},view=[0..2,0..6],scaling=unconstr
  ained,title=`Az x^2+2*x+1 függvény O(x^2)`);
```



>

Megfordítva is igaz. Az $x^2 \le x^2 + 2\,x + 1$ $(0 < x)$ miatt C=1 választással a $g(x) = x^2$ függvényre x^2 $O(x^2 + 2\,x + 1)$.
Tehát
az előző két függvényre $f(x)$ $O(g(x))$ és $g(x)$ $O(f(x))$ teljesül egyidejűleg.Ekkor azt mondjuk, hogy az f és a g **azonos rendű.**

2. Példa: Mutassuk meg, hogy $7\,x^2$ $O(x^3)$.

Megoldás: A $7\,x^2 < x^3$ egyenlőtlenség minden 7 -nél nagyobb x értékre fennáll. Így C=1 és k=7 választással beláttuk, hogy $7\,x^2$ $O(x^3)$

3. Példa: A 2. Példában beláttuk, hogy $7\,x^2$ $O(x^3)$. Igaz-e, hogy x^3 $O(7\,x^2)$?

Megoldás: Az a kérdés, hogy van-e olyan C és k állandó, amelyek mellett $x^3 \le C\,7\,x^2$. ha $k < x$. Az előző egyenlőtlenség egyenértékű az $x < 7\,C$ egyenlőtlenséggel. Ez pedig azt jelenti, hogy nem létezik megfelelő C, mert x tetszőleges nagy lehet. Tehát x^3 nem $O(7\,x^2)$.

A polinomok gyakran használhatók különböző függvények növekedésének becslésére.
A következő tétel segítséget nyújt ahhoz, hogy ne kelljen minden egyes esetben a szóbanforgó polinom növekedését vizsgálnunk.
A tétel állításának lényege: a polinom növekedését legmagasabb fokú tagja határozza meg.

1.Tétel: Legyen $f(x) = a_n x^n + a_{n-1} x^{(n-1)} + \ldots + a_1 x + a_0$, ahol $a_0, a_1, \ldots, a_n$ valós számok. Ekkor $f(x)$ $O(x^n)$.

Bizonyítás: A háromszög egyenlőtlenséget felhasználva, ha $1 < x$:

$$|f(x)| = \left| \sum_{i=0}^{n} a_i x^i \right| <= \sum_{i=0}^{n} |a_i| x^i$$

$$= x^n \left(|a_n| + \frac{|a_{n-1}|}{x} + \ldots + \left| \frac{a_1}{x^{(n-1)}} \right| + \left| \frac{a_0}{x^n} \right| \right)$$

$$<= x^n \left(\sum_{i=0}^{n} |a_i| \right),$$

tehát

$$|f(x)| \le C x^n,$$

ahol $C = \sum_{i=0}^{n} |a_i|$. Ezért $f(x)$ $O(x^n)$.

A következő néhány példa kapcsán feltételezzük, hogy a szereplő függvények értelmezési tartománya a pozitív egészek halmaza.

4. Példa: Hogyan tudjuk a nagy O jelölést az első n pozitív egész összegének becslésére felhasználni?

Megoldás: Induljunk ki a következő nyilvánvaló egyenlőtlenségből.

$$1 + 2 + \ldots + n \le n + n + n + \ldots + n = n^2.$$

Ebből $C = 1$ és $k = 1$ választással következik, hogy $1 + 2 + \ldots + n$ $O(n^2)$.

5. Példa: Adjunk nagy O becslést a faktoriális függvényre és annak logaritmusára. Faktoriális függvénynek nevezzük az $f(n) = n!$ függvényt, alol

n!=1*2* . . . *n

minden pozitív egészre és 0!=1.

Pl.: $1! = 1$, $2! = 1\,(2) = 2$, $3! = 1\,(2)\,(3) = 6$, $4! = 1\,(2)\,(3)\,(4) = 24$.

Megjegyezzük, hogy a függvény rendkívül gyorsan növekszik, pl.:

```
> `20!`=20!;
```

$$20! = 2432902008176640000$$

```
>
```

Megoldás: Könnyen kaphatunk nagy O becslést az $n!$ -ra, ha meggondoljuk, hogy a szorzat egyik tényezője sem nagyobb n-nél. Innen.

$$n! = 1\,(2)\,(3) \ldots n \le n\ n\ n \ldots n = n^n$$

Az egyenlőtlenség mutatja, hogy $n!$ $O(n^n)$. Az $n!$-ra fölírt egyenlőtlenség mindkét oldalának

logaritmusát véve kapjuk:

$$\ln(n!) \le \ln(n^n) = n\ln(n)$$

Ez pedig azt jelenti, hogy

$$\ln(n!) \quad O(n\ln(n))$$

6. Példa: Mutassuk meg, hogy n^k $O(e^n)$ (k pozitív egész)

Megoldás: Megmutatjuk, hogy $n^k < e^n$ teljesül minden $1 < n$ esetén.

Az $n^k < e^n$ egyenlőtlenség egyenértékű a mindkét oldal logaritmusának vételével adódó:

$$k\ln(n) < n$$

egyenlőtlenséggel. Ebből rendezéssel kapjuk a vele egyenértékű egyenlőtlenséget:

$$k < \frac{n}{\ln(n)}$$

Megmutatjuk, hogy $\ln(n) < \sqrt{n}$ minden $1 < n$ esetén, ami egyenértékű az előbbi és az eredeti egyenlőtlenséggel. Másként írva $0 < \sqrt{n} - \ln(n)$ igazolandó , ha $1 < n$

Bizonyítjuk, hogy a függvény minimuma pozitív.

```
> f:=x->sqrt(x)-ln(x);
```

$$f := x \to \sqrt{x} - \ln(x)$$

Képezzük a függvény deriváltját:

```
> Diff(f(x),x)=D(f)(x);
```

$$\frac{d}{dx}(\sqrt{x} - \ln(x)) = \frac{1}{2\sqrt{x}} - \frac{1}{x}$$

Hozuk a deriváltat közös nevezőre:

```
> Diff(f(x),x)=normal(D(f)(x));
```

$$\frac{d}{dx}(\sqrt{x} - \ln(x)) = \frac{x - 2\sqrt{x}}{2x^{(3/2)}}$$

A derivált zérushelye a lehetséges szélsőértékhely:

```
> zh:=solve(D(f)(x),x);
```

$$zh := 4$$

A második derivált értéke ezen a helyen:

```
> '(D@@2)(f)(zh)'=(D@@2)(f)(zh);
```

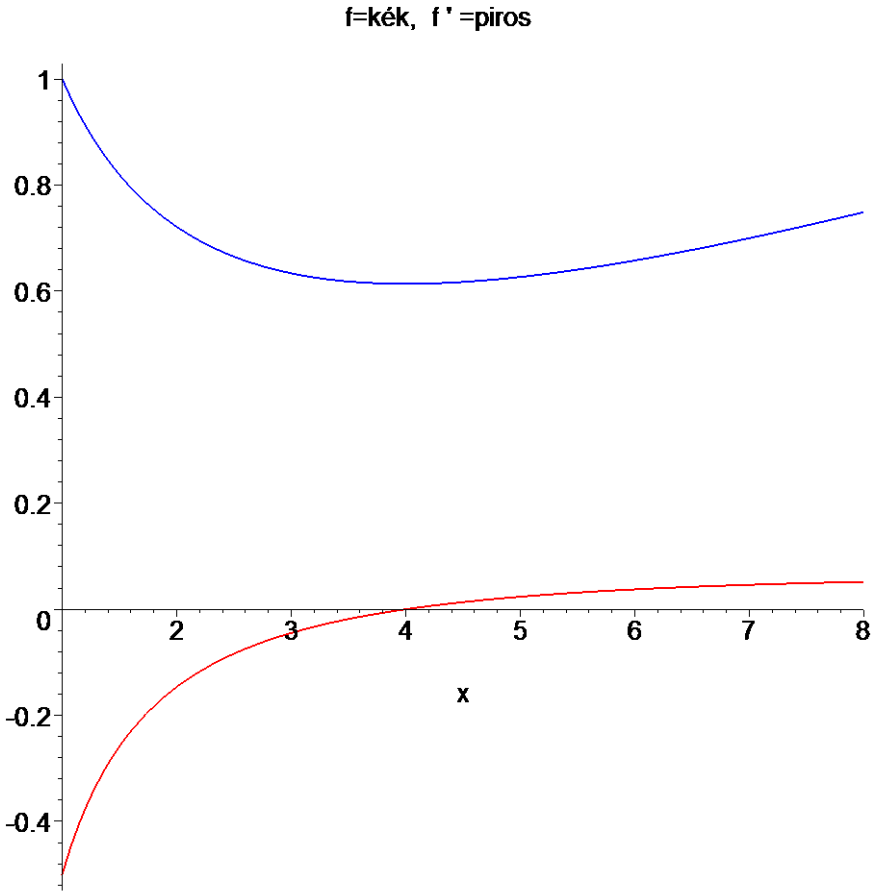
$$(D^{(2)})(f)(zh) = \frac{1}{32}$$

A második derivált az $x_0 = 4$ helyen pozitív, tehát itt a függvénynek minimuma van. Ennek értéke

```
> evalf(f(zh));
```

$$0.613705639$$

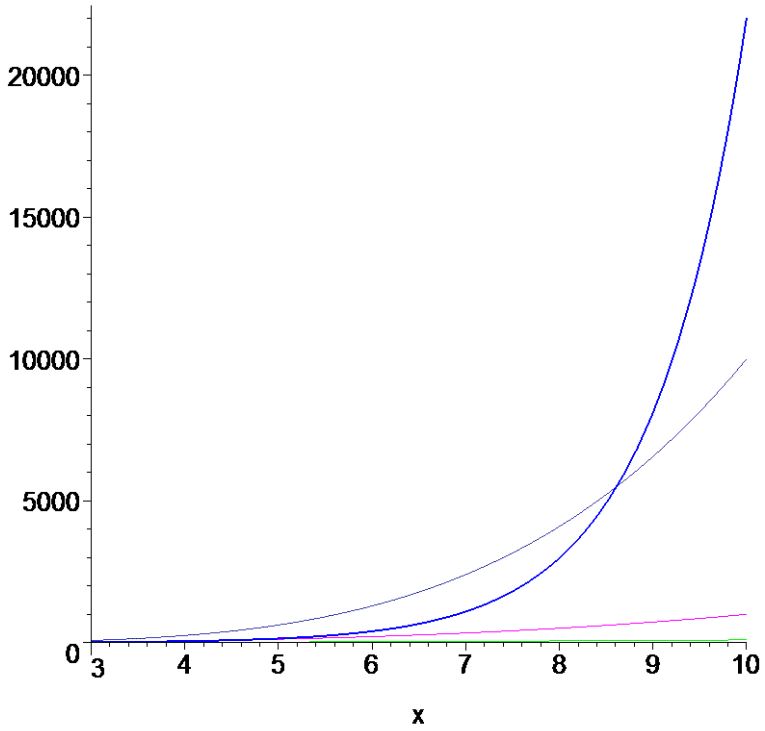
```
> plot([f(x),D(f)(x)],x=1..8,color=[blue,red],thickness=2,title
  =`f=kék,  f ' =piros`);
```



>

Ezzel igazoltuk, hogy n^k O(e^n) . Ez másként fogalmazva azt jelenti, hogy az $\mathbf{e}^x$ pozitív x-ekre gyorsabban nő bármely hatványfüggvénynél.

```
> plot([exp(x),x,x^2,x^3,x^4],x=3..10,color=[blue,red,green,magenta,navy],thickness=[2,1,1,1,1]);
```



Függvények kombinációinak növekedése

Sok esetben azalgoritmusok két vagy több részalgoritmusból épülnek föl. Ilyenkor a számítógép által a problémamegoldáshoz fölhasznált lépésszám az egyes részalgoritmusok lépésszámainak összege. Tehát csak úgy tudunk nagy O becslést adni az algoritmuslépésszámára, ha ilyen becslést adunk a részalgoritmusokra, majd kombináljuk ezeket.

Leggyakrabban függvények összegének és szorzatának becslésére van szükség.

2.Tétel: Tegyük föl, hogy $f_1(x)$ $O(g_1(x))$ és $f_2(x)$ $O(g_2(x))$. Ekkor $(f_1 + g_1)(x)$ $O(\max(g_1(x), g_2(x)))$.

Következmény: Tegyük föl, hogy $f_1(x)$ és $g_1(x)$ $O(g(x))$. Akkor $(f_1 + f_2)(x)$ $O(g(x))$.

3. Tétel: Tegyük föl, hogy $f_1(x)$ $O(g_1(x))$ és $f_2(x)$ $O(g_2(x))$. Ekkor $(f_1 f_2)(x)$ $O(g_1(x)\, g_2(x))$

7. Példa: Adjunk nagy O becslést az $f(n) = 3\, n \ln(n!) + (n^2 + 3) \ln(n)$ függvényre, ahol n pozitív egész.

Megoldás: Először a $3\, n \ln(n!)$ szorzatot becsüljük. Az 5. Példából tudjuk, hogy $\ln(n!)$ $O(n \ln(n))$ .Felhasználva ezt és azt a tényt, hogy $3\, n$ $O(n)$ az előző tételt alkalmazva adódik, hogy $3\, n \ln(n!)$ $O(n^2 \ln(n))$

Ezután az $(n^2 + 3) \ln(n)$ kifejezést becsüljük. Mivel $n^2 + 3 < 2\, n^2$, ha $2 < n$, $n^2 + 3$ $O(n^2)$.. Megint csak az előző tételt alkalmazva $(n^2 + 3) \ln(n)$ $O(n^2 \ln(n))$. A 2. Tételt alkalmazva adódik, hogy $f(n) = 3\, n \ln(n!) + n^2 \ln(n)$ $O(n^2 \ln(n))$.

Ahogy már említettük, a nagy O jelölést gyakran algoritmusok, eljárások lépésszámának becslésére használjuk. A becslések során gyakran használt függvények a következők:

$$1, \quad \ln(n), \quad n, \quad n \ln(n), \quad n^2, \quad 2^n, \quad n!$$

A sorban egymást követő függvények rendre gyorsabban nőnek a megelőzőnél. Ez másként fogalmazva azt is jelenti, hogy bármelyiket a rákövetkezővel osztva a kapott hányados ∞-ben vett határértéke 0.

>

A nagy Ω szimbólum

Definíció: Legyen f és g N->R vagy R->R. Azt mondjuk,hogy $f(x)$ $\Omega(g(x))$, ha léteznek olyan C és k állandók, amelyekre

$$C\,|g(x)| \le |f(x)|,$$

ha $k < x$.

Példa: Az $f(x) = 8\, x^3 + 5\, x^2 + 7$ $\Omega(g(x))$, ahol $g(x) = x^3$. Ez könnyen látható, hiszen $8\, x^3 \le 8\, x^3 + 5\, x^2 + 7 = f(x)$ minden pozitív x értékre. Ez egyenértékű azzal, hogy $g(x) = x^3$ $O(8\, x^3 + 5\, x^2 + 7)$, ami közvetlenül látható.

A nagy θ szimbólum

Gyakran szükség van arra, hogy a függvény növekedésének rendjét olyan viszonylag egyszerű referencia-függvényekkel (referencia = tájékoztatás) összehasonlítva mérjük, mint az x^n (n

pozitív egész) vagy c^x , ahol $1 < c$. A függvény növekedése rendjének meghatározása megkívánja, hogy a függvény méretére alsó- és felső korlátot is adjunk. Ez azt jelenti, hogy az adott $f(x)$ függvényhez olyan $g(x)$ referencia-függvényt keresünk, hogy $f(x)$ $O(g(x))$ és $f(x)$ $\Omega(g(x))$ is teljesüljön.

Definíció: Legyen f és g N->R vagy R->R. Akkor mondjuk, hogy $f(x)$ $\Theta(g(x))$, ha $f(x)$ $O(g(x))$ és $f(x)$ $\Omega(g(x))$ egyidejűleg. Ekkor azt mondjuk, hogy f nagy-Théta $g(x)$, vagy másképpen szólva $f(x)$ $g(x)$ rendű.

A definícióból következik, hogy ha $f(x)$ $\Theta(g(x))$, akkor $g(x)$ $\Theta(f(x))$ is teljesül. Általában a nagy Théta szimbólum $\Theta(g(x))$ alakú használata során a $g(x)$ viszonylag egyszerű referencia-függvény, mint pl.: x^n, c^x, $\log(x)$ és így tovább, és az $f(x)$ pedig viszonylag összetettebb függvény.

8. Példa:

A 4. Példában láttuk, hogy az első n pozitív egész összege $O(n^2)$. Igaz-e, hogy ennek az összegnek a rendje n^2 ?

Megoldás:

Legyen $f(n) = 1 + 2 + \ldots + n$. Mivel már beláttuk, hogy $f(n)$ $O(n^2)$, azt kell még megmutatnunk, hogy létezik olyan pozitív C állsndó, amelyre elegendően nagy n esetén $C\,n^2 < f(n)$ teljesül. Az összegre alsó becslést keresvén a tagok első felét elhagyhatjuk. A $\text{ceil}\left(\frac{n}{2}\right)$ (ceil=egész rész) -nél nem kisebb tagokat összegezve:

$$1 + 2 + \ldots + \text{n} >= \text{ceil}\left(\frac{n}{2}\right) + \text{ceil}\left(\frac{n}{2}\right) + 1 + \ldots + n$$

$$>= \text{ceil}\left(\frac{n}{2}\right) + \text{ceil}\left(\frac{n}{2}\right) + \ldots + \text{ceil}\left(\frac{n}{2}\right)$$

$$= \left(n - \text{ceil}\left(\frac{n}{2}\right) + 1\right) \text{ceil}\left(\frac{n}{2}\right)$$

$$>= \frac{n}{2}\frac{n}{2}$$

$$= \frac{n^2}{4}.$$

Ez azt jelenti, hogy $f(n)$ $\Omega(n^2)$. Ígytehát $f(n)$ rendje n^2, szimbólumokkal: $f(n)$ $\Theta(n^2)$.

A 4. és az előző példa alapján láthatjuk, hogy az $f(x)$ $\Theta(g(x))$ úgy mutatható meg, hogy keresünk olyan C_1 és C_2 pozitív valós számokat és pozitív valós k számot, amelyekre

$$C_1 \left| g(x) \right| <= \left| f(x) \right| <= C_2 \left| g(x) \right|,$$

ha $k \le x$. Ez ugyanis egyenértékű azzal, hogy $f(x)$ $O(g(x))$ és $f(x)$ $\Omega(g(x))$.

9. Példa:

Mutassuk meg, hogy $3\,x^2 + 8\,x\,\ln(x)$ $\Theta(x^2)$

Megoldás: Mivel $0 <= 8\,x\,\ln(x) \le 8\,x^2$, $3\,x^2 + 8\,x\,\ln(x) \le 11\,x^2$ minden $1 \le x$ esetén.

Következésképpen $3x^2 + 8x\ln(x)$ $O(x^2)$. Világos, hogy x^2 $O(3x^2 + 8x\ln(x))$. Ezért $3x^2 + 8x\ln(x)$ $\Theta(x^2)$.
A polinomok rendjének meghatározását egyszerűvé teszi az a tény, hogy a polinom rendjét legmagasabb fokú tagja határozza meg. Például az $f(x) = 4x^5 + 3x^4 + 2x + 1$ polinom x^5 rendű. Általánosan igaz a következő tétel:

Tétel: Legyen $f(x) = a_n x^n + a_{n-1} x^{(n-1)} + \ldots + a_1 x + a_0$, ahol az $a_0, a_1, \ldots, a_n$ valós számok és $a_n \neq 0$. Ekkor f(x) x^n rendű.

10. Példa:
A $3x^8 + 12x^7 + 24x^3 + 13$, $x^{17} - 23x^{14} + 4x + 2$, $-x^{99} + 4001x^{98} + 1002x$ rendre x^8, x^{17}, x^{99} rendű.

Az algoritmus fogalma

Matematikai problémák megoldása gyakran jól meghatározott lépéssorozat megvalósítását jelenti. Az ilyen lépéssorozatokat algoritmusnak nevezzük.
Kicsit pontosabban az algoritmus műveletek olyan egyértelmű sorozata, amelynek végrehajtása véges idő alatt befejeződik és mindig szolgáltat eredményt.
Elemezzük ideiglenes meghatározásunkat egy konkrét példa kapcsán!
Tegyük föl, hogy egy édesanya serdülő lányának a következőképpen írja le az almáspite készítésének lépéseit
1. Készíts tésztát;
2. Készítsd el a tölteléket;
3. Töltsd meg a töltelékkel a tésztát;
4. Süsd közepes tűznél 30 percig.
Nagyon valószínű, hogy ennek alapján a sütemény nem készül el! Az édesanya számára ezek az utasítások egyértelműek, nem kell részletezni azokat, a lány számára azonban nyílván további részletek lennének szükségesek. Célszerű bevezetnünk az úgynevezett primitív (eredeti) művelet fogalmát.
Definíció
Azt a legbonyolultabb és legösszetettebb műveletet, amelyet a végrehajtó (gép vagy ember) közvetlenül megért és végrehajt anélkül, hogy alapvetőbb összetevőire kéne bontani primitív műveletnek nevezzük.
Ezek után pontosabban is leírhatjuk az algoritmus fogalmát:

Definíció:
Az algoritmus műveletek olyan egyértelmű sorozata, amelynek végrehajtása véges idő alatt befejeződik és mindig szolgáltat eredményt. Ez részletezve a következőket jelenti:
1. Minden művelethez egyértelműen definiált a rákövetkező művelet;
2. Van eleje és vége;
3. Véges számú lépés után mindíg végetér;
4. Primitívekből (primitív műveletekből) épül föl.
Példa
Készítsünk algoritmust, amely meghatározza véges számú egész szám legnagyobbikát!

Megoldás

Az algoritmust a következő primitívekkel adhatjuk meg:

1. Tekintsük ideiglenes maximális elemnek a sorozat első elemét.
2. Hasonlítsuk össze a a sorozat következő elemét az ideiglenes maximummal és ha ez az elem nagyobb, akkor legyen ez az új maximum.
3. Ismételjük az előző lépést mindaddig, amíg van elem a sorozatban.
4. Álljunk meg, ha már nincs több eleme a sorozatnak. Az ekkor adódó ideiglenes maximum a sorozat legnagyobb eleme.

A fenti algoritmust természetesen számítógépel szeretnénk megvalósítani. Ekkor azonban csak olyan utasításokat használhatunk, amely az adott számítógépes nyelvben megengedett. Célszerű ezért előbb egy átmeneti, gép- és nyelvfüggetlen kódot alkalmaznunk, amely átmenetet képez az algoritmus angol nyelvű leírása és a programnyelvi kódja között. Az ilyen kódot **pszeudokód**nak nevezzük.

Az algoritmus pszeudokódja lehet az alábbi:

```
procedure max(a[1],a[2];, . .a[n];:integers)
  max:=a[1];
 for i:=2 to n
 if max< a[i] then max:= a[i];
{max a legnagyobb elem}
```

```
> `type/egeszek`:=list(integer);
```

$$type/egeszek := \text{list}(integer)$$

```
> L1:=[1,Pi,4];L2:=[1,2,3];
```

$$L1 := [1, \pi, 4]$$
$$L2 := [1, 2, 3]$$

```
> type(L1,egeszek);
  type(L2,egeszek);
```

$$false$$
$$true$$

```
> maxelem:=proc(L::egeszek)
    local maxi,i:# A maxi és az i lokális változók
    maxi:=L[1]:
    for i from 2 to nops(L) do
     if maxi<L[i] then maxi:=L[i] fi
    od:
    maxi;
    end:
```

```
> K:=[seq(rand() mod 1000,i=1..20)];
```

$$K := [534, 415, 465, 459, 869, 442, 840, 180, 450, 265, 23, 946, 657, 3, 29, 922, 199, 973, 344, 802]$$

```
> maxelem(K);
```

973

Az eljárást teljesebbé tehetjük azzal, hogy megjegyezzük a maximális elem indexét is.

```
> maxelem1:=proc(L::egeszek)
    local maxi,i,maxind:# A maxi és az i lokális változók
    maxi:=L[1]:
    maxind:=1:
    for i from 2 to nops(L) do
     if maxi<L[i] then
      maxi:=L[i]:maxind:=i:
     fi
    od:
    lprint(`A lista maximális eleme`,maxi,` és ez
   lista`,maxind,`-dik eleme`);
   end;
```

maxelem1 := **proc**(*L*::*egeszek*)
local *maxi*, *i*, *maxind*;
maxi := *L*[1];
maxind := 1;
for *i* **from** 2 **to** nops(*L*) **do if** *maxi* < *L*[*i*] **then** *maxi* := *L*[*i*]; *maxind* := *i* **end if**
end do;
lprint(*`A lista maximális eleme`*, *maxi*, *` és ez lista`*, *maxind*, *`-dik eleme`*)
end proc

```
> maxelem1(K);
`A lista maximális eleme`, 973, ` és ez lista`, 18, `-dik eleme`
>
```

Keresési algoritmusok

Sok esetben merül fel a következő probléma: Adott egy rendezett lista, s meg kell keresni, hogy egy bizonyos elemet tartalmaz-e a lista ,és ha igen, hányadik helyen. Az ilyen típusú problémákat **keresési problémá**nak nevezzük. Két kereső algoritmust vizsgálunk az alábbiakban.

Lineáris keresés

Álljon a lista az $a_1, a_2, \ldots, a_n$ elemekből. Azt keressük, hogy az **x** elem előfordul-e a listában, s ha igen, akkor hányadik helyen. A lineáris, vagy szekvenciális keresés esetén a következőképpen járunk el:
Összehasonlítjuk az x-et és az a_1-et. Ha $x = a_1$, akkor a megoldás az a_1 pozíciója, vagyis 1. Ha $x \neq a_1$, akkor összehasonlítjuk az x-et az a_2-vel. Ha $x = a_2$, akkor a megoldás az a_2 helye, vagyis 2. Ha $x \neq a_2$, akkor összehasonlítjuk az x-et az a_3-mal. Az eljárást addig folytatjuk, míg
meg nem találjuk a listában az x-et, vagy a lista végére nem érünk, anélkül, hogy megtalálnánk.
Az első esetben az x-szel megegyező elem helye, a második esetben 0 a megoldás.

a) A lineáris keresés pszeudokódja:

---

eljárás lineáris keresés (x: egész, a_1, a_2, . . . ,a_n: különböző egészek)
$i := 1$
AMÍG $(i \le n$ és $x \ne a_i)$ FENNÁLL
ISMÉTELD
$i := i + 1$
CIKLUSVÉG
HA $i \le n$ AKKOR *hely* := i
EGYÉBKÉNT *hely* := 0
{ A *hely* az x-szel egyenlő elem indexe, vagy 0, ha az x-et nem találtuk a listában }

---

b) A lineáris keresés pszeudokódja alapján megírhatjuk a Maple-eljárást. Mindenekelőtt definiáljuk az egészekből álló lista típusát!

```
> `type/egeszek`:=list(integer);
```

type/egeszek := list(*integer*)

```
> L1:=[1,Pi,4];L2:=[1,2,3];
```

$L1 := [1, \pi, 4]$

$L2 := [1, 2, 3]$

```
> type(L1,egeszek);
```

false

```
> type(L2,egeszek);
```

true

A Maple-eljárás az algoritmus primitívek Malpe-szintaktika szerinti angolra fordítása és a megfelelő típusú formális paraméterek, valamint a lokális változók felvétele csupán:

```
> lineark:=proc(x::integer,L::egeszek)
> local i,hely:
> i:=1:
> while(i<=nops(L) and x<>L[i]) do
>  i:=i+1
> od:
> if i<=nops(L) then hely:=i else
> hely:=0 fi:
> hely;
> end:
```

A következő összetett utasítás a véletlenszerűen választott, modulo 200 redukált ,egészek 100 elemű halmazából készített lista:

```
> K:=sort(convert({seq(rand() mod 200,i=1..100)},list));
```

$K := [3, 5, 8, 11, 15, 18, 19, 26, 28, 34, 35, 38, 42, 43, 47, 48, 49, 50, 52, 57, 58, 60,$
$64, 65, 67, 70, 72, 73, 77, 79, 83, 88, 89, 96, 97, 100, 110, 115, 116, 117, 119, 120,$
$122, 128, 129, 134, 135, 137, 138, 140, 143, 146, 147, 148, 152, 156, 159, 162, 166,$
$169, 170, 172, 174, 176, 178, 179, 182, 183, 184, 188, 191, 192, 193, 194, 195, 196,$
$198]$

Legyen az x is véletlenszerűen választott, mod 200 redukált, egész:

```
> x:=rand() mod 200;
```

$$x := 188$$

```
> a:=time():
> lineark(x,K);
> b:=time()-a;
```

$$70$$

$$b := 0.$$

```
>
```

Bináris keresés

A bináris keresés algoritmusa akkor alkalmazható, ha a lista (növekvőleg) rendezhető. (Pl.: a tagok valós számok, szavak, amelyek nyílván alfabetikusan rendezhetők stb.)
Az elhelyezendő **x** elemet a lista középső elemével (ha a lista páros elemszámú, akkor az első felének utolsó elemével) hasonlítjuk össze. Ezután a listát két részlistára bontjuk. Ha az x nem nagyobb a "középső" elemnél, akkor az első részlistában folytatjuk a keresést, egyébként a másodikban. A keresés minden egyes lépésnél lényegében feleződő részlistában folytatódik, így a lineáris keresésnél hatékonyabb. A hatékonyság kérdését később részletesen is megvizsgáljuk majd.
a) Az eljárás **pszeudokódj**a:

---

eljárás bináris keresés (x: egész, $a_1, a_2, \ldots, a_n$: különböző egészek)
$i := 1$ { i a keresés intervallumának baloldali végpontja}
$j := n$ { j a keresés intervallumának jobboldali végpontja}
AMÍG $i < j$ FENNÁLL
ISMÉTELD
KEZDET
$m := \text{floor}\left(\frac{i+j}{2}\right)$ { floor = "egész része"}
HA $a_m < x$ AKKOR $i := m + 1$
EGYÉBKÉNT $j := m$
VÉGE
HA $x = a_i$ AKKOR $hely := i$
EGYÉBKÉNT $hely := 0$
{ A **hely** az **x**-szel egyenlő elem indexe, vagy 0, ha az x-et nem találtuk a listában }

---

b) A **Maple-eljárás**:

```
> binark:=proc(x::integer,L::egeszek)
> local i,j,m,hely,L1:
>  i:=1:
>  j:=nops(L):
>  L1:=sort(L):
>  while i<j do
>    m:=floor((i+j)/2):
>    if x>L1[m] then i:=m+1
```

```
>    else j:=m
>   fi
>  od:
> if x=L1[i] then hely:=i
> else
>  hely:=0
> fi
> end:
> x:=rand() mod 1500;#A keresett elem előállítása
```

$$x := 82$$

```
> K:=sort(convert({seq(rand() mod 1500,i=1..1000)},list));#
  Az elemek listájának generálása
```

$K := [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 16, 23, 26, 29, 30, 32, 35, 37, 40, 43, 44, 46,$
$48, 49, 51, 56, 57, 59, 65, 68, 70, 72, 75, 79, 80, 83, 84, 87, 88, 89, 92, 93, 94, 97, 99,$
$100, 102, 104, 106, 107, 108, 109, 110, 111, 112, 115, 116, 117, 119, 120, 121, 122,$
$124, 125, 126, 127, 128, 129, 130, 132, 134, 136, 137, 138, 139, 141, 142, 147, 148,$
$152, 154, 158, 160, 162, 164, 171, 173, 174, 175, 178, 179, 181, 191, 195, 196, 201,$
$203, 208, 209, 212, 213, 215, 220, 222, 223, 229, 230, 231, 234, 236, 237, 238, 239,$
$245, 246, 248, 249, 255, 256, 257, 258, 259, 264, 265, 266, 267, 271, 273, 274, 275,$
$276, 279, 281, 283, 284, 287, 288, 290, 291, 292, 293, 297, 298, 299, 300, 301, 306,$
$307, 308, 309, 311, 312, 314, 315, 316, 317, 318, 320, 321, 323, 325, 330, 334, 337,$
$340, 342, 343, 346, 348, 349, 350, 353, 354, 360, 364, 369, 371, 372, 373, 375, 376,$
$377, 379, 380, 384, 386, 389, 391, 394, 395, 399, 401, 405, 406, 407, 408, 409, 411,$
$412, 417, 418, 419, 420, 421, 423, 426, 427, 428, 429, 431, 434, 436, 438, 439, 442,$
$443, 446, 447, 449, 450, 451, 453, 454, 455, 456, 459, 460, 462, 463, 467, 468, 469,$
$470, 475, 476, 477, 480, 481, 482, 485, 486, 487, 488, 490, 492, 493, 495, 496, 497,$
$498, 499, 501, 503, 504, 505, 507, 508, 509, 512, 513, 517, 520, 523, 525, 526, 528,$
$529, 530, 534, 535, 541, 543, 544, 545, 550, 553, 554, 557, 558, 560, 563, 565, 567,$
$574, 575, 576, 577, 582, 586, 589, 590, 591, 592, 595, 596, 597, 598, 599, 601, 603,$
$604, 607, 612, 614, 616, 617, 618, 619, 623, 625, 626, 627, 629, 630, 632, 635, 636,$
$638, 642, 643, 644, 645, 647, 648, 649, 654, 655, 656, 660, 661, 663, 664, 665, 666,$
$668, 671, 672, 674, 679, 680, 681, 683, 684, 685, 686, 690, 691, 697, 698, 702, 703,$
$704, 705, 710, 712, 713, 714, 715, 716, 717, 721, 723, 724, 725, 726, 728, 730, 731,$
$737, 740, 743, 744, 747, 748, 749, 754, 757, 762, 763, 764, 765, 768, 769, 770, 771,$
$777, 779, 780, 781, 782, 784, 785, 786, 788, 789, 790, 794, 795, 796, 797, 798, 799,$
$801, 803, 804, 806, 807, 809, 810, 811, 817, 822, 825, 826, 827, 830, 836, 837, 838,$
$839, 840, 841, 843, 844, 845, 846, 847, 848, 849, 851, 852, 853, 856, 858, 861, 862,$
$863, 864, 865, 866, 868, 869, 871, 872, 873, 875, 876, 877, 882, 887, 890, 896, 901,$
$904, 906, 907, 910, 912, 913, 916, 917, 919, 920, 923, 925, 926, 931, 932, 933, 934,$
$935, 938, 939, 940, 941, 944, 945, 946, 951, 955, 956, 959, 964, 966, 967, 969, 970,$

971, 973, 976, 981, 982, 983, 984, 986, 988, 991, 993, 995, 998, 1001, 1002, 1004, 1007, 1008, 1009, 1013, 1015, 1017, 1019, 1023, 1028, 1030, 1031, 1033, 1034, 1035, 1038, 1040, 1042, 1043, 1045, 1047, 1050, 1051, 1052, 1053, 1054, 1056, 1062, 1063, 1067, 1069, 1074, 1076, 1077, 1078, 1079, 1081, 1083, 1085, 1086, 1091, 1093, 1095, 1097, 1098, 1099, 1100, 1103, 1108, 1109, 1115, 1119, 1122, 1124, 1126, 1130, 1131, 1132, 1136, 1137, 1141, 1142, 1143, 1148, 1149, 1153, 1154, 1156, 1161, 1164, 1165, 1166, 1170, 1172, 1173, 1174, 1179, 1180, 1184, 1185, 1187, 1188, 1189, 1193, 1194, 1196, 1197, 1198, 1200, 1204, 1205, 1206, 1207, 1211, 1212, 1214, 1216, 1217, 1218, 1221, 1222, 1223, 1226, 1227, 1230, 1233, 1234, 1235, 1236, 1244, 1245, 1246, 1247, 1249, 1253, 1254, 1256, 1257, 1262, 1267, 1268, 1276, 1281, 1282, 1283, 1284, 1285, 1286, 1289, 1290, 1295, 1296, 1299, 1302, 1303, 1309, 1312, 1314, 1315, 1318, 1321, 1323, 1324, 1325, 1328, 1329, 1330, 1339, 1341, 1342, 1343, 1345, 1346, 1347, 1350, 1352, 1353, 1355, 1356, 1357, 1360, 1362, 1364, 1365, 1366, 1370, 1372, 1375, 1376, 1378, 1384, 1387, 1388, 1389, 1391, 1396, 1398, 1400, 1402, 1403, 1404, 1406, 1408, 1409, 1412, 1414, 1416, 1419, 1420, 1421, 1422, 1423, 1424, 1425, 1428, 1429, 1432, 1435, 1439, 1441, 1442, 1445, 1447, 1452, 1455, 1457, 1459, 1461, 1463, 1466, 1470, 1471, 1472, 1473, 1474, 1477, 1478, 1480, 1482, 1484, 1487, 1497, 1498]

```
> binark(x,K);
```

0

Idő-komplexitás

Az algoritmusok összetettségét, hatékonyságát vizsgálhatjuk a szükséges memória-kapacitás vagy a végrehajtáshoz szükséges bit-műveletek száma alapján. Bizonyos bit-művelet végrahajtásához természetesen adott számítógép esetén adott időintervallum rendelhető. Ilyen értelemben beszélhetünk az **algoritmus idő-komplexitás**áról. Mivel azonban ez gépfüggő, a továbbiakban a végrehajtandó operációk számával mérjük az algoritmusok idő-komplexitását.

Példa: Hasonlítsuk először össze a lineáris- és a bináris keresés algoritmusát a futási idő (CPU-idő)alapján:

```
> K:=sort(convert({seq(rand() mod 20000,i=1..10000)},list)):# A véletlenszerűen választott lista
```

```
> a:=time():# A start ideje
> binark(6793,K);
> `CPU-idő`:=time()-a; # A befejezés időpontjából kivonva a start időt, adódik a végrehajtás CPU-idő
```

2708

CPU-idő := 0.015

```
>
> c:=time():
> lineark(6793,K);
> b:=time()-c;
```

2708

$$b := 0.$$

Hívjuk meg egymás után többször mindkét eljárást adjuk össze a felhasznált CPU-időket, majd számítsuk ki a kapott összes idők hányadosát!

```
> linido:=0: binido:=0: szam:=30:#a keresések száma
> for i to szam do
> x:=rand() mod 40000:
> K:=sort(convert({seq(rand() mod 40000,i=1..20000)},list)):
> a:=time():
> binark(x,K):
> b:=time()-a;
> binido:=binido+b:
> c:=time():
> lineark(x,K);
> d:=time()-c;
> linido:=linido+d:
> od:
> idoarany:=binido/linido;
```

$$idoarany := 0.1023255814$$

>

A vizsgálat azt látszik igazolni, hogy - a várakozásnak megfelelően -a bináris keresés kevesebb időt igényel.

Vizsgáljuk most meg észletesen a kétféle keresési algoritmus idő-komplexitását!

1. Lineáris keresés: Az idő-komplexitás mérésének mértékéül az elvégzendő összehasonlítások számát választhatjuk. A ciklus-utasítás minden egyes végrehajtásakor két összehasonlítást végzünk: egyet annak vizsgálatára, hogy a lista végére értünk-e már egyet pedig az x-nek a
lista illetékes elemével való összehasonlítására. Végül még egy összehasonlítás történik a cikluson kívül. Következésképp, ha $x = a_i$, akkor $2\,i + 1$ összehasonlítás történik. A legtöbb összehasonlítást akkor végezzük, ha az x nem eleme a listának.Ekkor $2\,n$ összehasonlítást végzünk annak megállapítására, hogy x nem egyenlő a_i-vel, i = 1,2, . . . ,n-re. Egy összehasonlítás történik a ciklusból való kilépéskor, egy pedig a cikluson kívül. Tehát az O (ordo) szimbólummal leírva a lineáris keresés legföljebb $O(n)$ összehasonlítást igényel.

Az elvégzett komplexitás-analízist **legrosszabb-eset analízis**nek mondjuk. Azt vizsgáltuk, hogy rögzített input-méret mellett legföljebb hány operációt kell végrehajtani. Tehát a **legrosszabb-eset analízis azt mutatja meg, hogy az adott algoritmus esetén hány lépés garantálja a megoldást.**

2. Bináris keresés: Az egyszerűség kedvéért tételezzük föl, hogy $n = 2^k$ listaelemünk van: $a_1, a_2, \ldots, a_n$, ahol k nem-negatív egész. Ekkor $k = \log_2(n)$. (Ha az n, az elemek száma, nem 2-hatvány, akkora listát egy hosszabb, $2^{(k+1)}$ elemű lista részlistájaként tekinthetjük, amelyre:

$$2^k < n < 2^{(k+1)}.$$

Az algoritmus minden egyes lépésénél összehasonlítjuk az i-t és a j-t, tehát az első és az

utolsó elemet, megállapítandó, hogy a redukált lista több mint egy elemet tartalmaz-e. Ha $i < j$, akkor mégegy öszehasonlítást végzünk minden egyes lépésnél annak eldöntésére, hogy az x nagyobb-e, mint a redukált lista középső eleme.

Az első lépés során a keresést $2^{(k-1)}$ elemre redukáljuk, s ennek során két összehasonlítást tettünk. Az eljárást folytatjuk és minden redukciónál két összehasonlítást teszünk: a $2^{(k-1)}$ elemű listát két összehasonlítással $2^{(k-2)}$ eleműre csökkentjük, mig utóljára a 2 elemű listát két összehasonlítással 1 eleműre csökkentjük. Végül az egy elemű listánál egy összehasonlítás megmutatja, hogy nincs több elem és egy pedig azt , hogy a megmaradt elem egyenlő-e az x-szel.

Tehát összesen $2k + 2 = 2\log_2(n) + 2$ összehasonlítás történt a 2^k elemű lista esetén.

(Ha az n nem 2-hatvány, akkor az eredeti listát kiegészítjük $2^{(k+1)}$ elemű listára, ahol $k = \text{floor}(\log_2(n))$ és a keresés legfeljebb $2\,\text{floor}(\log_2(n)) + 2$ összehasonlítást igényel.)

Következésképpen a bináris keresés legfölbjebb $O(\log(n))$ összehasonlítást ígényel. Tehát a bináris keresés- a legrosszabb eseteket összehasonlítva - hatékonyabb, mint a lineáris keresés.

Az egyes algoritmusok idő-komplexitásának leírására leggyakrabban használt függvényeket és a használt terminológiát mutatja az alábbi táblázat:

Komplexitás	Terminológia
$O(1)$	konstans komplexitás
$O(\ln(n))$	logaritmikus komplexitás
$O(n)$	lineáris komplexitás
$O(n\ln(n))$	$n\ln(n)$ komplexitás
$O(n^b)$	polinomiális komplexitás
$O(b^n)$, ahol $1 < b$	exponenciális komplexitás
$O(n!)$	faktoriális komplexitás

Feltételezve, hogy a számítógép minden egyes bit-műveletre $10^{(-9)}$ sec időt használ föl a különböző méretű és időkomplexítású problémákat kivitelező algoritmus kivitelezésének időigényét tartalmazza az alábbi tábázat. A * szimbólum 10^{100} évnél nagyobb időigényt jelöl.

Az algoritmus kivitelezésére a számítógép által felhasznált idő						
A probléma mérete	Felhasznált bit-operációk					
n	log n	n	n log n	n^2	2^n	n!
10	3×10^{-9}s	10^{-8}s	3×10^{-8}s	10^{-7}s	10^{-6}s	3×10^{-3}s
10^2	7×10^{-9}s	10^{-7}s	7×10^{-7}s	10^{-5}s	4×10^{13}év	*
10^3	1.0×10^{-8}s	10^{-6}s	1×10^{-5}s	10^{-3}s	*	*
10^4	1.3×10^{-8}s	10^{-5}s	1×10^{-4}s	10^{-1}s	*	*
10^5	1.7×10^{-8}s	10^{-4}s	2×10^{-3}s	10s	*	*
10^6	2×10^{-8}s	10^{-3}s	2×10^{-2}s	17min	*	*

Példák, feladatok

1. Példa: Vizsgáljuk meg, hogy a következő függvényekre teljesül-e, hogy $f(x)$ O(x):

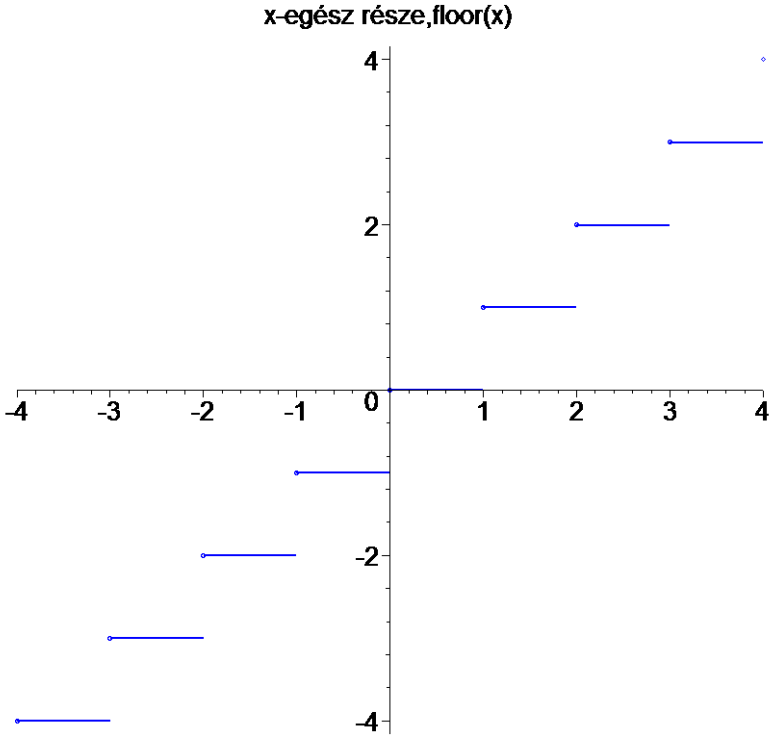
a) $f(x) = 10$; b) $f(x) = 3x + 7$;

c) $f(x) = x^2 + x + 1$; d) $f(x) = 5\ln(x)$;

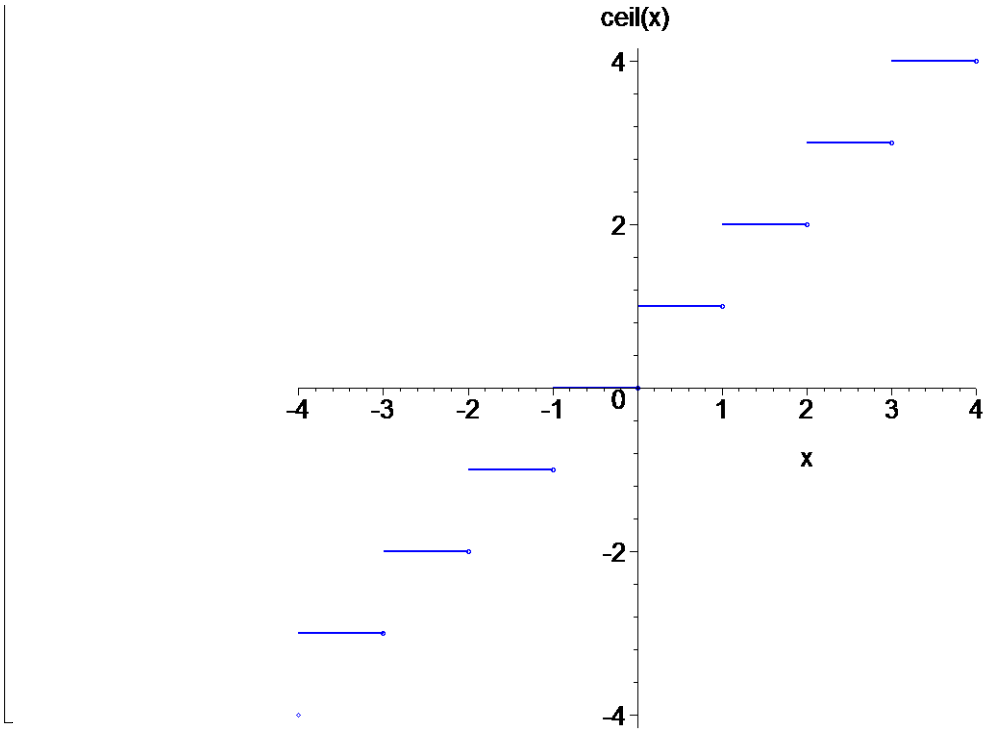
e) $f(x) = \text{floor}(x)$; f) $f(x) = \text{ceil}\left(\frac{x}{2}\right)$ (floor(x) : az x-nél nem nagyobb egészek legnagyobbika ceil(x): az x-nél nem kisebb egészek legkisebbike)

Ábrázoljuk a floor és a ceil függvényeket!

```
> restart:
> pontok:=seq([i,i],i=-4..3):
> pont:=plot([pontok],color=blue,style=point,symbol=circle):
> gorbe:=plot(floor(x),x=-4..4,color=blue,thickness=2,discont=t
  rue):
> plots[display]({pont,gorbe},title=`x-egész része,floor(x)`);
```




```
>
> pontok:=seq([i,i],i=-3..4):
> pont:=plot([pontok],color=blue,style=point,symbol=circle):
> gorbe:=plot(ceil(x),x=-4..4,color=blue,thickness=2,discont=tr
  ue):
> plots[display]({pont,gorbe},title=`ceil(x)`);
```




```
> 
```

Megoldás:

a) Igen, mert $|10| \le |x|$, minden $10 < x$ esetén.

b) Igen, mert $|3x + 7| \le |4x| = 4|x|$ minden $7 < x$ esetén.

c) Nem, ugyanis nem létezik olyan C állandó, amely mellett $|x^2 + x + 1| \le C|x|$ teljesülne minden elég nagy x esetén.

d) Igen. Ez következik abból, hogy $\ln(x) < x$ teljesül minden pozitív x értékre.

e) Igen, Ez következik abból, hogy $\text{floor}(x) \le x$, s így minden $0 < x$ esetén $|\text{floor}(x)| \le |x|$.

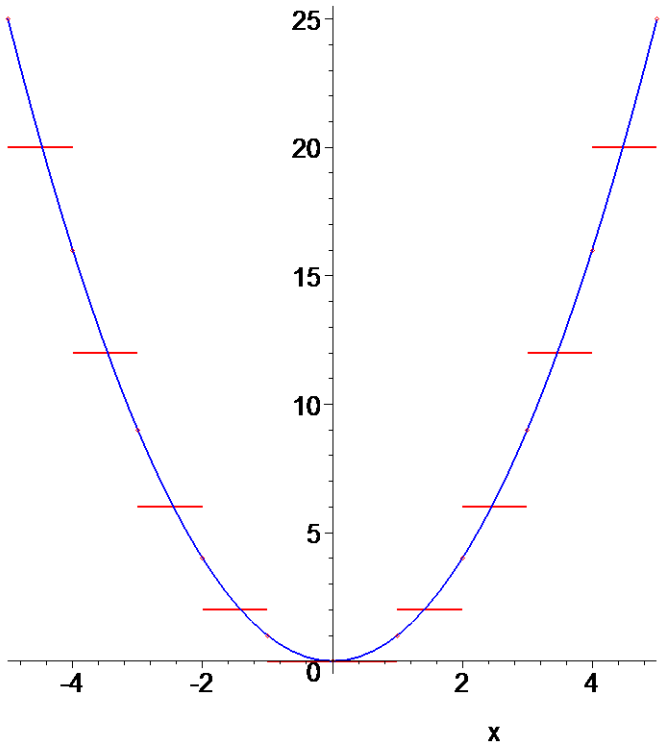
f) Igen. Ez következik abból a tényből, hogy $\text{ceil}\left(\frac{x}{2}\right) \le \frac{x}{2} + 1$. Tehát

$$\left|\text{ceil}\left(\frac{x}{2}\right)\right| \le \left|\frac{x}{2} + 1\right| <= |x| \text{ minden } 2 < x \text{ esetén.}$$

2. Példa: Legyen $f(x) = \text{floor}(x)\,\text{ceil}(x)$. Igazoljuk, hogy $f(x)$ $O(x^2)$.

Megoldás: Ábrázoljuk először - a tájékozódás kedvéért- az f függvényt és az x^2 függvényt azonos koordinátarendszerben:

```
> plot([x^2,floor(x)*ceil(x)],x=-5..5,discont=true,color=[blue,
  red],thickness=2);
```



Az ábra alapján igenlő válasz látszik valószínűnek. Valóban mivel $\text{ceil}(x) \le x + 1$ és $\text{floor}(x) \le x$ teljesül minden x-re, ezért ha $0 < x$, akkor $\text{floor}(x)\,\text{ceil}(x) \le x\,(x + 1)$<=$2\,x^2$, ha $1 \le x$

1. Feladat: Döntse el, hogy a következő függvények közül melyik $O(x^2)$:

a) $f(x) = 17\,x + 11$; b) $f(x) = x^2 + 1000$;

c) $f(x) = x\,\ln(x)$; d) $f(x) = \frac{x^4}{2}$;

e) $f(x) = 2^x$; f) $f(x) = \text{ceil}(x)^2$;

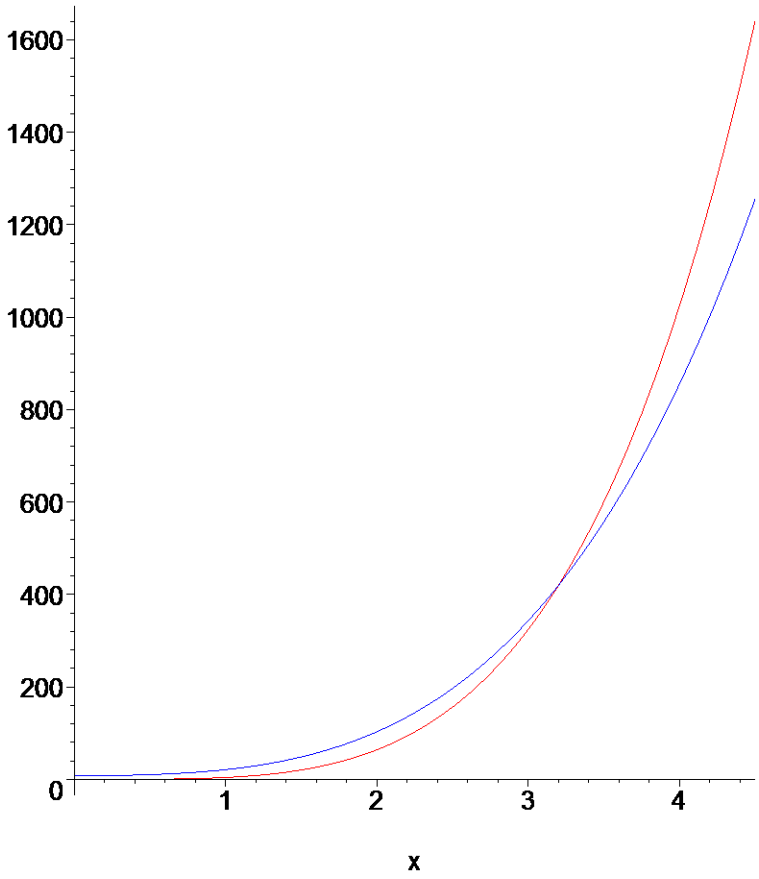
3. Példa: Mutassuk meg a definíciót fölhasználva, hogy ha $x^4 + 9\,x^3 + 4\,x + 7$ $O(x^4)$.

Megoldás: Az alacsonyabb fokú tagokra felső korlátot kell adnunk:

Ha $9 < x$, akkor $x^4 + 9\,x^3 + 4\,x + 7 \le x^4 + x^4 + x^4 + x^4 = 4\,x^4$. Ezért $x^4 + 9\,x^3 + 4\,x + 7$ $O(x^4)$.$C = 4$ és $k = 9$ mellett ui. $x^4 + 9\,x^3 + 4\,x + 7 \le C\,x^4$

Szemléltetve:

```
> plot([x^4+9*x^3+4*x+7,4*x^4],x=0..4.5,color=[blue,red]);
```



>

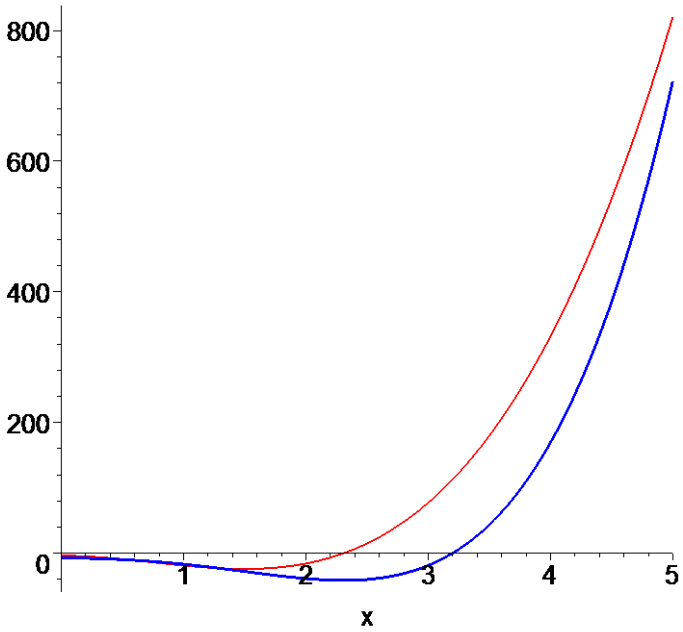
Az ábra alapján úgy látszik, hogy finomíthatjuk a becslést, vagyis csökkenthetjük pl. a k értékét. Vizsgáljuk a $g(x) = 4x^4 - (x^4 + 9x^3 + 4x + 7) = 3x^4 - 9x^3 - 4x - 7$ függvényt, tehát a két függvény különbségét!

```
> g:=x-> 3*x^4-9*x^3-4*x-7;
```

$$g := x \to 3x^4 - 9x^3 - 4x - 7$$

Ábrázoljuk a függvényt és deriváltját azonos koordinátarendszerben:

```
> plot([g(x),D(g)(x)],x=0..5,color=[blue,red],thickness=[3,2]);
```



Az ábra alapján az sejthető, hogy pl. $k = 3.5$ is megfelel. A különbségfüggvény szélsőértékét is meghatározhatjuk. A derivált komplex gyökei:

```
> gyok:=[solve(D(g)(x),x)];
```

$$gyok := \left[\frac{(1017+72\sqrt{97})^{(1/3)}}{12} + \frac{27}{4\,(1017+72\sqrt{97})^{(1/3)}} + \frac{3}{4}, -\frac{(1017+72\sqrt{97})^{(1/3)}}{24} - \frac{27}{8\,(1017+72\sqrt{97})^{(1/3)}} + \frac{3}{4} + \frac{1}{2} I \sqrt{3} \left(\frac{(1017+72\sqrt{97})^{(1/3)}}{12} - \frac{27}{4\,(1017+72\sqrt{97})^{(1/3)}} \right), -\frac{(1017+72\sqrt{97})^{(1/3)}}{24} - \frac{27}{8\,(1017+72\sqrt{97})^{(1/3)}} + \frac{3}{4} - \frac{1}{2} I \sqrt{3} \left(\frac{(1017+72\sqrt{97})^{(1/3)}}{12} - \frac{27}{4\,(1017+72\sqrt{97})^{(1/3)}} \right) \right]$$

Válasszuk ki a valós gyököket:

```
> valos:=select(type,gyok,realcons);
```

$$valos := \left[\frac{(1017+72\sqrt{97})^{(1/3)}}{12} + \frac{27}{4\,(1017+72\sqrt{97})^{(1/3)}} + \frac{3}{4} \right]$$

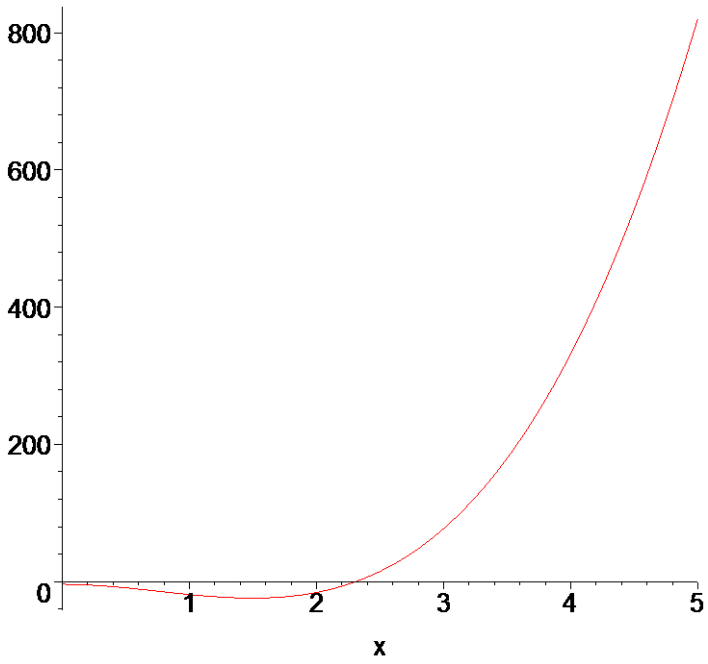
Adjuk meg a gyök közelítő értékét:

```
> zh:=evalf(valos);
```

$$zh := [2.312341166]$$

Rajzoljuk föl a deriváltat pozitív x-ekre:

```
> plot(D(g)(x),x=0..5);
```



>

A különbség-függvénynek a $zh = 2.312341166$ helyen minimuma van.

Keresük meg a különbsé-függvény zérushelyét:

```
> zhg:=fsolve(g(x),x=3..5);
```

$$zhg := 3.201233421$$

Tehát a k = 3.201233421 is megfelel.

2. Feladat: A definíciót felhasználva mutassa meg, hogy $2^x + 17 \ \ O(3^x)$.

4. Példa: Mutassuk meg, hogy $\frac{x^2+1}{x+1} \ O(x)$.

Megoldás: Bontsuk föl a függvényt polinom és valódi tört összegére:

$$\frac{x^2+1}{x+1} = \frac{x^2-1+2}{x+1} = \frac{x^2-1}{x+1} + \frac{2}{x+1} = x - 1 + \frac{2}{x+1}.$$

Az átalakítást polinomosztással is elvégezhettük volna

```
> tort:=(x^2+1)/(x+1);
```

$$tort := \frac{x^2+1}{x+1}$$

```
> hanyados:=quo(numer(tort),denom(tort),x,'r');
```

$$hanyados := x - 1$$

```
> tort=hanyados+r/denom(tort);
```

$$\frac{x^2+1}{x+1} = x - 1 + \frac{2}{x+1}$$

Ebből az alakból látható, hogy ha $1 < x$, akkor a függvény kisebb x-nél, tehát a függvény $O(x)$.

3. Feladat: Mutassuk meg, hogy $\frac{x^3+2x}{2x+1} \ O(x^2)$.

5. Példa: Keressük meg azt a legkisebb egészt, amelyre $f(x)$ $O(x^n)$ az alábbi f függvényekre:

a) $f(x) = 2x^3 + x^2 \ln(x)$; b) $f(x) = 3x^3 + \ln(x)^4$;

c) $f(x) = \frac{x^4 + x^2 + 1}{x^3 + 1}$; d) $f(x) = \frac{x^4 + 5\ln(x)}{x^4 + 1}$.

Megoldás: Mivel $\ln(x) < x$ minden pozitív x értékre

a) $2x^3 + x^2 \ln(x) < 2x^3 + x^3 = 3x^3$, s így $f(x)$ $O(x^3)$, tehát $f(x)$ $O(x^3)$, de ha $n < 3$, akkor már nem teljesül f-re $O(x^n)$, vagyis $n = 3$.

b) Az előzőhöz hasonló gondolatmenettel $n = 3$.

c) Végezzük el a polinomosztást:

```
> tort:=(x^4+x^2+1)/(x^3+1);
```

$$tort := \frac{x^4 + x^2 + 1}{x^3 + 1}$$

```
> hanyados:=quo(numer(tort),denom(tort),x,'r');
```

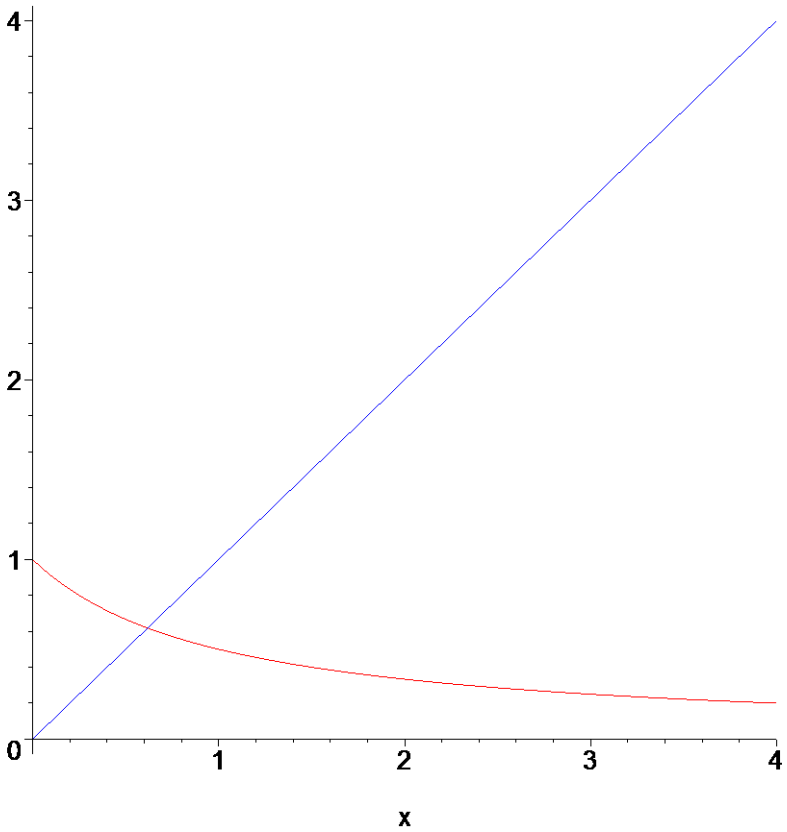
$$hanyados := x$$

```
> tort=hanyados+r/denom(tort);
```

$$\frac{x^4 + x^2 + 1}{x^3 + 1} = x + \frac{x^2 + 1 - x}{x^3 + 1}$$

A második tag, a $\frac{x^2 + 1 - x}{x^3 + 1}$ kifejezés valódi tört, tehát elég nagy x értékre kisebb x-nél, sőt bármely rögzített pozitív számnál. Szemléltesük is ezt a tényt:

```
> plot([x,(x^2+1-x)/(x^3+1)],x=0..4,color=[blue,red]);
```



Tehát $f(x)\ O(x)$

d) Bontsuk szét a törtet:

$$\frac{x^4+5\ln(x)}{x^4+1}=\frac{x^4+1+\ln(x)-1}{x^4+1}=1+\frac{\ln(x)-1}{x^4+1}, \text{ ahol}$$

$$\frac{\ln(x)-1}{x^4+1}<\frac{x-1}{x^4+1}<\frac{x^4+1}{x^4+1}=1, \text{ ha } 1<x.$$

Tehát $f(x)\ O(1)$. Másképpen $n=0$.

4. Feladat: Keressük meg azt a legkisebb egészt, amelyre $f(x)\ O(x^n)$ az alábbi f függvényekre:

a) $f(x)=2x^2+x^3\ln(x)$; b) $f(x)=3x^5+\ln(x)^4$;

c) $f(x)=\frac{x^4+x^2+1}{x^4+1}$; d) $f(x)=\frac{x^3+5\ln(x)}{x^4+1}$.

6. Példa: Mutassuk meg, hogy $x^2+4x+17\ \ O(x^3)$, de x^3 nem $O(x^2+4x+17)$.

Megoldás: Egyrészt $x^2+4x+17\le x^2+x^2+x^2=3x^2\le 3x^3$ minden $17<x$ esetén,tehát $x^2+4x+17\ \ O(x^3)$. Másrészt ha $x^3\ \ O(x^2+4x+17)$ volna, akkor $x^3\le C(x^2+4x+17)<=3Cx^2$ teljesülne minden elég nagy x értékre. Ez azt jelentené, hogy $x\le 3C$ teljesülne, ami nyílván lehetetlen állandó C-re és elég nagy x-re.

5. Feladat: Mutassa meg, hogy $3x^4+1\ \ O\left(\frac{x^5}{10}\right)$, de $\frac{x^5}{10}$ nem $O(3x^4+1)$.

7.Példa: Mutassuk meg, hogy $3x^4+1\ \ O\left(\frac{x^4}{2}\right)$ és $\frac{x^4}{2}\ \ O(3x^4+1)$.

Megoldás: Az első részhez: $3x^4+1\le 4x^4=8\frac{x^4}{2}$,minden $1<x$ esetén. A második részhez:

$\frac{x^4}{2}\le 3x^4<=1(3x^4+1)$ minden x -re.

6. Feladat: Mutassa meg, hogy $x\ln(x)\ \ O(x^2)$, de x^2 nem $O(x\ln(x))$.

8. Példa: Adjuk meg a lehető legjobb nagy O becslést a következő függvényekre:

a) $(n^2+8)(n+1)$; b) $(n\ln(n)+n^2)(n^3+2)$; c) $(n!+2^n)(n^3+\ln(n^2+1))$.

Megoldás:

a) A meghatározó tag az n^2 n-nel való szorzata, tehát a függvény $O(n^3)$.

Részletezve $(n^2+8)(n+1)=n^3+n^2+8n+8\le 4n^3$, ha $1\le n$. Tehát nem túl szigorú becslést végezve $k=1$ és $C=4$ választásigazoltuk a függvényről, hogy $O(x^3)$.

b) Mivel $n\ln(n)<n$, az első tényező meghatározó tagja az n^2, míg a második tényezőé az n^3. Így a függvény $O(n^5)$. Ez a definíció alapján - mint tudjuk - azt jelenti, hogy

$$(n\ln(n)+n^2)(n^3+2)<Cn^5 \qquad (0<x)$$

Ez pedig $0<x$-ra egyenértékű a

$$\frac{(n\ln(n)+n^2)(n^3+2)}{n^5}<C$$

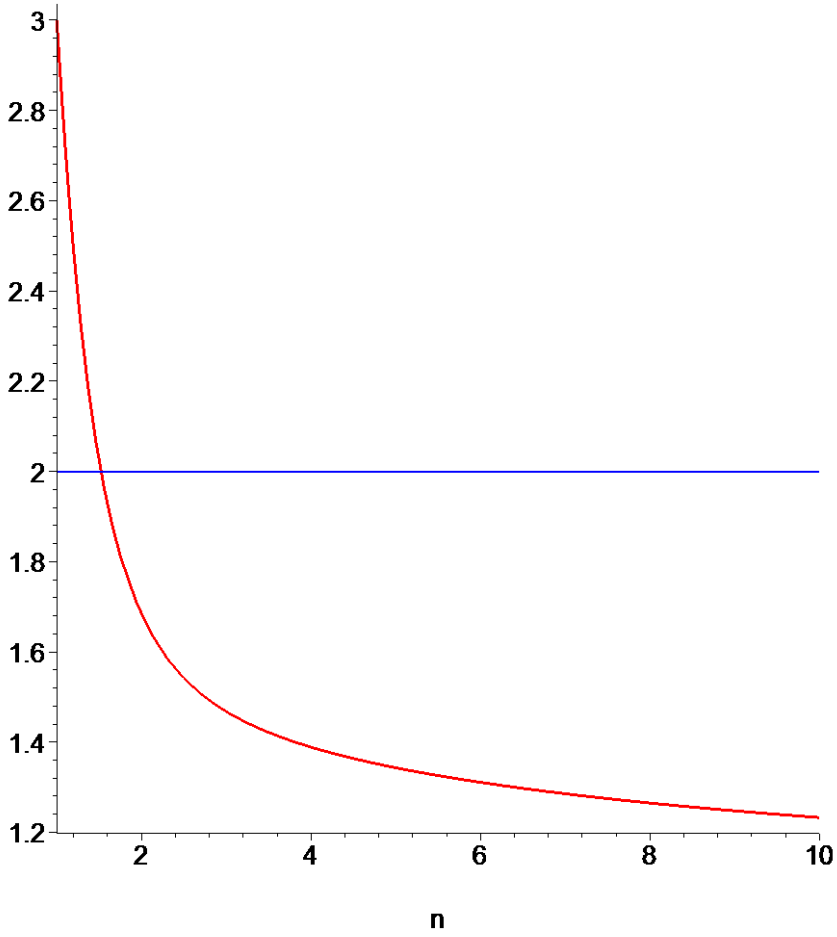
A tört végtelenben vett határértéke:

```
> Limit((n*ln(n)+n^2)*(n^3+2)/(n^5),n=infinity)=limit((n*ln(n)+
  n^2)*(n^3+2)/(n^5),n=infinity);
```

$$\lim_{n \to \infty} \frac{(n \ln(n) + n^2)(n^3 + 2)}{n^5} = 1$$

Ez pedig azt jelenti, hogy minden $1 < C$ megfelel. Legyen pl. $C = 2$.

```
> plot([2,(n*ln(n)+n^2)*(n^3+2)/(n^5)],n=1..10,color=[blue,red]
  ,thickness=[2,3]);
```




```
>
```

Láthatóan a hányados "elég hamar " kisebb lesz mint 2.

A példából látható, hogy ha $\lim_{x \to \infty} \left| \frac{f(x)}{g(x)} \right| = C$, ahol C valós szám, akkor $f(x) \ O(g(x))$

c) Az első tényezőt illetően megjegyezzük, hogy

$$2^n < n!,$$

ha $4 \le n$, ezért az első tényező meghatározó tagja az $n!$. A második tényező meghatározó tagja az

n^3. Ezért a függvény $O(n!\, n^3)$

9. Példa: Adjunk nagy O becslést az alábbi függvények mindegyikére. Az f függvényt $O(g)$ becslő függvény legyen a lehető legegyszerűbb és a legkisebb rendű!

a) $n \ln(n^2 + 1) + n^2 \ln(n)$;

b) $(n \ln(n)+1)^2 + (\ln(n)+1)(n^2+1)$;

c) $n^{(2^n)} + n^{(n^2)}$;

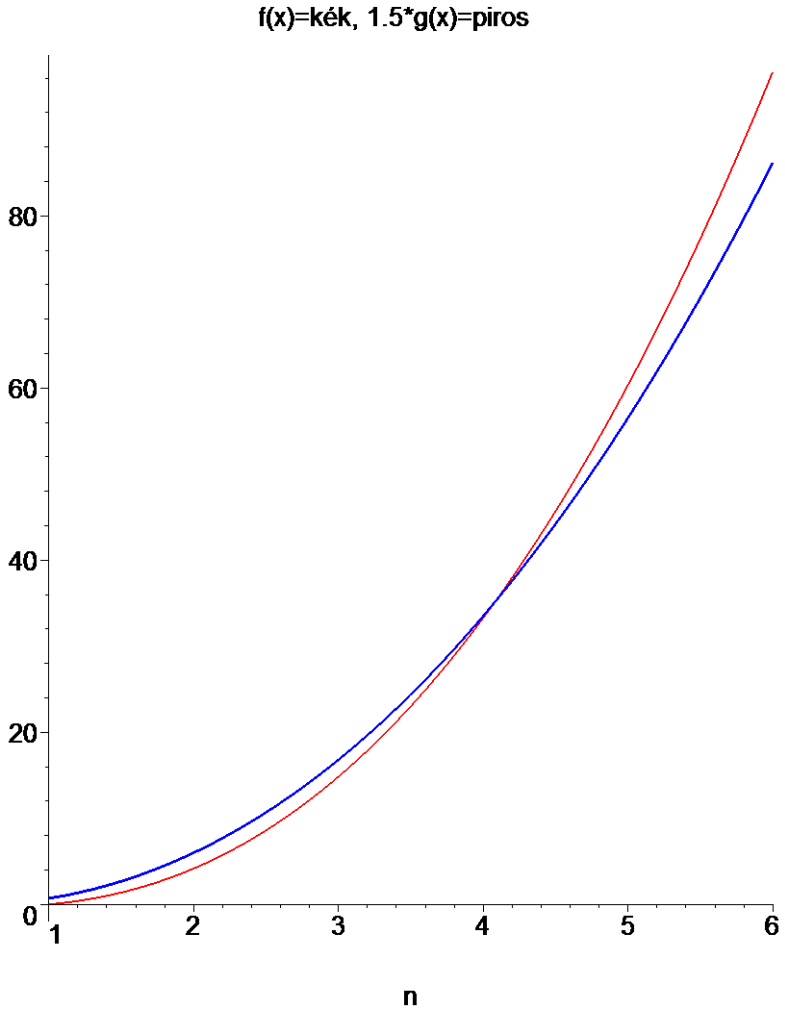
Megoldás:

a) Mindenekelőtt megjegyezzük, hogy az $\ln(n^2+1)$ és az $\ln(n)$ azonos nagy O osztályba tartoznak, mi mivel $\ln(n^2) = 2\ln(n)$ $(0 < n)$. Ezért itt a második tag gyorsabban növekszik és így

a legegyszerűbb pontos becslés: $O(n^2 \ln(n))$

Az alábbi ábra alapján úgy tűnik, hogy...

```
> plot([n*ln(n^2+1)+n^2*ln(n),1.5*n^2*ln(n)],n=1..6,color=[blue
  ,red],thickness=[3,2],title=`f(x)=kék, 1.5*g(x)=piros`);
```



... $C = 1.5$ és k = 4.5 megfelel.

Pontosabban, vegyük az 1.5 $g(x)$ – $f(x)$ különbségfüggvényt. Ennek zérushelye.

```
> gyok:=solve(n*ln(n^2+1)+n^2*ln(n)=1.5*n^2*ln(n));
```

$$gyok := 4.082826035$$

```
> f:=n->1.5*n^2*ln(n)-n*ln(n^2+1)-n^2*ln(n);
```

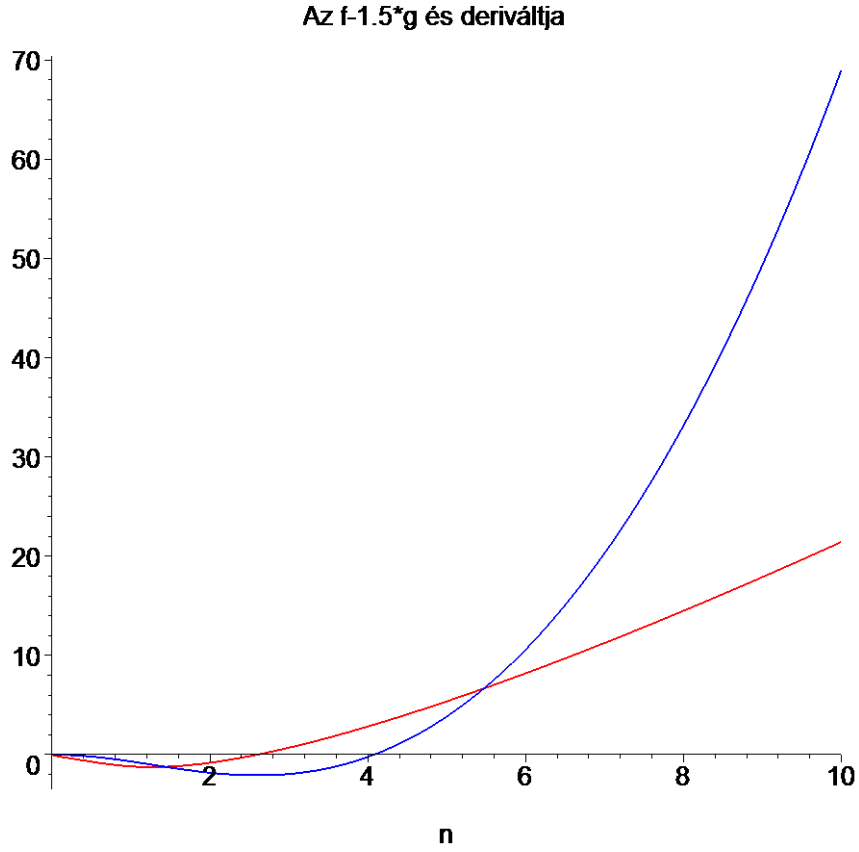
$$f := n \to 1.5\, n^2 \ln(n) - n \ln(n^2+1) - n^2 \ln(n)$$

A különbségfüggvény lehetséges szélsőértékhelye:

```
> solve(D(f)(n));
```

$$2.604287102$$

```
> plot([f(n),D(f)(n)],n=0..10,color=[blue,red],thickness=[2,2],
  title=`Az f-1.5*g és deriváltja`);
```




```
>
```

A függvény és deriváltja viselkedése alapján a becslés jó volt, sőt $C = 1.5$ mellett $k = 4.1$ is jó.

b) Az első tag $n^2 \ln(n)^2$-tel egy osztályba tartozik nagy O, míg a második tag némileg kisebb osztályba, vagyis az $O(n^2 \ln(n))$ osztályba. (Minden esetben elhagyhatjuk az alacsonyabb rendű tagokat, mert ezeket "felülbírálják" a megtartott tagok- éppen ez a nagy O becslés lényege).

Ennélfogva a válasz $O(n^2 \ln(n)^2)$

c) Itt az egyetlen kérdés, hogy a 2^n vagy az n^2 kifejezés nő-e gyorsabban. Természetesen az első, így a legjobb nagy O becslés: $O(n^{(2^n)})$.

7. Feladat: Adjunk nagy O becslést az alábbi függvények mindegyikére.Az f függvénytO(g) becslő függvény legyen a lehető legegyszerűbb és a legkisebb rendű!

a) $(n^3 + n^2 \ln(n))(\ln(n) + 1) + (17 \ln(n) + 19)(n^3 + 2)$;

b) $(2^n + n^2)(n^3 + 3^n)$;

c) $(n^n + n\,2^n + 5^n)(n! + 5^n)$.

10. Példa: Az első feladat minden függvényére vonatkozólag döntsük el, hogy a függvény $\Omega(x^2)$ vagy $\Theta(x^2)$!

Megoldás:

a) $f(x) = 17x + 11$. Ez a függvény $\Theta(x)$, tehát nem $\Theta(x^2)$, mivel az x^2 gyorsabban nő, mint az x. Pontosabban fogalmazva x^2 nem $O(17x + 11)$. Ugyanígy indokolható, hogy nem $\Omega(x^2)$.

b) Az $f(x) = x^2 + 1000 \quad \Theta(x^2)$; ugyanis a " +1000 " elhanyagolható, mivel ez alacsonyabb rendű tag. Természeesen, mivel $f(x)\ \Theta(x^2)$ egyuttal $\Omega(x^2)$ is.

c) Az $f(x) = x \ln(x)$ lassabban nő, mint az x^2 függvény, mivel az $\ln(x)$ lassabban nő az x függvénynél. Ezért az f függvény nem $\Theta(x^2)$ és nem is $\Omega(x^2)$.

d) Az $f(x) = \frac{x^4}{2}$ gyorsaban nő, mintaz x^2. Ezért $f(x)$ nem $\Theta(x^2)$ de $\Omega(x^2)$.

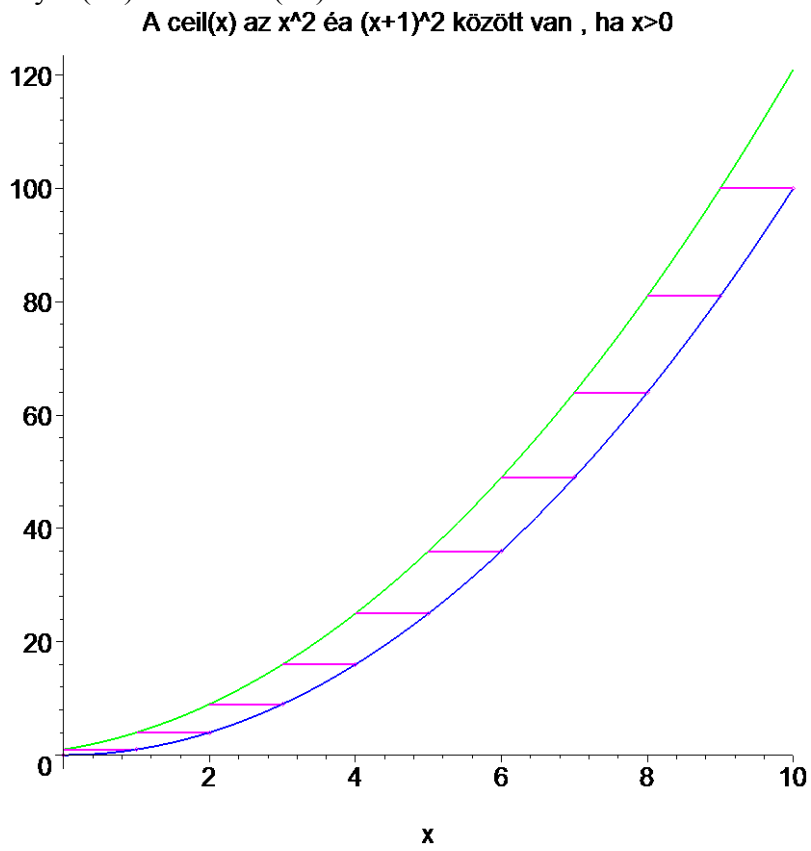
e) Az 1-nél nagyobb alapú exponenciális függvény bármely polinomnál gyorsabban nő, ezért ez a függvény nem $\Theta(x^2)$ de $\Omega(x^2)$.

f) Az $f(x) = \text{ceil}(x)^2$nagy x értékekre közel van az x^2 függvényhez. Pontosabban:

$x^2 \le \text{ceil}(x)^2 <= (x+1)^2$, ha $0 < x$.

```
> plot([x^2,
  ceil(x)^2,(x+1)^2],x=0..10,color=[blue,magenta,green],discont
  =true,thickness=2,title=`A ceil(x) az x^2 éa (x+1)^2 között
  van , ha x>0`);
>
```

Ezért az f függvény $\Theta(x^2)$ és ezért $\Omega(x^2)$ is.




```
>
>
```

8. Feladat: Mutassa meg, hogy

a) $3x + 7 \quad \Theta(x)$;

b) $2x^2 + x - 7 \quad \Theta(x^2)$;

c) $\ln(x^2 + 1) \quad \Theta(\log_2(x))$;

d) $\log_{10}(x) \quad \Theta(\log_2(x))$.