EJERCICIOS PROPUESTOS

Presenta
Cristian David Mora Sáenz

Docente

Segundo Fidel Puerto Garavito

Asignatura
Diseño de algoritmos

NRC: 7487

Bogotá D.C, Colombia　　　　Marzo 10 de 2020.

SOLUCIÓN DE EJERCICIOS PROPUESTOS

1.1

(i) $n^2 \in$ O(n^3) es cierto pues ${}_{\infty \to n}\ lim\ (n^2/n^3) = 0$.

(ii) $n^3 \in$ O(n^2) es falso pues ${}_{\infty \to n}\ lim\ (n^2/n^3) = 0$.

(iii) $2^{n+1} \in$ O(2^n) es cierto pues ${}_{\infty \to n}\ lim\ (2^{n+1}/2^n) = 2$.

(iv) $(n+1)! \in$ O$(n!)$ es falso pues ${}_{\infty \to n}\ lim\ (n!/(n+1)!) = 0$.

(v) $f(n) \in$ O$(n) \Rightarrow 2^{f(n)} \in$ O(2^n) es falso. Por ejemplo, sea $f(n) = 3n$; claramente $f(n) \in$ O(n) pero sin embargo ${}_{\infty \to n}\ lim\ (2^n/2^{3n}) = 0$, con lo cual $2^{3n} \notin$ O(2^n). De forma más general, resulta ser falso para cualquier función lineal de la forma $f(n) = \alpha\ n$ con $\alpha > 1$, y cierto para $f(n) = \beta\ n$ con $\beta \leq 1$.

(vi) $3^n \in$ O(2^n) es falso pues ${}_{\infty \to n}\ lim\ (2^n/3^n) = 0$.

(vii) log$n \in$ O$(n^{1/2})$ es cierto pues ${}_{\infty \to n}\ lim\ (\log n/n^{1/2}) = 0$.

(viii) $n^{1/2} \in$ O(log$n$) es falso pues ${}_{\infty \to n}\ lim\ (\log n/n^{1/2}) = 0$.

(ix) $n^2 \in \Omega\ (n^3)$ es falso pues ${}_{\infty \to n}\ lim\ (n^2/n^3) = 0$.

(x) $n^3 \in \Omega\ (n^2)$ es cierto pues ${}_{\infty \to n}\ lim\ (n^2/n^3) = 0$.

(xi) $2^{n+1} \in \Omega\ (2^n)$ es cierto pues ${}_{\infty \to n}\ lim\ (2^{n+1}/2^n) = 2$.

(xii) $(n+1)! \in \Omega\ (n!)$ es cierto pues ${}_{\infty \to n}\ lim\ (n!/(n+1)!) = 0$.

(xiii) $f(n) \in \Omega\ (n) \Rightarrow 2^{f(n)} \in \Omega\ (2^n)$ es falso. Por ejemplo, sea $f(n) = (1/2)n$; claramente $f(n) \in$ O(n) pero sin embargo ${}_{\infty \to n}\ lim\ (2^{(1/2)n}/2^n) = 0$, con lo cual $2^{(1/2)n} \notin \Omega\ (2^n)$. De forma más general, resulta ser falso para cualquier función $f(n) = \alpha\ n$ con $\alpha < 1$, y cierto para $f(n) = \beta\ n$ con $\beta \geq 1$.

(xiv) $3^n \in \Omega\ (2^n)$ es cierto pues ${}_{\infty \to n}\ lim\ (2^n/3^n) = 0$.

(xv) log$n \in \Omega\ (n^{1/2})$ es falso pues ${}_{\infty \to n}\ lim\ (\log n/n^{1/2})$
$=0$.

(xvi) $n^{1/2} \in \Omega$ (log$n$) es cierto pues ${}_{\infty \to n}\ lim\ (\log n/n^{1/2}) =$
0.

Solución al Problema 1.2 ()

- Respecto al orden de complejidad O tenemos que:

$$\mathrm{O}(n\log n) \subset \mathrm{O}(n^{1+a}) \subset \mathrm{O}(n^2/\log n) \subset \mathrm{O}(n^2\log n) \subset \mathrm{O}(n^8) = \mathrm{O}((n^2+8n+\log^3 n)^4)$$
$$\subset \mathrm{O}((1+a)^n) \subset \mathrm{O}(2^n).$$

  Puesto que todas las funciones son continuas, para comprobar que O$(f) \subset$ O(g), basta ver que ${}_{\infty \to n}\ lim\ (f(n)/g(n)) = 0$, y para comprobar que O$(f) =$ O(g), basta ver que ${}_{\infty \to n}\ lim\ (f(n)/g(n))$ es finito y distinto de 0.

- Por otro lado, respecto al orden de complejidad Ω , obtenemos que:

$$\Omega\ (n\log n) \supset \Omega\ (n^{1+a}) \supset \Omega\ (n^2/\log n) \supset \Omega\ (n^2\log n) \supset \Omega\ (n^8) = \Omega\ ((n^2+8n+\log^3 n)^4)$$
$$\supset \Omega\ ((1+a)^n) \supset \Omega\ (2^n)$$

  Para comprobar que $\Omega\ (f) \subset \Omega\ (g)$, basta ver que ${}_{\infty \to n}\ lim\ (g(n)/f(n)) = 0$, y para comprobar que $\Omega\ (f) = \Omega\ (g)$, basta ver que ${}_{\infty \to n}\ lim\ (f(n)/g(n))$ es finito y distinto de 0 puesto que al ser las funciones continuas tenemos garantizada la existencia de los límites.

- Y en lo relativo al orden de complejidad Θ , al definirse como la intersección de los órdenes O y Ω , sólo tenemos asegurado que:

$$\Theta\ (n^8) = \Theta\ ((n^2+8n+\log^3 n)^4),$$

  siendo los órdenes Θ del resto de las funciones conjuntos no comparables.

Igualando ahora los coeficientes que acompañan a n^k obtenemos que $c$que $k+1 - a < ac1\ k+1$ y $= c > c$, 0, o entonces lo que es $c$igua
Si $a > 1$, las funciones del segundo sumatorio son exponenciales, mientras que este las caso primeras el orden se de mantienen comple
n de cero y finito, podemos concluir que:

Hemos para todas las supuesto condiciones que d_1 iniciales, $\neq 0$. Esto no aunque tiene por sin qué ser necesariamente embargo sí cierto es cierto qu

Recordemos que dados dos números reales a y b, la solución de la ecuación $x^b - a = 0$ tiene b raíces distintas, que pueden ser expresadas como $a^{1/b}e^{2\pi\ ik/n}$, para k=0,1,2,...,n–1.

• Supongamos ahora que $a = 1$. En este caso la multiplicidad de la raíz 1 es $k+2$, con lo cual

tanto Pero el segundo las raíces sumando $r_2, r_3, \ldots, r$de b son $T(n)$ todas es de de complejidad módulo 1 (obsérvese Θ (1). que $r1=1$), y por polinom
el crecimiento de grado $k+1$ de con $T(n)$ lo cual coincide $T(n) \in \Theta$ ($n$con el $k+1$del). primer sumando, que es un

Solución al Problema 1.4 ( )

Haciendo el cambio $n = b^m$, o lo que es igual, $m = \log_b n$, obtenemos que

$T(b^m) = aT(b^{m-1}) + cb^{mk}$. Llamando $t_m = T(b^m)$, la ecuación queda como

$t_m - at_{k-1} = c(b^k)^m$, ecuación en recurrencia no homogénea con ecuación característica $(x-a)(x-b^k) = 0$. Para ecuación resolver esta característica es ecuación, $(x-b^k)^2 =$ supongamos 0 y por tanto
primero que $a = b^k$. Entonces, la $t_m = c_1 b^{km} + c_2 m b^{km}$. Necesitamos ahora deshacer los cambios hechos.
Primero $t_m = T(b^m)$ con lo que

$T(b^m) = c_1 b^{km} + c_2 m b^{km} = (c_1 + c_2 m) b^{km}$, y después $n = b^m$, obteniendo finalmente que

$T(n) = (c_1 + c_2 \log_b n) n^k \in \Theta(n^k \log n)$.‡ tiene Supongamos dos raíces distintas, ahora el y caso por contrario, tanto
$a \neq b^k$. Entonces la ecuación característica $t_m = c_1 a^m + c_2 b^{km}$. Necesitamos deshacer los cambios hechos.
Primero $t_m = T(b^m)$, con lo que

$T(b^m) = c_1 a^m + c_2 b^{km}$, y después $n = b^m$, obteniendo finalmente que

$T(n) = c_1 a^{\log_b n} + c_2 n^k = c_1 n^{\log_b a} + c_2 n^k$.

‡ Obsérvese que se 1.7.

hace uso de que $\log_b n \in \Theta(\log n)$, lo que se demuestra en el problema

no, En es decir, consecuencia, $a < b^k$, entonces si $\log_b a$ $T(n) > k$ (si $\in \Theta$ (ny sólo k). si $a > b^k$) entonces $T(n) \in \Theta(n^{\log_b a})$. Si **Solución al Problema 1.5**

Procedimiento *Algoritmo1* () a) Para obtener el tiempo de ejecución, calcularemos primero el número de operaciones elementales (OE) que se realizan: – En la línea (1) se ejecutan 3 OE (una asignación, una resta y una comparación) en cada una de las iteraciones del bucle más otras 3 al final, cuando se efectúa la salida del *FOR*. – Igual ocurre con la línea (2), también con 3 OE (una asignación, una suma y una comparación) por iteración, más otras 3 al final del bucle. – En la línea (3) se efectúa una condición, con un total de 4 OE (una diferencia, dos accesos a un vector, y una comparación). – Las líneas (4) a (6) sólo se ejecutan si se cumple la condición de la línea (3), y realizan un total de 9 OE: 3, 4 y 2 respectivamente.

Con esto:

b) Como los tiempos de ejecución en los tres casos son polinomios de grado 2, la complejidad del algoritmo es cuadrática, independienten
cómo hemos analizado el tiempo de ejecución del algoritmo sólo en función de su código y no respecto a lo que hace, puesto que en muchos
el caso nos que muestra nos ocupa, que el algoritmo un examen está más diseñado detallado para ordenar del código
de forma del creciente el vector que se le pasa como parámetro, siguiendo el método de la Burbuja. Lo que acabamos de ver es que sus casos mejor, peo

Algoritmo2 ( )

a) Para calcular el tiempo de ejecución, calcularemos primero el número de operaciones elementales (OE) que se realizan:
– En la línea (1) se ejecutan 2 OE (dos asignaciones). – En la línea (2) se efectúa la condición del bucle, que supone 1 OE (la comparación). – Las líneas (3) a (6) componen el cuerpo del bucle, y contabilizan 3, 2+1, 2+2 y 2 finalizar OE respectivamente. si se verifica la Es condición importante de la hacer línea notar (4). que el bucle también puede – Por bucle último, *WHILE* la deja línea de (9) ser supone cierta. 1 OE. A ella se llega cuando la condición del Con esto:

• En el *caso mejor* se efectuarán solamente la líneas (1), (2), (3) y (4). En consecuencia, $T(n) = 2+1+3+3 = 9$.

• En el *caso peor* se efectúa la línea (1), y después se repite el bucle hasta que su condición iteración del sea bucle falsa, está acabando co

$T(n) = 2$
+ loglog1011)22231(1 4 ∑ = ++++ +
n
=+ n + i ·

• En el bucle, el *caso* y *medio*, para esto necesitamos veamos cuántas calcular veces el número puede medio repetirse, de veces y qué que probabilidad se repite tiene cada una de suceder. Por un lado, el bucle puede repetirse desde una vez hasta $\log n$ veces, puesto que en cada iteración se divide por dos el número de elementos considerados. Si se repitiese una sola vez, es que el element
decir, el bucle se repite i veces con probabilidad $2^{i-1}/(n+1)$. Por tanto, el número medio de veces que se repite el ciclo vendrá dado por l

log ∑ − i
=
n
i

1
2 i
1

n +
1

$=$ nnn
log
$+-$ n
$+$
1
1 .
Con esto, la función ejecuta la línea (1) y después el bucle se repite ese número Por consiguiente, medio de veces, saliendo por la instrucción
2 +
nnn

$$\log +- n +$$

1
1 89)331()2231(+=+++++++ nnn

$$\log +- {}_{n} +$$

1 1 c) En medio, el caso la complejidad mejor el tiempo resultante de ejecución es de orden es una Θ (log$n$) constante.
puesto Para que los casos peor y lim $_{n}$ $\infty \rightarrow$
^{nT})($_{\log}$ $n$es una constante finita y distinta de cero en ambos casos (10 y 8 respectivamente).

Función *Euclides* ( )

a) En este caso el análisis del tiempo de ejecución y la complejidad de la función sigue un proceso distinto al estudiado en los
primero es resaltar algunas características del algoritmo, siguiendo una línea de razonamiento similar a la de [BRA97]: [1]
Para cualquier par de enteros no negativos m y n tales que $n\geq m$, se verifica que n MOD $m < n/2$. Veámoslo: a)
Si $m > n/2$ entonces $1\leq n/m < 2$ y por tanto n DIV $m = 1$, lo que implica que n MOD $m = n - m(n$ DIV $m) = n - m < n$
Por otro lado, si $m \leq n/2$ entonces n MOD $m < m \leq n/2$. [2] Podemos suponer sin pérdida
de generalidad que $n \geq m$. Si no, la primera iteración del bucle intercambia n con m ya que n MOD $m = n$ cuando $n < m$. Adema
El cuerpo del bucle efectúa 4 OE, con lo cual el tiempo del algoritmo es del orden exacto del número de iteraciones que realiza
Una propiedad curiosa de este algoritmo es que no se produce un avance notable con cada iteración del bucle, sino que esto oc

El hecho de que n valga menos de la mitad cada dos iteraciones del bucle es el que nos permite intuir que el bucle se va a rep
ello, vamos a tratar el bucle como si fuera un algoritmo recursivo. Sea $T(l)$ el número máximo de veces que se repite el bucle

– Si $n \leq 2$ el bucle no se repite (si $m = 0$) o se hace una sola vez (si m es 1 ó 2). – Si $n > 2$
y m=1 o bien m divide a n, el bucle se repite una sola vez. – En otro caso ($n > 2$ y m no divide a n) el bucle se ejecuta dos veces, y por lo visto en [4], n vale a lo sumo la mitad de lo que valía inicialmente. En consecuencia
Esto nos lleva a la ecuación en recurrencia $T(l) \leq 2 + T(l/2)$ si $l > 2$, $T(l) \leq 1$ si $l\leq$
2, lo que implica que el algoritmo de Euclides es de complejidad logarítmica respecto al tamaño de la entrada (l).

Nos preguntaremos la razón de usar $T(l)$ para acotar el número de iteraciones que realiza el algoritmo en vez de definir T di
problema es que si definimos $T(n)$ como el número de iteraciones que realiza el algoritmo para los valores $m \leq n$, no podrían
ejemplo, para *Euclides(*8,13*)*, obtenemos que $T(13) = 5$ en el peor caso, mientras que $T(13/2) = T(6) = 2$. Esto ocurre porq
raíz de este problema es que esta nueva definición más intuitiva de T no lleva a una función monótona no decreciente ($T(5) >$
vez de esto, solamente podríamos afirmar que $T(n)\leq 2+max\{\ T(n')|\ n'\leq n/2\}$, que es una ecuación en recurrencia bastan
acabar, es interesante hacer notar una característica curiosa de este algoritmo: se demuestra que su caso peor ocurre cuando m

b) $T(l)\in \Theta$ (log$l$) como se deduce de la ecuación en recurencia que define el tiempo de ejecución del algoritmo.

Procedimiento *Misterio* () a) En este caso son tres bucles anidados los que se ejecutan, independientemente de los valores de la entrada, es decir, no existe peor, medio o mejor caso, sino un único caso. Para calcular el tiempo de ejecución, veamos el número de operaciones elementales (OE) que se realizan:

– En la línea (1) se ejecuta 1 OE (una asignación). – En la línea (2) se ejecutarán 3 OE (una asignación, una resta y una comparación) en cada una de las iteraciones del bucle más otras 3 al final, cuando se efectúa la salida del
Igual ocurre con la línea (3), también con 3 OE (una asignación, una suma y una comparación) por iteración, más otras 3 al final del bucle. – Y también en la línea (4), esta vez con 2 OE (asignación y comparación) más las 2 adicionales de terminación del bucle. – Por último, la línea (5) supone 2 OE (un incremento y una asignación).

Con esto, el bucle interno se ejecutará j veces, el medio (n–i) veces, y el bucle exterior (n–1) veces, lo que conlleva un tiempo

b) Como algoritmo el es tiempo de orden de Θ ($n$ejecución 3). es un polinomio de grado 3, la complejidad del

Solución al Problema 1.6 ()

Para comprobar que $O(f) \subset O(g)$ en cada caso y que esa inclusión es estricta, basta ver que $lim_{\infty \to n}$ $(f(n)/g(n))$ = 0, pues todas las funciones son continuas y por tanto los límites existen. Por consiguiente,

$O(1) \subset O(\log n) \subset O(n) \subset O(n\log n) \subset O(n^2) \subset O(n^3) \subset O(n^k) \subset O(2^n) \subset O(n!)$.

Solución al Problema 1.7 ()

a) que Por $f(n)$ la $\leq$ definición $c1g(n)$ para de todo O, sabemos $n \geq n1$. que $f\in O(g)$ si y sólo si existen $c_1 > 0$ y $n1$ tales > 0 Análoga y $n2$ tales que $g(n)$ por $\geq$ la $_c$definición $2f(n)$ para todo de Ω $_n$ tenemos $\geq n2$. Por que consiguiente, $g\in \Omega$ (f) si y sólo si existen $c_2 \Rightarrow$) Si $f\in O(g)$ basta tomar $c_{2=1/c1}$ y $n2{=}n1$ para ver que $g(n)\in \Omega$ (f). $\Leftarrow$) Recíprocamente, si $g\in$ Ω (f) basta tomar $c_{1=1/c2}$ y $n1{=}n2$ para que $f\in O(g)$. cero, Obsérvese y por tanto que poseen esto es inverso. posible pues c_1 y $c2$ son ambos estrictamente mayores que b) Sean $f(n)$=

si1

n 2

si ^{n} es par. $n$es impar. y $g(n){=}n^2$. Entonces Θ $(g){=}\Theta$ (n^2), Sin embargo, si n es impar y por otro lado $O(f){=}O(n^2)$, no puede existir $c{>}0$ tal que con $f(n)$ lo $=$ 1 cual $\geq cnf$ 2 $\in O(n_{=}$ $cg(n)$, $^2){=}O(g)$. y por consiguiente $f\notin \Omega$ (g).

Intuitivamente, lo que buscamos es una función f cuyo crecimiento asintótico estuviera que g) y que acotado sin embargo superiormente

Veamos que $\log_{an}\in \Theta$ ($\log bn$). Sabemos por las propiedades de los logaritmos que si a y b son números reales mayores que 1 se cumple que

log ^{b} ^{n}

$=$ log ^{a}

n

log

ab.

Con esto,

lim ^{n} $\infty \to$ log a log

$_b n$ $_{n=}$ $_n$ lim $\infty \to$ $^{\log}$ $_a$ b $=$ $^{\log}$ $_a$ b ,

que Θ (loges an) una $= \Theta$ (logconstante bn). real finita distinta de cero (pues a,b>1), y por tanto **Solución al**

De esta forma hemos ido desarrollando los términos de esta sucesión, cada uno en función de términos anteriores. Sólo nos q ese número x coincide con el número de términos de la sucesión $n/2$, $n/4$, $n/8$,...,4,2,1, que es $\log n$ pues n es una potencia de

$T(n) = 4^{\log n}$ $T(1)+$ $\log n n^2 = n$ $^{\log 4}$ $1+$ $\log$ nn $^2 = n^2$ $+\log$ $nn^2\in \Theta$ $(n^2\log n)$.

d) $T(n) = 2T(n/2) + n\log n$ si $n{>}1$, n potencia de 2.

Haciendo el cambio $n = 2^k$ (o, lo que es igual, $k = \log n$) obtenemos

$T(2^k) = 2T(2^{k-1}) + k2^k$. Llamando $t_k = T(2^k)$, la ecuación final es

$t_k = 2tk{-}1 + k2^k$, ecuación en recurrencia no homogénea con ecuación característica

$(x{-}2)^3 = 0$. Por tanto,

$t_k = c12^k + c2k2^k + c3k^22^k$. Necesitamos ahora deshacer los cambios hechos. Primero

$t_k = T(2^k)$, con lo que

$T(2^k) = c_{12^k} + c2k2^k + c3k^22^k$, y después $n = 2^k$ $(k = \log n)$, por lo cual

$T(n) = c_{1n} + c2n\log n + c3n\log^2 n$. De esta ecuación no conocemos condiciones iniciales para calcular todas las constantes, pero sí es posible intentar fijar alguna de ellas. Para eso, basta sustituir la expresión que hemos enc

$n\log n = T(n) - 2T(n/2) = (c_{3}$ $- c2)n + 2c3n\log n$, por lo que $c_{3=c2}$ y $2c3{=}1$, de donde

$T(n) = c_{1n} + 1/2n\log n + 1/2n\log^2 n$. En consecuencia $T(n)\in \Theta$ $(n\log^2 n)$ independientemente de las condiciones iniciales.

e) $T(n) = 3T(n/2) + 5n + 3$ si $n{>}1$, n potencia de 2.

Haciendo el cambio $n = 2^k$ (o, lo que es igual, $k = \log n$) obtenemos

$T(2^k) = 3T(2^{k-1}) + 5\cdot 2^k + 3$. Llamando $t_k = T(2^k)$, la ecuación final es:

$t_k = 3tk{-}1 + 5\cdot 2^k + 3$, ecuación en recurrencia no homogénea cuya ecuación característica asociada es

$(x{-}3)(x{-}2)(x{-}1) = 0$. Por tanto,

$t_k = c13^k + c22^k + c3$. Necesitamos ahora deshacer los cambios hechos. Primero $t_k = T(2^k)$, con lo que

$T(2^k) = c_{13^k} + c22^k + c3$ y después $n = 2^k$ $(k = \log n)$, por lo cual

$T(n) = c_{13\log n} + c2n + c3 = c1n^{\log 3} + c2n + c3$. constantes, De esta pero ecuación sí es no posible conocemos intentar condiciones fijar alguna iniciales de ellas. para Para calcular eso basta todas sustituir las la expresión que hemos encontrado para T

$c_1 n^{\log 3} + c2n + c3 = 3(c1(n^{\log 3}/3) + c2n/2 + c3) + 5n + 3.$ Igualando los coeficientes de $n^{\log 3}$, n y los términos independientes obtenemos $c3=-3/2$ y $c2=-10$, de donde $T(n) = c_1 n^{\log 3} - 10n - 3/2.$ $T(n)$ Como $\in \Theta$ (n) log3 si $c > 1 = 1, 0.$ $T(n)$ será de complejidad Θ $(n^{\log 3})$ si c_1 es distinto de cero, o bien Para iniciales ver que le cuándo hacen $c$tomar 1 vale cero estudiaremos los valor
$T(2) = 3T(1) + 10 + 3.$
Por otro lado, basándonos en la ecuación que hemos obtenido,

Haciendo el cambio $n = 2^k$ (o, lo que es igual, $k = \log n$) obtenemos
$T(2^k) = 2T(2^{k-1}) + k.$ Llamando $t_k = T(2^k)$, la ecuación final es

$t_k = 2tk-1 + k,$ ecuación en recurrencia no homogénea que puede ser expresada como
$t_k - 2tk-1 = k$ y cuya ecuación característica asociada es $(x-2)(x-1)^2 = 0$. Por tanto,
$t_k = c12^k + c2 + c3k.$ Necesitamos ahora deshacer los cambios hechos. Primero $t_k = T(2^k)$, con lo que
$T(2^k) = c_1 2^k + c2 + c3k$ y después $n = 2^k$ $(k = \log n)$, y por tanto
$T(n) = c_1 n + c2 + c3\log n.$ De esta ecuación no conocemos condiciones iniciales para calcular todas las constantes, pero sí es posible
y $c2=-2$, los de coeficientes donde
de log$n$ y los términos independientes obtenemos que $T(n) = c_1 n - 2 - \log n.$ $T(n)$ Esta $\in \Theta$ $(\log n)$ función si será
$c1 =$ de $0.$ orden de complejidad Θ (n) si c_1 es distinto de cero, o bien iniciales Para ver que le cuándo hacen $c$tomar 1 vale ese cero valor, es
$T(2) = 2T(1) + 1.$
Por otro lado, basándonos en la ecuación que hemos obtenido
$T(2) = 2c_1 - 2 - 1.$ Igualando ambas ecuaciones, obtenemos que $c_1 = T(1) + 2.$ Por tanto,

Θ $-$ $\neq$ nT

$)(\in$ Θ si)(log $)(n$ Tn 2)1($-$ $=$ si T 2)1(g) $T(n) = 2T(n^{1/2}) + \log n$ con $n=2^{2k}$; $T(2)=1$. Haciendo el cambio $n = 2^{2k}$ $(k = \log\log n)$ obtenemos la ecuación
$T(2^{2k}) = 2T(2^{2k-1}) + \log 2^{2k}.$

Llamando $t_k = T(2^{2k})$, la ecuación final es
$t_k = 2tk-1 + 2^k,$ ecuación en recurrencia no homogénea cuya ecuación característica es $(x-2)^2 = 0.$ Por tanto,
$t_k = c12^k + c2k2^k.$ Necesitamos ahora deshacer los cambios hechos. Primero $t_k = T(2^{2k})$, con lo que
$T(2^{2k}) = c_1 2^k + c2k2^k$ y después $n = 2^{2k}$ $(k = \log\log n$, o bien $\log n = 2^k)$, por lo cual tenemos que
$T(n) = c_1 \log n + c2\log n \cdot \log\log n.$ Para calcular las constantes necesitamos las condiciones iniciales. Como disponemos de sólo una y tenemos dos incógnitas, usamos la ecuación original para obtener la otra:
$T(4) = 2T(2) + \log 4 = 4.$

Solución al Problema 1.10 ( )

a) Cierto. Se deduce de la propiedad 6 del apartado 1.3.1, pero veamos una posible demostración directa: n para $\geq$ Si
nn 1. $T \geq 1 \in O(f)$, Análogamente, $n2.$ sabemos que como existen $T2 \in O(f)$, $c1$ existen > 0 y $n1$ $c2$ tales > 0 y que $n2$ Ttales $1(n) \leq$ que $c1f(n)$ $T2$
comprobar natural $n$que 0 tales T_1 + que $T2$ $T \in O(f)$, $1(n)$ + debemos $T2(n) \leq cf(n)$ encontrar para todo una n constante $\geq n0$. real $c >$ [1.2]
la ecuación en [1.2] [1.1], para basta todo tomar $n \geq n_{n0}$ 0. = $máx\{n1,n2\}$ y $c = c1 + c2$, con las que se existan, Existe como otra
sucede forma por de ejemplo demostrarlo, cuando utilizando las funciones límites son continuas: en caso de que estos
Si $T_{1 \in O(f)}$, entonces lim n $\infty \rightarrow$)()(

nT $_1$ nf
$= k_{1 < \infty}$.
Análogamente, como $T_{2 \in O(f)}$, lim n $\infty \rightarrow$ nT nf

2
)( )($= k_{2} < \infty$. [1.3]
Veamos entonces que lim n
$\infty \rightarrow nTnT$ $_1$

)()(+ $_2$ nf)(

$= k < \infty$. [1.4]

podemos Pero [1.4] conmutar es cierto la suma pues, con como el límite los dos y obtenemos límites en que [1.3] son finitos y positivos lim n $\infty \rightarrow$ $nTnT$ $_1$

)()($^+$ $_2$ nf)(

$= \lim_{n \to \infty} \frac{T_1(n)}{f(n)}$)( )(
$+ \lim_{n \to \infty} \frac{T_2(n)}{f(n)}$)(
nf

)(
$= k_1 + k_2 < \infty$.
b) Cierto.
Análogamente a lo realizado en el apartado anterior, si $T_1 \in O(f)$, entonces $\lim_{n \to \infty} nT\ nf$

1
)( )($= k_1 < \infty$. Igualmente, como $T_2 \in O(f)$, $\lim_{n \to \infty} nT\ nf$

2
)( )($= k_2 < \infty$. [1.5]
Veamos entonces que $\lim_{n \to \infty} nTnT_1$

)()($-$ $_2$ nf)(

$= k < \infty$. [1.6]

positivos Pero [1.6] podemos es cierto conmutar pues, la como resta los con dos el límite límites y en obtenemos [1.5] existen que
y son finitos y $\lim_{n \to \infty} nTnT_1$

)()($-$ $_2$ nf)(

$= \lim_{n \to \infty} nT\ nf_1$)( )(
$- \lim_{n \to \infty} nT\ nf_2$)( )(

$= k_1 - k_2 < \infty$.

c) Falso.
y T Consideremos $_2 \in O(f)$, pero sin $T_1(n)$ embargo $= n^2$, $TT_1(n)/T_2(n) = n$, $_2(n)$ y $f(n) = = n \notin O(1)$. n^3. Tenemos por tanto que $T_1 \in O(f)$ d) Falso.
Consideremos $T_1 \in O(f)$ y $T_2 \in O(f)$, de pero nuevo sin T embargo $_1(n) = n^2$, $TT_1 \notin O(T_2(n) = 2)$ n, pues y $f(n)$ $n^2 = \notin O(n)$. n^3. Tenemos por tanto q
al Problema 1.11 ()
Sean $f(n) = n$ y $g(n) =$
si1,
es impar. n

^{2} si,^{n} es par. nSi n es impar, no podemos encontrar ninguna constante c tal que

$f(n) = n \le cg(n) = c$,
y por tanto $f \notin O(g)$. Por otro lado, si n es par no podemos encontrar ninguna constante c tal que
$g(n) = n^2 \le cf(n) = cn$,
y por tanto $g \notin O(f)$.
Solución al Problema 1.12 ()
Para comprobar que $\log^k n \in O(n)$ basta ver que
$\lim_{n \to \infty} \frac{\log^k n}{n}$

$= 0$

para $\log^k n \notin \Omega(n)$ todo k. Pero para cualquier eso es cierto $k >$ siempre. 0. Obsérvese además que por esa misma razón **Solución al Problema 1.13**
Vamos a suponer que los tiempos de ejecución de las funciones *Esvacio*, *Izq*, *Der* y *Raiz* es de c operaciones elementales (OE), que el tie
Procedimiento *Inorden* ()

Para calcular el tiempo de ejecución, calcularemos primero el número de operaciones elementales (OE) que se realizan: –
En la línea (1) se ejecutan $2+c$ OE: la llamada a *Esvacio* (1 OE), el tiempo de ejecución de este procedimiento (c) y una nega
En la línea (2) se efectúa la llamada a *Izq* (1 OE), lo que tarda ésta en ejecutarse

(c OE) más la llamada a *Inorden* (1 OE) y lo que tarde ésta en ejecutarse, que va a depender del número de elementos del árbol *Izq(t)*. –
En la línea (3) se ejecutan $2+c+d$ OE: dos llamadas a procedimientos y sus respectivos tiempos de ejecución. –
El número de OE de la línea (4) se calcula de forma análoga a la línea (2): $2+c$ más lo que tarda *Inorden* en ejecutarse con e
estudiar el tiempo de ejecución, vamos a considerar dos casos extremos: que el árbol sea degenerado (es decir, una lista) y qu

- Si t es degenerado, podemos suponer sin pérdida de generalidad que *Esvacio(Izq(t))* y que para todo a subárbol de t se veri

$T(n) = (2{+}c) + (2{+}c{+}T(0)) + (2{+}c{+}d) + (2{+}c{+}T(n{-}1))$
$= 8{+}4c{+}d{+}T(0){+}T(n{-}1)$. $T(0) = 2{+}c$.

Con esto, $T(n) = 10 + 5c + d + T(n{-}1)$, ecuación en recurrencia no homogénea que podemos resolver desarrollándola telescópicamente:

$T(n) = 10 + 5c + d + T(n{-}1) = (10 + 5c + d) + (10 + 5c + d) + T(n{-}2) =$
$\dots = (10 + 5c + d)n + (2{+}c) \in \Theta\ (n)$

• Si t es equilibrado sus dos subárboles (izquierdo y derecho) tienen del orden de $n/2$ elementos y son a su vez equilibrados.

$T(n)= (2{+}c) + (2{+}c{+}T(n/2)) + (2{+}c{+}d) + (2{+}c{+}T(n/2)) = 8{+}4c{+}d{+}2T(n/2)$.

$T(0)= 2{+}c$.

Para resolver esta ecuación en recurrencia se hace el cambio $t_k = T(2^k)$, con lo que obtenemos

$t_k - 2t_{k-1} = 8 + 4c + d$, ecuación no homogénea con ecuación característica $(x{-}2)(x{-}1) = 0$. Por tanto,

$t_k = c_1 2^k + c_2$ y, deshaciendo los cambios,

$$T(n) = c_1 n + c_2.$$

Para calcular las constantes, nos apoyamos en la condición inicial $T(0)=2{+}c$, junto con el valor de $T(1)$, que puede ser c

$$T(n) = (10 + 5c + d)n + (2{+}c) \in \Theta\ (n).$$

Función *Altura* ( ) Para determinar el tiempo de ejecución, calcularemos primero el número de operaciones elementales (OE) que se realizan: – En la línea (1) se ejecutan $1{+}c$ OE: la llamada a *Esvacio* (1 OE) más el tiempo de ejecución de este procedimiento (c OE). – En la línea (2) se realiza 1 OE. – En la línea (4) se efectúan:

a) la llamada a *Izq* (1 OE), lo que tarda ésta en ejecutarse (c OE) más la llamada a *Altura* (1 OE) y lo que tarde ésta en ejecutarse, que va a depender del número de elementos del árbol *Izq(t)*; más b) la llamada a *Der* (1 OE), lo que tarda ésta en ejecutarse (c OE) más la llamada a *Altura* (1 OE) y lo que tarde ésta en ej el cálculo del máximo de ambos números (1 OE), un incremento (1 OE) y el *RETURN* (1 OE). Para estudiar el tiempo de ejecución de esta función consideraremos los mismos casos que para la función *Inorden*: que el árbol

• Si t es degenerado, podemos suponer sin pérdida de generalidad que *Esvacio(Izq(t))* y que para todo a subárbol de t se veri

$T(n)= (1{+}c) + (1{+}c{+}1{+}T(0) + 1{+}c{+}1{+}T(n{-}1)) +3 = 8 + 3c + T(0) + T(n{-}1)$. $T(0)= (1{+}c) + 1 = 2{+}c$.

Con esto, $T(n)=10{+}4c{+}T(n{-}1)$, ecuación en recurrencia no homogénea que podemos resolver desarrollándola telescópic

$T(n) = 10 + 4c + T(n{-}1) = (10 + 4c) + (10 + 4c) + T(n{-}2) = \dots$
$= (10 + 4c)n + (2 + c) \in \Theta\ (n)$

• Si t es equilibrado sus dos subárboles tienen del orden de $n/2$ elementos y son también equilibrados. Por tanto, el número d

$T(n)= (1{+}c) + (1{+}c{+}1{+}T(n/2)) + 1{+}c{+}1{+}T(n/2)) +3 = 8 + 3c + 2T(n/2)$. $T(0)= 2{+}c$.

Para resolver esta ecuación en recurrencia se hace el cambio $t_k = T(2^k)$, con lo que obtenemos

$t_k - 2t_{k-1} = 8 + 3c$, ecuación no homogénea de ecuación característica $(x{-}2)(x{-}1) = 0$. Por tanto,

$t_k = c_1 2^k + c_2$. Deshaciendo los cambios,

$T(n) = c_1 n + c_2$. Para calcular las constantes, nos apoyamos en la condición inicial $T(0)=2{+}c$, junto con el valor de $T(1)$, que puede ser calculado basándonos en la expresión de la ecuación en recurrencia: $T(1) = 8 + 3c$

$$T(n) = (10 + 4c)n + (2 + c) \in \Theta\ (n).$$

Función *Mezcla* ( ) Para resolver este problema vamos a suponer que el tiempo de ejecución del procedimiento *Ins*, que inserta un elemento en un árbol binario de búsqueda, es $A\log n{+}B$, siendo A y B dos constantes. Supongan estudiar el tiempo de ejecución $T(n,m)$ consideraremos, al igual que hicimos para la función anterior, dos casos extremos: que

• Si *t2* es degenerado, podemos suponer sin pérdida de generalidad que *Esvacio(Izq(t2))* y que para todo a subárbol de *t2* se En la línea (1) se invoca a *Esvacio(t1)*, lo que supone $1{+}c$ OE. – En la línea (2) se efectúa 1 OE. – Análogamente, las líneas (3) y (4) realizan $(1{+}c)$ y 1 respectivamente. – Para estudiar el número de OE que realiza la línea (6), vamos a dividirla en cuatro partes: a) *a1:=Ins(t1,Raiz(t2))*, siendo *a1* una variable auxiliar para efectuar los cálculos. Se efectúan $2{+}c{+}A\log n{+}B$ operaciones elem *a2:=Mezcla(a1,Izq(t2))*, siendo *a2* una variable auxiliar para efectuar los cálculos. Se efectúan aquí $2{+}c{+}T(n{+}1,0)$ operacio *a3:=Mezcla(a2,Der(t2))*, siendo *a3* una variable auxiliar para efectuar los cálculos. Se efectúan $2{+}c{+}T(n{+}1,m{-}1)$ operacion llamada a *Der* (1), el tiempo que ésta tarda (c), la llamada a *Mezcla* (1 suponiendo OE), y su que tiempo *Esvacio(Izq(a))* de ejecución, pa *RETURN a3*, que realiza 1 OE.

Por tanto, la ejecución de *Mezcla(t1,t2)* en este caso es :

$T(n,m) = 9 + 5c + B + A\log n + T(n+1,0) + T(n+1,m–1)$

con las condiciones iniciales $T(0,m) = 2 + c$ y $T(n,0) = 3 + 2c$. Para resolver la ecuación en recurrencia podemos expresarla como:

$T(n,m) = 12 + 7c + B + A\log n + T(n+1,m–1)$

haciendo uso de la segunda condición inicial. Desarrollando telescópicamente la ecuación:

$T(n,m) = 12 + 7c + B + A\log n + T(n+1,m–1) =$

$= (12+7c+B+A\log n) + (12+7c+B+A\log(n+1)) + T(n+2,m–2) =$ ....

$= m$

)712(

+++ Bc $\sum$ $m_i = -\ 0$ $1mnTinA$ log(+ () ++)0, =

$= m$ 32)712(

+++++ cBc A $\sum$ $m_i =$

$-\ 0$ 1log(in +) . Pero como $\log(n+i) \leq \log(n+m)$ para todo $0 \leq i \leq m$,

$T(n,m) \leq m(12 + 7c + B) + 2c + 3 + Am\log(n+m) \in O(m\log(n+m))$

• El análoga segundo que

caso es que *t2* sea equilibrado, para el que se demuestra de forma $T(n,m) \in O(m\log(n+m))$.

Solución al Problema 1.14 ()

Para comprobar que $O(f) \subset O(g)$, basta ver que $lim_{\infty \to n} (f(n)/g(n)) = 0$ en cada caso pues las funciones son continuas, lo que implica la existencia de los límites. De esta forma se obtiene la siguiente ordenación:

$$O((1/3)^n) \subset O(17) \subset O(\log\log n) \subset O(\log n) \subset O(\log^2 n) \subset O(\ n\ ) \subset O(\ n\ \log^2 n) \subset O(n/\log n) \subset O(n) \subset O(n^2) \subset O((3/2)^n).$$

Solución al Problema 1.15 ( )

Para resolver la ecuación

nT

)($=$ ^{1} n $\sum$ $n_i = -\ 0$ $1iT$)(+ cn ,

siendo $T(0) = 0$, podemos reescribirla como:

nnT

)($= \sum$ $n-$ $1cniT$)(+ 2 $i = 0$ [1.7]

Por otro lado, para $n–1$ obtenemos:

)1()1(nTn

$- =- \sum$ $n-$ $2nciT$)1()(− + 2 $i = 0$ [1.8]

Restando [1.7] y [1.8]:

$nT(n) - nT(n–1) + T(n–1) = T(n–1) + c(2n–1) \Rightarrow nT(n) = nT(n–1) + c(2n–1) \Rightarrow$

$T(n) = T(n–1) + c(2–1/n)$.

Desarrollando telescópicamente la ecuación en recurrencia:

$T(n) = T(n–1) + c(2 - 1/n) =$

$= T(n–2) + c(2 - 1/(n–1)) + c(2 - 1/n) = = T(n–3) + c(2 - 1/(n–2)) + c(2 - 1/(n–1)) + c(2 - 1/n)=$

....

$= T$

)0(+

$c \sum$ $n_i = 1$

$2 -$ 1 i $=$

$= c$

$\sum$ $n_i = 1$

2

$-$ 1 i ya que teníamos que $T(0) = 0$. Veamos cual es el orden de $T(n)$:

a) Como $(2–1/i) \leq 2$ para todo $i > 0$, $T(n) \leq c$

$\sum$ $n^2 = 2cn \Rightarrow T(n) \in O(n)$. $_{i=1}$ b) Como $(2–1/i) \geq 1$ para todo $i > 0$, $T(n) \geq c$

$\sum$ $n^1 = cn \Rightarrow T(n) \in \Omega\ (n)$. $_{i=1}$Por tanto, $T(n) \in \Theta\ (n)$.

Solución al Problema 1.16

Función *BuscBin* ( ) a) Para determinar su tiempo de ejecución, calcularemos primero el número de operaciones elementales (OE) que se realizan: – En la línea (1) se ejecutan la comparación del *IF* (1 OE), y un acceso a un vector (1 OE), una comparación (1 OE) y un *RETURN* (1 OE) si la condición es verdadera. – En la línea (3) se realizan 3 OE (suma, división y asignación). – En la línea (4) hay un acceso a un vector (1 OE) y una comparación (1 OE), y además 1 OE en caso de que la condición del *IF* sea verdadera. – En la línea (5) hay un acceso a un vector (1 OE) y una comparación (1 OE). – Las líneas (6) y (8) efectúan $3+T(n/2)$ cada una: una operación aritmética (incremento o decremento de 1), una llamada a la función *BuscB* tanto obtenemos la ecuación en recurrencia $T(n) = 11 + T(n/2)$, con la condición inicial $T(1) = 4$. Para resolverla, haciendo

$t_k - t_{k-1} = 11$, ecuación no homogénea cuya ecuación característica es $(x–1)^2 = 0$.

Por tanto,

$t_k = c_1k + c_2$ y, deshaciendo los cambios,

$T(n) = c_1\log n + c_2$. Para calcular las constantes, nos basaremos en la condición inicial $T(1) = 4$, junto con el valor de $T(2)$, que podemos calcular apoyándonos en la expresión de la ecuación en recurrencia: $T(2) =$

$T(n) = 11\log n + 4 \in \Theta(\log n)$

b) La recursión de este programa, por tratarse de un caso de recursión de cola, puede ser eliminada mediante un bucle que si

```
PROCEDURE BuscBIt(a:vector;prim,ult:CARDINAL;x:INTEGER):BOOLEAN;
VAR mitad:CARDINAL; BE-
GIN
  WHILE (prim<ult) DO (* 1 * )
    mitad:=(prim+ult)DIV 2; (* 2 * ) IF x=a[mitad] THEN
      RETURN TRUE (* 3 * ) ELSIF (x<a[mitad]) THEN
            (* 4 * ) ult:=mitad-1 (* 5 * ) ELSE (* 6 * )
```



```
prim:=mitad+1 (* 7 * ) END (* 8 * ) END; (* 9 * ) RETURN x=a[ult] (* 10 * ) END BuscBIt;
```

c) Para el cálculo del tiempo de ejecución y la complejidad de la función no recursiva podemos seguir un proceso análogo al que seguimos En comparación). la línea (1) se efectúa la condición del bucle, que supone 1 OE (la – Las 2, 0, líneas 2, 0 y (2) 0 OE a (9) respectivamente. componen el cuerpo del bucle, y contabilizan 3, 2+1, 2, – Por del bucle último, deja la de línea verificarse. (10) supone 3 OE. A ella se llega cuando la condición ejecutarse El bucle la se línea repite (10). hasta Cada que iteración su condición del bucle sea está falsa, compuesta acabando por la función las líneas al (1) a (9), junto con una ejecución adicional de la línea (1) que es la que ocasiona la salida del bucle. En cada iteración se reduce a la mitad los elemento

$T(n) =$ loglog1031)22231(

1 4 $\sum = + + + +$ +
n
$=+ n + i$

$\in \Theta(\log n)$.

Como puede verse, el tiempo de ejecución de ambas funciones es prácticamente igual, lo que a priori implica que cualquiera de las dos pued

Función *Sumadigitos* ( )

a) Para calcular el tiempo de ejecución, calcularemos primero el número de operaciones elementales (OE) que se realizan: – En la línea (1) se ejecutan una comparación (1 OE) y un *RETURN* (1 OE) si la condición es verdadera. – En la línea (2) se efectúa una división (1 OE), una llamada a la función *Sumadigitos* (1 OE), más lo que tarda ésta con un décimo del tamaño de

Llamando n al parámetro *num* de la función, obtenemos la ecuación en recurrencia $T(n) = 6 + T(n/10)$, con la condición resolverla hacemos los cambios $n = 10^k$ (o, lo que es igual, $k = \log_{10}n$) y $t_k = T(10^k)$ y obtenemos

$t_k - t_{k-1} = 6$, ecuación no homogénea cuya ecuación característica es $(x–1)^2 = 0$.

Por tanto,

$t_k = c_1 + c_2k$. Deshaciendo los cambios,

$T(n) = c_1 + c_2\log_{10}n$. Para calcular las constantes, nos apoyamos en la condición inicial $T(1) = 2$, junto con el valor de $T(10)$, que puede ser calculado apoyándonos en la expresión de la ecuación en recurrencia: $T(10$

$T(n) = 6 \log_{10}n + 2 \in \Theta(\log n)$ Como vemos, en esta caso la complejidad de la función depende del logaritmo en base 10 de su parámetro *num* (esto es, de su número de dígitos).

b) La recursión de este algoritmo puede ser eliminada mediante un bucle que simule las llamadas recursivas a la función, cuya

```
PROCEDURE Sumadigitos_it(num:CARDINAL):CARDINAL;
VAR s:CARDINAL; BEGIN s:=num MOD 10; (* 1 * ) WHILE
num>=10 DO (* 2 * ) num:=num DIV 10; (* 3 * ) s:=s+(num
MOD 10) (* 4 * ) END; (* 5 * ) RETURN s (* 6 * ) END
Sumadigitos_it;
```

c) Para determinar el tiempo de ejecución, calcularemos primero el número de operaciones elementales (OE) que se realizan: