1. Adjon meg a $10, 3, 5, 2, 7, 1, 18, 4, 12, 17, 6$ sorozatban egy leghosszabb növekvő részsorozatot az órán tanult dinamikus programozást használó algoritmussal!

   Megoldás:

$i =$	1	2	3	4	5	6	7	8	9	10	11
$A[i] =$	10	3	5	2	7	1	18	4	12	17	6
$L[i] =$	1	1	2	1	3	1	4	2	4	5	3
előző$[j] =$	-	-	2	-	3	-	5	2	5	9	3

   Leghosszabb növekvő részsorozat hossza: max{1,1,2,1,3,1,4,2,4,5,3} = 5. A maximum $i = 10$-nél van.
   előző[10] = 9, előző[9] = 5, előző[5] = 3, előző[3] = 2,
   egy maximális növekvő részsorozat $A[2]$,$A[3]$,$A[5]$,$A[9]$,$A[10]$, azaz 3,5,7,12,17.

2. Egy f fokú létrán bizonyos fokok annyira rozogák, hogy ha rálépünk, leszakadnak. Szerencsére tudjuk, hogy melyik fokok ilyenek, hova nem szabad lépnünk. Egy lépéssel legfeljebb 3 fokot tudunk lépni. Adjon dinamikus programozást használó algoritmust ami meghatározza, hogy (a) a létra aljától fel tudunk-e jutni a létra legfelső fokára! (b) a létra aljától hányféleképpen tudunk feljutni a létra legfelső fokára! *(Feltehető, hogy a legfelső fokra rá szabad lépni.)* Mennyi az algoritmusok lépésszáma?

   Megoldás: Legyen NR az a tömb amiben $NR[i]$ akkor igaz, ha az i-edik fok nem rozoga.

   (a) A részfeladatok azok lesznek, hogy az i-edik fokra fel tudunk-e jutni ($1 \leq i \leq f$). Az $F[i]$ táblázatot töltjük ki úgy, hogy $F[i]$ igaz, ha fel tudunk az i-edik fokra jutni.

   Vegyük észre, hogy a feladat az $F[f]$ értéket kérdezi.

   A kitöltés kezdete: $F[0]$ = igaz, azaz a földről nem esünk le, $F[1] = NR[1]$.

   A rekurzió: mivel az i-edik fokra eggyel, kettővel vagy hárommal lejjebbről jöhetünk, pontosan akkor tudunk az i-edikre lépni, ha a három megelőző valamelyikére el lehet jutni és az i-edik fok nem rozoga. Ezért az általános rekurzió: $F[i] = NR[i] \wedge (F[i-1] \vee F[i-2] \vee F[i-3])$.

   Látszik, hogy ez csak $i \geq 3$ esetben működik. Az $i = 2$ esetre hasonló formula jó, csak ott nem szerepel a nem létező $F[i-3]$, azaz $F[2] = NR[2] \wedge (F[1] \vee F[0])$.

   Ha a táblázatot $i = 0, 1, 2, \ldots, f$ sorrendben töltjük ki, akkor minden elemnél konstans sok lépést használunk, az algoritmus lépésszáma $O(n)$.

   (b) Hasonló az (a) részhez, de most nem logikai változó kell. Legyen $M[i]$ az, hogy hányféleképpen tudunk feljutni az i-edik fokra.

   A feladat megoldását $M[f]$ szolgáltatja.

   A kitöltés kezdete: legyen minden $M[i] = 0$ ha $i > 0$ és $M[0] = 1$. Mivel az i-re lépés csak az előző három valamelyikéről történhet, ezért a lehetséges feljutások számát a három előző szám összegeként kapjuk meg. Ebben úgy vesszük figyelembe a rozoga fokokat, hogy azoknál az M értéke 0 lesz, tehát nem adnak semmit a keresett számhoz. Ezek alapján:

   Ha $NR[1]$ igaz, akkor legyen $M[1] = 1$, ha $NR[2]$ igaz, akkor $M[2] = M[1] + M[0]$ és általában, ha $NR[i]$ igaz, akkor legyen $M[i] = M[i-1] + M[i-2] + M[i-3]$.

   Most is $i = 0, 1, 2, \ldots, f$ sorrendben kell kitölteni a táblázatot és akkor a használni kívánt értékek már rendelkezésünkre állnak, minden i-re konstans sok **összeadás** lesz, azaz összesen $O(n)$ **összeadás**, az egyszerűség kedvéért ezt tekintjük a lépésszámnak is.

   Ha azt is figyelembe vesszük, hogy az összeadandó számok nem feltétlenül konstans méretűek, akkor lépésszám nagyobb lesz. Indukcióval könnyen belátható azonban, hogy a legnagyobb szám is legfeljebb 3^n. Ennek leírásához $O(n)$ számjegy elegendő. Mivel az összeadást a jegyek számában lineáris algoritmussal el tudjuk végezni, az algoritmus teljes lépésszáma $O(n^2)$.

3. Adott $n+1$ darab ($n \geq 1$) pozitív egész szám: $s_1, s_2, \ldots, s_n, b$ és azt szeretnénk eldönteni, hogy előáll-e a b szám néhány s_i szám összegeként. (Mindegyik s_i legfeljebb egyszer használható fel az összegben.) Adjunk dinamikus programozást használó $O(n \cdot b)$ lépésszámú algoritmust erre a feladatra (melynek neve Részhalmaz-összeg probléma, rövidítése RH) az alábbi részfeladatok felhasználásával:
$L(i,c)$ jelentése ($0 \leq i \leq n$ és $0 \leq c \leq b$ esetén) legyen az, hogy elő lehet-e állítani az $s_1, \ldots, s_i$ számokból a c számot.

   Megoldás:

   1. Részfeladatok: adottak a feladatban, azaz $L(i,c) = 0$, ha c nem áll elő és 1, ha előáll $s_1, s_2, \ldots, s_i$-ből összegként. (Lényegében egy $(n+1) \cdot (b+1)$ méretű táblázatot töltünk ki, aminek $L(i,c)$ áll az i. sorának c. cellájában.)
   2. Sorrend: i megy 0-tól n-ig és egy belső ciklusban minden i-re c megy 0-tól b-ig (vagyis soronként haladunk a táblázatban fentről lefele, a soron belül balról jobbra).
   3. Eleje: $L(i,0) = 1$ minden i-re, mert a 0-t mindig ki lehet úgy, hozni, hogy egy számot se választunk ki és $L(0,c) = 0$, ha $c > 0$, mert ha egy számot se választhatunk, akkor 0-nál nagyobb összeg nem tud kijönni. (Azaz a táblázat első oszlopa csupa 1-es, az első sor többi eleme pedig csupa 0.)
   4. Továbblépés, ha $i \geq 1$ és $c \geq 1$: $L(i,c) = 1$ pontosan akkor, ha $L(i-1,c) = 1$ vagy ha $L(i-1, c-s_i) = 1$ (ez utóbbi eset csak akkor játszik, ha $c - s_i \geq 0$). Ez azért helyes, mert c vagy s_i nélkül áll elő, csupán $s_1, \ldots, s_{i-1}$ közül néhányból vagy s_i-t is használjuk és ekkor $c - s_i$-t kell előállítani $s_1, \ldots, s_{i-1}$ közül néhányból.
   5. Végeredmény: nekünk $L(n,b)$ kell, itt derül ki, hogy ki lehet-e hozni b-t az s_i számokból. (Vagyis a táblázat jobb alsó elemét akarjuk.)
   6. LSZ: Minden részfeladatnál konstans sok lépés van és $(n+1) \cdot (b+1)$, azaz $O(n \cdot b)$ részfeladat van.
   7. Jóság: mindegyik résznél megnéztük (lásd feljebb)
   8. Mik a számok, amikből kijön az összeg: azt kell meghatározni, hogy melyik s_i-k összegeként áll elő b, ha előáll egyáltalán. Itt elég egy jó részhalmazát megadni az s_i számoknak és ezt úgy lehet, hogy $L(i,c)$ számolásakor feljegyezzük, hogy az első (s_i nélkül) vagy a második eset (s_i-vel) miatt állt elő c (ha mindkettőnél előáll, akkor az egyik esetet vesszük csak) és $L(n,b) = 1$ esetén visszafelé haladva minden s_i-ről kiderül, hogy kell vagy nem kell b-hez. Itt minden lépésben csökkenik i értéke eggyel, azaz ez a visszalépkedés (mivel csak egy jó részhalmazt keresünk, azaz csak egy utat akarunk megtalálni) $O(n)$ lépésben megvan.

4. Egy $m \times m$ méretű táblázat mezőin lépkedünk a bal alsó sarokból a jobb felső sarokba úgy, hogy egy lépésben a táblázatban vagy felfelé vagy jobbra egyet lépünk, de van néhány „tiltott" mező, ahova nem léphetünk. Adjon egy dinamikus programozást használó eljárást, ami meghatározza, hogy hányféleképpen érhetünk célba!

   Megoldás: Definiáljunk egy $m+1$ sorból és oszlopból álló T táblázatot, amelyben a $T[i,j]$ elem jelentése, hogy hányféleképpen tudunk eljutni az (i,j) mezőbe. (A bal alsó sarok az (1,1) mező, a jobb felső az (m,m).)

   Legyenek a T 0. sorában és oszlopában az értékek nullák, $T[1,1] = 1$.

   A rekurziós formulához vegyük észre, hogy az (i,j) mezőbe csak az $(i-1,j)$ vagy az $(i,j-1)$ mezőkből léphetünk és ha T-ben a tiltott mezőknek megfelelő értékeket 0-ra állítjuk, akkor a

$$T[i,j] = \begin{cases} T[i-1,j] + T[i,j-1], & \text{ha } (i,j) \text{ nem tiltott} \\ 0, & \text{ha } (i,j) \text{ tiltott} \end{cases}$$

   formula $i \geq 1$, $j \geq 1$ esetén helyes lesz.

   Ez a táblázat kitölthető soronként, hiszen akkor az (i,j)-hez érve a két felhasználandó érték már rendelkezésünkre áll.

Lépésszám: a táblázat mérete $(m+1)^2 \in O(n)$, hiszen az input mérete $n = m^2$. Minden elemhez konstans sok összeadás szükséges, ezért az összeadások száma is $O(n)$. Itt is figyelnünk kell azonban a számok méretére. Ha egyetlen tiltott mező sincs, akkor a legnagyobb szám $\binom{2m}{m}$. Mivel $\sum_{i=1}^{2m}\binom{2m}{i} = 2^{2m}$, ezért $\binom{2m}{m} \leq 2^{2m}$. Ennek jegyeinek száma $O(m)$. Így a lépésszám $O(m^3)$, azaz $O(n^{1,5})$.

Megjegyzés: A továbbiakban az egyszerűség kedvéért feltesszük, hogy aritmetikai műveleteket 1 lépésben el tudunk végezni tetszőleges nagyságú számokkal.

5. Legyen $s_1 s_2 \dots s_n$ és $t_1 t_2 \dots t_m$ egy n és egy m hosszú karaktersorozat. Azt szeretnénk, hogy az $n \times m$ méretű A mátrix $A[i,j]$ eleme tartalmazza azt a legnagyobb k számot, melyre az $s_1 s_2 \dots s_i$ és a $t_1 t_2 \dots t_j$ sorozatok utolsó k karaktere megegyezik. Adjon eljárást, ami az A tömböt $O(nm)$ lépésben kitölti.

Megoldás: Ha $i = 1$ vagy $j = 1$, akkor egyszerű, hiszen az egyik sorozat 1 elemű, az érték csak 0 vagy 1 lehet.

$$A[1,j] = \begin{cases} 1, & \text{ha } s_1 = t_j \\ 0, & \text{ha } s_1 \neq t_j \end{cases} \qquad A[i,1] = \begin{cases} 1, & \text{ha } s_i = t_1 \\ 0, & \text{ha } s_i \neq t_1 \end{cases}$$

A továbbiakhoz azt kell észrevenni, hogy ha az s_i és t_j különbözik, akkor nyilván $k = 0$, egyébként pedig a leghosszabb ilyen közös rész az eggyel rövidebb kezdőszeletekre kapott egyező rész kiterjesztése,

$$A[i,j] = \begin{cases} 1 + A[i-1, j-1], & \text{ha } s_i = t_j \\ 0, & \text{ha } s_i \neq t_j \end{cases}$$

Amennyiben a táblázatot soronként töltjük ki, akkor mindig rendelkezésünkre áll a szükséges információ, egy elem kiszámolása konstans művelet, a táblázat mérete nm, ezért az egész összesen $O(nm)$.

6. Egy $n \times n$ méretű táblázat minden eleme egy pozitív egész szám. A táblázat bal alsó sarkából akarunk eljutni a jobb felső sarkába úgy, hogy egy lépésben a táblázatban vagy felfelé vagy jobbra egyet lépünk. Azt szeretnénk, hogy a lépegetés során látott elemek növekvő sorrendben kövessék egymást. Adjon $O(n^2)$ lépésszámú algoritmust, ami meghatározza, hogy (a) hány a szabályoknak megfelelő út van! (b) mekkora a legnagyobb értékű, a szabályoknak megfelelő út, ha egy út értéke a benne szereplő számok szorzata!

Megoldás: (a) Dinamikus programozást használunk: Jelölje A az eredeti (adott) táblázatot. Készítünk egy $n \times n$ méretű T táblázatot, amiben a $T[i,j]$ értéke az lesz, hogy hányféleképpen tudunk a szabályoknak megfelelően A-ban az i-edik sor j-edik elemébe jutni. Ha $A[1,1]$ a bal alsó mező és $A[n,n]$ a jobb felső, ekkor $T[n,n]$ adja a keresett választ.

Legyen $T[1,1] = 1$. Mivel az (i,j) pozícióba csak balról vagy lentről jöhetünk, akkor van jó út, ha ezek közül legalább az egyikbe el lehet jutni, és onnan az utolsó lépés is érvényes, azaz teljesül, hogy az $A[i,j]$ nagyobb a megelőző elemnél.

Ezért az első sor $j > 1$ esetben $T[1,j] = T[1,j-1]$, ha $A[1,j] > A[1,j-1]$, különben meg $T[1,j] = 0$. (A sor egy darabig 1 lesz, egy idő után meg 0, ha volt egy nem növekvő lépés.) Hasonlóan. az $i > 1$ esetben $T[i,1] = T[i-1,1]$, ha $A[i,1] > A[i-1,1]$, különben meg $T[i,1] = 0$.

Az általános rekurzió, amikor $i, j > 1$:

- ha $A[i,j] > A[i-1,j]$ és $A[i,j] > A[i,j-1]$, akkor $T[i,j] = T[i-1,j] + T[i,j-1]$
- ha $A[i,j] > A[i-1,j]$ és $A[i,j] \leq A[i,j-1]$, akkor $T[i,j] = T[i-1,j]$
- ha $A[i,j] \leq A[i-1,j]$ és $A[i,j] > A[i,j-1]$, akkor $T[i,j] = T[i,j-1]$
- ha $A[i,j] \leq A[i-1,j]$ és $A[i,j] \leq A[i,j-1]$, akkor $T[i,j] = 0$.

Ha az első sor és oszlop meghatározása után a táblázatot soronként töltjük ki, akkor mindig rendelkezésünkre áll a szükséges információ, így minden elem kiszámolása konstans sok műveletet fog használni.

Mivel a T tömb mérete n^2 és minden elemének kitöltése konstans sok művelet, ezért a lépésszám $O(n^2)$.

(b) Az előzőhöz hasonló, de most a maximális értéket tároljuk, ezért minden érvényes lépésnél szorozni kell az aktuális A értékkel, az összeadás helyett pedig a nagyobb értéket kell választani.

Ha ez most egy S tömb, akkor $S[1,1] = A[1,1]$, mivel eddig ez az érték. Az első sor és oszlop kitöltése így alakul: $S[1,j] = S[1,j-1] \cdot A[1,j]$, ha $A[1,j] > A[1,j-1]$, különben meg $S[1,j] = 0$. Hasonlóan az $i > 1$ esetben $S[i,1] = S[i-1,1] \cdot A[i,1]$, ha $A[i,1] > A[i-1,1]$, különben meg $S[i,1] = 0$.

A továbbiakban $i, j > 1$, ekkor pedig

- ha $A[i,j] > A[i-1,j]$ és $A[i,j] > A[i,j-1]$, akkor $S[i,j] = A[i,j] \cdot \max\{S[i-1,j], S[i,j-1]\}$
- ha $A[i,j] > A[i-1,j]$ és $A[i,j] \leq A[i,j-1]$, akkor $S[i,j] = A[i,j] \cdot S[i-1,j]$
- ha $A[i,j] \leq A[i-1,j]$ és $A[i,j] > A[i,j-1]$, akkor $S[i,j] = A[i,j] \cdot S[i,j-1]$
- ha $A[i,j] \leq A[i-1,j]$ és $A[i,j] \leq A[i,j-1]$, akkor $S[i,j] = 0$.

Ha az első sor és oszlop meghatározása után a táblázatot soronként töltjük ki, akkor mindig rendelkezésünkre áll a szükséges információ, így minden elem kiszámolása konstans sok műveletet fog használni. Ezért a teljes lépésszám most is $O(n^2)$.

---

7. Van n fájlunk, az i-edik fájl hosszát jelölje a h_i. Tegyük fel, hogy a h_i számok pozitív egészek. Mentéshez két egyformán L méretű lemez áll rendelkezésünkre (L pozitív egész szám). A cél, hogy minél nagyobb k számra az első k darab fájl mindegyikét mentsük ki a lemezekre (a fájlok sorrendje rögzített). Fájlokat szétvágni nem szabad, minden fájl teljes egészében kerül az egyik vagy a másik lemezre. Adjon algoritmust, ami adott L és h_i számokhoz meghatározza, hogy melyik fájlt melyik lemezre tegyük ahhoz, hogy k a lehető legnagyobb legyen. Az algoritmus lépésszáma legyen $O(L^2)$.

   Megoldás: Feltehető, hogy $n \leq 2L$ mert ennél több pozitív hosszúságú állományt biztosan nem lehet elhelyezni $2L$ területen.

   Egy irányított gráfot konstruálunk. Ennek csúcsai (a,b,k) alakú hármasok, ahol a,b,k nem negatív egészek, $k \leq n$, $a \leq L$, $b \leq L$, és $a + b = h_1 + \cdots + h_k = s_k$. A kezdőcsúcs (0,0,0). Az (a,b,k) csúcsok közül bizonyosak (nem feltétlenül mind) annak felelnek meg, hogy az első k állományt el tudjuk elhelyezni a két lemezre úgy, hogy a mennyiségű adatot tudunk a szabályok szerint eltenni az első lemezre, b mennyiséget a másodikra.

   Az (a,b,k) csúcsból legfeljebb két él indul ki: egy $(a + h_{k+1},b,k + 1)$-be és egy $(a,b + h_{k+1},k + 1)$-be, amennyiben ezek létező csúcsok.

   A kapott gráf nyilván DAG: él mentén a harmadik komponens nő. Egy adott k-ra legfeljebb $L + 1$ csúcs lehet, amelynek a 3. komponense k, hiszen $0 \leq a \leq L$ és s_k értéke meghatározza b értékét is. Így a gráfnak összesen legfeljebb $(L + 1)(L + 1) = O(L^2)$ csúcsa van, az élek száma legfeljebb a csúcsszám kétszerese, ezért az éllistás megadás mérete $O(L^2)$. Ekkora költséggel tudunk keresni a gráfban $(0, 0, 0)$-ból induló leghosszabb utat. Egy ℓ élű ilyen út, ha végigmegyünk rajta, éppen az első ℓ állomány szabályok szerinti elhelyezését adja.

8. Egy n és egy m karakterből álló szövegben meg akarjuk találni a legnagyobb azonos darabot, azaz ha az egyik szöveg $a_1a_2 \cdots a_n$ és a másik $b_1b_2 \cdots b_m$, akkor olyan $1 \leq i \leq n$ és $1 \leq j \leq m$ indexeket keresünk, hogy $a_{i+1}a_{i+2} \ldots a_{i+t} = b_{j+1}b_{j+2} \ldots b_{j+t}$ teljesüljön a lehető legnagyobb t számra. Adjon erre a feladatra $O(nm)$ lépést használó algoritmust.

   Megoldás: Vegyük észre, hogy ha a 5. feladatban leírt táblázatot elkészítjük erre az esetre, akkor a táblázatbeli legnagyobb érték pont a megfelelő t lesz. Ennek helye megadja a megfelelő $i + t$ és $j + t$ indexeket, amikből i és j is megkapható.

   A táblázat kitöltése az előző feladat szerint $O(nm)$ lépés. A maximális érték megtalálása további $nm-1$ összehasonlítással megoldható, ezek után i és j kiszámolás konstans lépés. Ez összesen is $O(nm)$.

9. Adott egy n és egy m hosszú 0-1 sorozat, $a_1, a_2, \ldots, a_n$, illetve $b_1, b_2, \ldots, b_m$. Ezek alapján egy T tömböt töltöttünk ki a következő módon:

   Ha $0 \leq i \leq n$, akkor $T[i, 0] = 0$. Ha $0 \leq j \leq m$, akkor $T[0, j] = 0$.

   Ha $1 \leq i \leq n$ és $1 \leq j \leq m$, akkor $T[i, j] = \begin{cases} T[i-1, j-1] + 1 & \text{ha } a_i = b_j \\ \max\{T[i, j-1],\, T[i-1, j]\} & \text{ha } a_i \neq b_j \end{cases}$

   Mi a jelentése a $T[i, j]$ értéknek! A két sorozatnak milyen tulajdonságát adja meg a $T[n, m]$ érték?

   Megoldás: $T[i, j] =$ az $a_1 \cdots a_i$ és $b_1 \cdots b_j$ sorozatokban mekkora a legnagyobb közös részsorozat hossza, ezért $T[n, m]$ a két teljes sorozatban a leghosszabb közös részsorozat hossza.

   Ez igaz, ha $i = 0$ vagy $j = 0$. A rekurziós képlet meg azt mutatja, hogy a legnagyobb közös részsorozatban vagy $a_i = b_j$, és a korábbiakból kell a leghosszabb, ezért a hossz $T[i-1, j-1] + 1$. Ha meg $a_i \neq b_j$, akkor vagy a b_j vagy az a_i nincs benne a leghosszabb közös részsorozatban, ezt mutatja a második lehetőség.

   Megjegyzés: ilyen feladatoknál érdemes valami egyszerű példán megnézni mi történik, az alapján megtippelni a szabályt (amit persze be is kell bizonyítani).

10. Tekintsük azt a rekurzív algoritmust az n. Fibonacci-szám, F_n kiszámolására, ami $n = 0$ és $n = 1$ esetben 1-et ad vissza, $n \geq 2$ esetben pedig meghívja saját magát $n-1$ és $n-2$ inputtal, majd visszaadja az így kapott két érték összegét. Lássuk be, hogy ennek az eljárásnak a lépésszáma $\Omega(2^{\frac{n}{2}})$.

    Megoldás: Jelöljük $L(n)$-nel az algoritmus lépésszámát, ha az input mérete n. Ekkor $L(0) = L(1) = 1$, és $L(n) \geq L(n-1) + L(n-2)$, ha $n \geq 2$, ezért $k \geq 2$ esetén

$$L(2k) \geq L(2k-1) + L(2k-2) \geq L(2k-2) + L(2k-3) + L(2k-2) \geq 2L(2k-2),$$

    és ugyanez igaz $k = 1$ esetén is, hiszen ekkor $L(2k) = L(2) \geq L(1) + L(0) = 1 + 1 = 2L(0) = 2L(2k-2)$. Ez alapján ha $n \in \mathbb{N}$ páros, akkor

$$L(n) \geq 2L(n-2) \geq \ldots \geq 2^{n/2} L(0) = 2^{n/2},$$

    míg ha $n \geq 3$ páratlan, akkor

$$L(n) \geq 2L(n-2) \geq \ldots \geq 2^{(n-1)/2} L(1) = \frac{1}{\sqrt{2}} 2^{n/2}.$$

    Így tehát $L(n) \geq \frac{1}{\sqrt{2}} 2^{n/2}$, ha $n \geq 0$ (ez igaz $n = 0$ és $n = 1$ esetén is: $L(0) = 1 \geq \frac{1}{\sqrt{2}} \cdot 1$ és $L(1) = 1 \geq \frac{1}{\sqrt{2}} \cdot 2^{1/2} = 1$), tehát pl. az $c = 1/\sqrt{2}$ és $n_0 = 1$ választás mutatja, hogy $L(n) = \Omega(2^{n/2})$ (sőt $n_0 = 0$ is jó).

    2. megoldás: Bizonyíthatunk indukcióval is. Válasszunk egy olyan $c > 0$ értéket, hogy $L(n) \geq c2^{n/2}$ teljesüljön $n = 0, 1$ esetén. Mivel $L(0) = L(1) \geq 1$, így például a $c = 1/2$ is megfelelő. Tegyük fel, hogy $L(i) \geq c2^{i/2}$ teljesül minden $0 \leq i \leq n$ esetén. Belátjuk, hogy ekkor teljesül $n+1$-re is.

$$L(n+1) \geq L(n) + L(n-1) \geq c2^{n/2} + c2^{(n-1)/2} \geq c2^{(n-1)/2}(2^{1/2} + 1) \geq c2^{(n-1)/2} \cdot 2 = c2^{(n+1)/2}.$$

    Tehát például a $c = 1/2$ és $n_0 = 0$ választás mutatja, hogy $L(n) = \Omega(2^{n/2})$.