ALGORITMIA

Tema 1

Soluciones a Problemas de Examen Cursos Anteriores

Grado en Ingeniería Informática en Tecnologías de la Información
Escuela Politécnica de Ingeniería – Campus de Gijón
Universidad de Oviedo

ÍNDICE

Prólogo

Este documento contiene los enunciados, y sus soluciones, de las preguntas relacionadas con el tema "Análisis de Algoritmos" formuladas en los exámenes de la asignatura desde el curso 2015-2016.

El contenido y la metodología de la asignatura ha variado con el tiempo. Por tanto, puede haber preguntas (o apartados de preguntas) que estén fuera del alcance de la materia/enfoque de este curso académico (cuanto mayor sea la distancia temporal con el curso actual mayor es la probabilidad de que pueda suceder).

Curso Académico 2021 – 2022

Sea la tabla adjunta donde n es la talla del problema, $T(n)$ los pasos del algoritmo, $C(n)$ el tiempo en segundos de una implementación en lenguaje C y $P(n)$ el tiempo en segundos de una implementación en lenguaje P. Para todo valor de n se mantiene la razón matemática mostrada en la tabla y se acepta la siguiente equivalencia: "*1 paso = 1 segundo*". Completar, en la zona de respuestas, las frases/expresión con precisión ("*La pregunta es correcta sí y solo sí todas las respuestas son correctas. En caso contrario se considerará totalmente incorrecta*").

n	$T(n)$	$C(n)$	$P(n)$
1	1	0.1	10
3	9	0.9	90
5	25	2.5	250
7	49	4.9	490
9	81	8.1	810
…	…	…	…

Respuesta

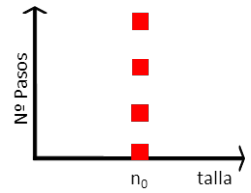
$T(n) \in \boldsymbol{\theta(n^2)}$

$C(n)$ es $\mathbf{0.1}T(n)$

$P(n)$ es $\mathbf{100}C(n)$

Por el **Principio de Invarianza** son "*iguales*".

En la figura adjunta se muestra el número de pasos contabilizados, en *distintas simulaciones*, de un algoritmo para un valor concreto (n_0) de la talla del problema. ¿qué está *diciendo* la figura? Razonar la respuesta.



Respuesta

El algoritmo presenta mejor y peor caso dado que, para un valor concreto de la talla, el número de pasos cambia (no es constante) en función de *algo* que no depende del tamaño del problema, de *algo* que depende de las *instacias*.

Sean las funciones $f(n) = \left(\sqrt[3]{27}\right)n^3 + 3n^2$ y $g(n) = n^2\log_2(4) + 2n$, con $n \geq 1$. **Demostrar** la relación de dominancia entre ambas funciones.

Respuesta

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{\left(\sqrt[3]{27}\right)n^3 + 3n^2}{n^2\log_2(4) + 2n}\right) = \lim_{n\to\infty}\left(\frac{3n^2(n+1)}{2n(n+1)}\right) = \lim_{n\to\infty}\left(\frac{3n}{2}\right) = \lim_{n\to\infty}(1.5n) = \infty$$

El límite es $\infty \Longrightarrow f(n)\ domina\ a\ g(n)$.

Planteado al revés

$$\lim_{n\to\infty}\left(\frac{g(n)}{f(n)}\right) = \lim_{n\to\infty}\left(\frac{n^2\log_2(4) + 2n}{\left(\sqrt[3]{27}\right)n^3 + 3n^2}\right) = \lim_{n\to\infty}\left(\frac{2n(n+1)}{3n^2(n+1)}\right) = \lim_{n\to\infty}\left(\frac{2}{3n}\right) = \lim_{n\to\infty}\left(\frac{1.5}{n}\right) = 0$$

El límite es $0 \Longrightarrow g(n)\ es\ domina\ por\ f(n)$

Sea el algoritmo de la figura adjunta. Calcular su complejidad, siendo riguroso en la aplicación y explicación de los pasos y cálculos seguidos.

```
funcion Examen(V[1...m]: vector de enteros; m: entero) retorna entero
    var suma, i: entero fvar
    si (m ≥ 1) entonces
        si (par(m)) entonces
            retorna m + Examen(V, m – 2);
        sino
            suma = 0; i = m;
            mientras (1 ≤ i) hacer
                suma = suma + i;
                i = i – 2;
            fmientras
            retorna suma;
        fsi
    sino
        retorna m;
    fsi
ffuncion
```

Solución

La talla del problema es m: número de elementos del vector *V* que son tratados (su dimensión). El algoritmo no presenta mejor/peor caso: fijada cualquier talla del problema el algoritmo siempre hace el mismo número de pasos, independientemente del contenido (instancias) del vector. Pero sí que, en función de que la talla sea par o $impar$, hace unos pasos u otros. En consecuencia, se debe calcular la más grande/pequeña de la acotación inferior/superior. Se procede entonces a:

- Acotar para el supuesto "m es par", que denotamos por $T_{par}(m)$
  Es un fragmento de código recursivo que mantiene la paridad de m inalterada durante las sucesivas llamadas. Su ecuación es:

$$T_{par}(m) = \begin{cases} c_1 & si\ (1 > m) \\ T_{par}(m-2) + c_2 & si\ (1 \leq m) \end{cases}$$

  En la solución del ejercicio 14 del "*boletín de problemas resueltos*" está el desarrollo correspondiente a la ecuación anterior, resultado que $T_{par}(m) \in \theta(m)$.

- Acotar para el supuesto "m es impar", que denotamos por $T_{impar}(m)$
  Es un fragmento de código iterativo, cuyo número de pasos se puede aproximar por

$$T_{impar}(m) \cong 3 + \sum_{i=0}^{\frac{m}{2}} 2 = 3 + 2\left(\frac{m}{2} + 1\right) = 3 + m + 1 = 4 + m \in \theta(m)$$

  donde el 3 corresponde a las operaciones básicas/elementales "suma = 0", "i = m" y "retorna suma", que se hacen una sola vez, y el 2 se refiere a los pasos "suma = suma + i" e "i = i – 2", cuya ejecución se repite $\left(\frac{m}{2} + 1\right)$ veces. La comparación "$1 \leq i$" no se ha contabilizado al considerarla ligada al "iterador". Si se hubiese hecho, igual de correcto, el número 2 pasaría a ser un 3.
  En este supuesto m es impar y $\frac{m}{2}$ denota la división entera (m div 2). Consecuentemente, $2\left(\frac{m}{2} + 1\right) = 2\left(\frac{m+1}{2}\right) = m + 1$

Al coincidir las cotas se puede concluir que $T(m) \in \theta(m)$.

Curso Académico 2020 – 2021

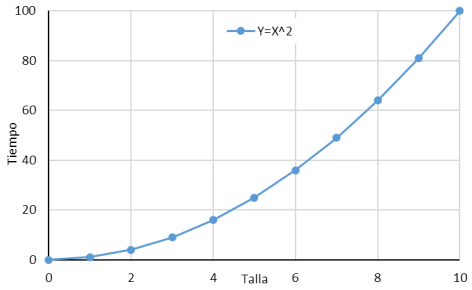
La siguiente ecuación de recurrencia (izquierda) no sabemos resolverla directamente. Completar la ecuación de recurrencia (derecha) que la acota superiormente.

$$T(n) = \begin{cases} c_1 & n \leq 1 \\ T(n-2) + T(ndiv2) + T(n-1) + c_2 & n > 1 \end{cases}$$

Respuesta

$$T(n) = \begin{cases} c_1 & n \leq 1 \\ \boxed{3T(n-1)} + c_2 & n > 1 \end{cases}$$

La figura adjunta representa el tiempo de ejecución (segundos) de una implementación de un algoritmo para algunos valores de la talla. ¿qué complejidad tendrá el algoritmo para la talla 15? Razonar la respuesta.



Respuesta

No se sabe. En análisis empírico o a posteriori no permite extrapolar con seguridad fuera del intervalo de observación. Ejemplo:

```
if (n<11)
  "algo de θ(n²)"
else
  "algo de θ(n)"
```

Sea $f(n) = 10nlog(n) + n^2$ y $g(n) = log(n) + 10n$, ¿Cuál de las siguientes afirmaciones serán ciertas? Enumere la(s) propiedad(es) que avala(n) la respuesta.

a) $\theta(f(n)) * \theta(g(n)) \in \theta(n^3)$
b) $\theta(f(n)) + \theta(g(n)) \in \theta(nlog(n))$
c) $\theta(f(n)) * \theta(g(n)) \in \theta(n^2log(n))$
d) $\theta(f(n)) + \theta(g(n)) \in \theta(n^2)$

Respuesta

a) Regla del producto: $\varphi(f(n)) * \varphi(g(n)) = \varphi(f(n) * g(n))$ con $\varphi = \{\Omega, O, \theta\}$

d) Regla de la suma: $\varphi(f(n)) + \varphi(g(n)) = \varphi(f(n) + g(n)) = \varphi\left(max(f(n), g(n))\right)$ con $\varphi = \{\Omega, O, \theta\}$

Sea el algoritmo de la figura adjunta. Calcular su complejidad, siendo riguroso en la aplicación y explicación de los pasos y cálculos seguidos. **Resolver en una hoja aparte**.

```
{n ≥ 1}
funcion Examen(n: entero) retorna entero
    si (1 = n) entonces retorna 1 sino retorna (Tratar(n)+1)*Examen(n div 2) fsi
ffuncion
funcion Tratar(m: entero) retorna entero
    int thr  thr=m*100
    si (par(thr)) entonces retorna (thr div 2) sino retorna thr fsi
ffuncion
```

Solución

Comenzamos el análisis con la función $Tratar$ llamada por la función principal. $Tratar$ es iterativa y su talla es el valor del parámetro m. $Tratar$ toma decisiones en función de m, de su talla, pero no presenta mejor / peor caso (M/P). Si la talla es par hace 2 pasos (una operación de división y el retorno) que $\in \theta(1)$, y si es $impar$ hace 1 paso (retorno) que también $\in \theta(1)$. Como las acotaciones inferior y superior coinciden $Tratar(m) \in \theta(1)$.

La talla de la función $Examen$ es $n \geq 1$. A priori no tiene M/P. $Tratar$ no presenta M/P, consecuentemente $Examen$ tampoco tiene M/P. $Examen$ es una función recursiva, con ecuación $T(n) = \begin{cases} c_1 & si\ (1 = n) \\ T(n\ div\ 2) + c_2 & si\ (n > 1) \end{cases}$

Se sabe que $Tratar(m) \in \theta(1)$ y que las operaciones $+$ y $*$ son "pasos", por lo que el coste de "$(Tratar(n) + 1) *$ …" queda correctamente modelizado por la constante c_2. Esta ecuación de recurrencia ya está resuelta en los apuntes de teoría/problemas. De forma breve: $T(n) = T(n\ div\ 2) + c_2 = T(n\ div\ 4) + 2c_2 = \cdots = T(n\ div\ 2^i) + ic_2$. La base se alcanza cuando $(n\ div\ 2^i) = 1 \Longrightarrow i = \log_2 n \Longrightarrow T(n) = \cdots = c_1 + \log_2(n)\, c_2 \in \theta(\log_2 n)$

Curso Académico 2019 – 2020

Sean las figuras adjuntas. Completar los recuadros en rojo con información precisa y lo más simple posible para que se cumplan la siguiente afirmación: "el algoritmo iterativo ∈ $\theta(\log_3(n))$ y el recursivo a ∈ $\theta(3^n)$.

```
Función Examen1(…; n: entero)
        var i: entero fvar
        i = 1
        mientras (i < n) hacer
                hacer_algo(i) ∈ θ(1)
                i = i*3
        fmientras
        retorna 0
fprocedimiento
```

$$T(n) = \begin{cases} c_1 & si\ n = 1 \\ \boxed{3T(n-1)} + c_2 & si\ n > 1 \end{cases}$$

Sean las figuras adjuntas. Completar los recuadros en rojo con información precisa y lo más simple posible para que se cumplan la siguiente afirmación: "el algoritmo iterativo ∈ $\theta(\log_4(n))$ y el recursivo a ∈ $\theta(5^n)$.

```
Función Examen1(…; n: entero)
        var i: entero fvar
        i = n
        mientras (i > 1) hacer
                hacer_algo(i) ∈ θ(1)
                i = i div 4
        fmientras
        retorna 0
fprocedimiento
```

$$T(n) = \begin{cases} c_1 & si\ n = 1 \\ \boxed{5T(n-1)} + c_2 & si\ n > 1 \end{cases}$$

Complete la siguiente frase

> "*En el análisis asintótico se debe elegir la **mayor** de las cotas inferiores y la **menor** de las superiores*"

Complete la siguiente frase

> "*En el análisis asintótico se debe elegir la **menor** de las cotas superiores y la **mayor** de las inferiores*"

Sean $f(n) = n^{\frac{3}{2}}, g(n) = n^2\ y\ h(n) = n$. Demostrar las relaciones de dominancia entre las 3 funciones.

$$\lim_{n\to\infty}\left(\frac{g(n)}{f(n)}\right) = \lim_{n\to\infty}\left(\frac{n^2}{n^{\frac{3}{2}}}\right) = \lim_{n\to\infty}\left(\frac{n^2}{n\sqrt{n}}\right) = \lim_{n\to\infty}\left(\sqrt{n}\right) = \infty \Longrightarrow g(n)\ domina\ a\ f(n)$$

$$\lim_{n\to\infty}\left(\frac{f(n)}{h(n)}\right) = \lim_{n\to\infty}\left(\frac{n^{\frac{3}{2}}}{n}\right) = \lim_{n\to\infty}\left(\frac{n\sqrt{n}}{n}\right) = \lim_{n\to\infty}\left(\sqrt{n}\right) = \infty \Longrightarrow f(n)\ domina\ a\ h(n)$$

$$\lim_{n\to\infty}\left(\frac{g(n)}{h(n)}\right) = \lim_{n\to\infty}\left(\frac{n^2}{n}\right) = \lim_{n\to\infty}(n) = \infty \Longrightarrow g(n)\ domina\ a\ h(n)$$

En consecuencia $O(n) \subset O\left(n^{\frac{3}{2}}\right) \subset O(n^2)$ y también $\Omega(n^2) \subset \Omega\left(n^{\frac{3}{2}}\right) \subset \Omega(n)$.

Sea el algoritmo de la figura adjunta. Calcular su complejidad, mostrando todos los cálculos.

```
{n > 1}
Funcion Examen(V[1...n]: vector de enteros; n: entero) retorna (r : entero)
    si (n <= 1) entonces
        retorna V[1]
    sino
        si (V[n]>V[n-1]) entonces
            retorna Examen(V, ndiv2) * Examen(V, ndiv4)
        sino
            retorna Examen(V, n-1) + Examen(V, n-3)
        fsi
    fsi
ffuncion
```

Solución:

Talla: La talla de problema es **n**, la dimensión del vector.

Sí hay mejor y peor caso. El mejor caso es cuando el vector esté ordenado ascendentemente.

Ecuación de recurrencia y complejidad para el mejor caso:

$$T(n)_{MC} = \begin{cases} c_1 & si\ n \leq 1 \\ T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + c_2 & si\ n > 1 \end{cases}$$

Por simplicidad y claridad en la formulación se utilizará $\frac{n}{x}$ para denotar la división entera, esto es, $(n\ div\ x)$. Como se ha visto en clase CEX, la expansión de la ecuación de recurrencia anterior genera 2^i llamadas, de diferentes tamaños, en cada paso, por lo que, al igual que en clase CEX, se procede a su acotación. Por tanto,

$$T(n)_{MC} = \begin{cases} c_1 & si\ n \leq 1 \\ 2T\left(\frac{n}{4}\right) + c_2 & si\ n > 1 \end{cases}$$

Por un lado:

$T(n) = 2T\left(\frac{n}{4}\right) + c_2 = 2\left[2T\left(\frac{n}{16}\right) + c_2\right] + c_2 = 4T\left(\frac{n}{16}\right) + 3c_2 = 4\left[2T\left(\frac{n}{64}\right) + c_2\right] + 3c_2 = 8T\left(\frac{n}{64}\right) + 7c_2 = \cdots$

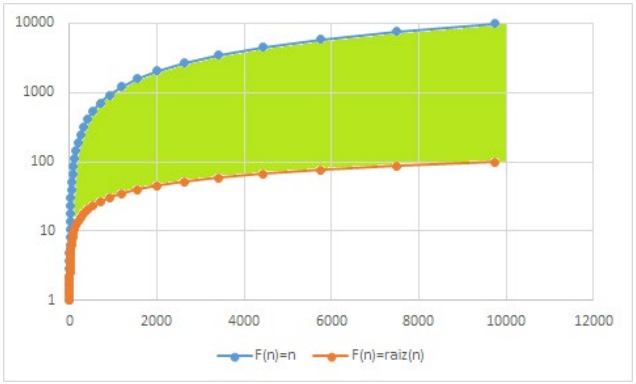
$= 2^i T\left(\frac{n}{4^i}\right) + (2^i - 1)c_2$. Cabe recordar que $4^i = 2^{2i}$ (su equivalente $\log_4(n) = \frac{\log_2(n)}{2}$) y que $a^{\frac{r}{m}} = \sqrt[m]{a^r}$. La base se alcanza cuando $n = 1$, esto es, cuando $i = \log_4(n)$. Por tanto, $T(n) = 2^{\log_4 n}c_1 + \left(2^{\log_4 n} - 1\right)c_2 = (c_1 + c_2)2^{\log_4 n} - c_2 \approx 2^{\log_4 n} = 2^{\frac{\log_2 n}{2}} = \sqrt{2^{\log_2(n)}} = \sqrt{n} \in \theta\left(\sqrt{n}\right)$.

Por el otro:

$T(n) = 2T\left(\frac{n}{2}\right) + c_2 = 2\left[2T\left(\frac{n}{4}\right) + c_2\right] + c_2 = 4T\left(\frac{n}{4}\right) + 3c_2 = 4\left[2T\left(\frac{n}{8}\right) + c_2\right] + 3c_2 = 8T\left(\frac{n}{8}\right) + 7c_2 = \cdots$

$= 2^i T\left(\frac{n}{2^i}\right) + (2^i - 1)c_2$. La base se alcanza cuando $n = 1$, esto es, cuando $i = \log_2(n)$. Por tanto, $T(n) = 2^{\log_2 n}c_1 + \left(2^{\log_2 n} - 1\right)c_2 = (c_1 + c_2)2^{\log_2 n} - c_2 \approx 2^{\log_2 n} = n \in \theta(n)$.

Gráficamente



Cuya interpretación es que la función estará "moviéndose" en la zona sombreada en color verde.

Para el peor caso la ecuación de recurrencia es:

$$T(n)_{PC} = \begin{cases} c_1 & si\ n \leq 1 \\ T(n-1) + T(n-3) + c_2 & si\ n > 1 \end{cases}$$

Sucede lo mismo que en el análisis del mejor caso, por lo que se procede a su acotación.

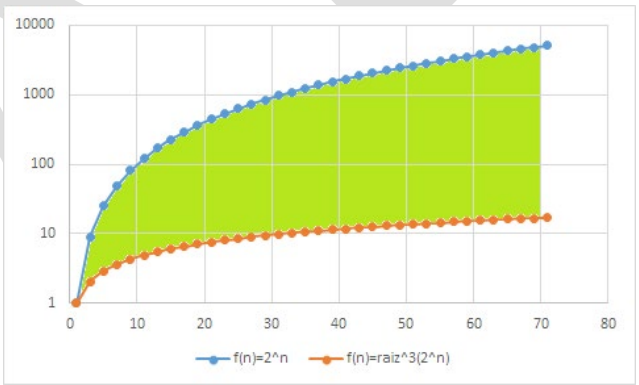
Por un lado:

$T(n) = 2T(n-1) + c_2 = 2[2T(n-2) + c_2] + c_2 = 4T(n-2) + 3c_2 = 4[2T(n-3) + c_2] + 3c_2 = \cdots$
$= 2^i T(n-i) + (2^i - 1)c_2$. La base se alcanza cuando $n = 1$, esto es, cuando $i = n - 1$. Por tanto, $T(n) = 2^{n-1}c_1 + (2^{n-1} - 1)c_2 = (c_1 + c_2)2^{n-1} - c_2 \approx 2^n = n \in \theta(2^n)$

Por el otro:

$T(n) = 2T(n-3) + c_2 = 2[2T(n-6) + c_2] + c_2 = 4T(n-6) + 3c_2 = 4[2T(n-9) + c_2] + 3c_2 = \cdots$
$= 2^i T(n-3i) + (2^i - 1)c_2$. La base se alcanza cuando $n = 1$, esto es, cuando $i = \frac{n-1}{3} \cong \frac{n}{3}$. Por tanto, $T(n) = 2^{\frac{n}{3}}c_1 + \left(2^{\frac{n}{3}} - 1\right)c_2 = (c_1 + c_2)2^{\frac{n}{3}} - c_2 \approx 2^{\frac{n}{3}} = \sqrt[3]{2^n} \in \theta\left(\sqrt[3]{2^n}\right)$

Gráficamente



En consecuencia el algoritmo se caracterizad por: $T(n) \in \Omega\left(\sqrt{n}\right)$ y $T(n) \in O(2^n)$.

Curso Académico 2018 – 2019

Dado el siguiente algoritmo calcular su complejidad, mostrando los cálculos necesarios para su obtención.

```
Funcion Examen(V[1...n]: vector de enteros; n, m: enteros) retorna (res : entero)
   var k, res: entero fvar
   k = 1
   res = 0
   si (n < m) entonces
      mientras (k <= 100) hacer
         res = res + V[k]
         k = k + 1
      fmientras
   sino
      mientras (k <= n) hacer
         res = res + V[k]
         k = k + 1
      fmientras
   fsi
   retorna res
ffuncion
```

Solución:

Talla: La talla de problema es **n**, la dimensión del vector.

No hay mejor y peor caso. La sentencia alternativa ($n < m$) cuestiona el tamaño del problema, de tal modo que no determina mejor y peor caso.

Si ($n < m$), la talla del problema no es lo suficientemente grande, tiene lugar un bucle que siempre itera 100 veces, independientemente de la talla, y en cada iteración realiza 2 sentencias de coste unitario. Es decir $2 + \sum_{k=1}^{100} 2 = 2 + 200 \in \theta(1)$. Nótese que, según la teoría, el análisis asintótico se estudia $\forall n \geq n_0 \ldots$, por tanto, lo relevante, lo importante, lo que hay que estudiar en el análisis de la complejidad es el caso ($n \geq m$).

Cuando ($n \geq m$) tiene lugar un bucle mientras cuya variable k toma valor inicial 1 y n el final. El número de iteraciones depende, por tanto, de la talla del problema. Cada iteración del bucle realiza 2 sentencias de coste unitario, por lo que su complejidad será:

$$2 + \sum_{k=1}^{n} 2 = 2 + 2n \in \theta(n)$$

Por consiguiente, complejidad $\in \theta(n)$.

Anteriores Cursos Académicos

Curso Académico 2017 – 2018

[1 punto] Sea f(n) = $10n^3 + 3$ y se sabe que para $c_1 = 12$ se cumple $10n^3 + 3 \leq c_1 n^3 \ \forall\ n \geq 2$. Y que para $c_2 = 15$ se cumple $10\ n^3 + 3 \leq c_2 n^4 \ \forall\ n \geq 1$. Indica cuáles de las siguientes afirmaciones son ciertas y cuáles son falsas para la función f(n):

a) f(n) es $O(n^4)$
b) f(n) es $O(n^3)$
c) La cota más representativa de f(n) es $O(n^4)$
d) La cota más representativa de f(n) es $O(n^3)$

CIERTAS: a, b, d	**FALSAS: c**

[2 puntos] Completa las siguientes sentencias:

a) $O(n) \subset O(n \log n)$ (*)
b) $\theta(f(n)) + \theta(g(n)) = \theta(\max(f(n), g(n)))$
c) si $f(n) \in \theta(h(n))$ entonces $a\ f(n) + b \in \theta(h(n)) \quad \forall (a \in R^+ \wedge b \in R)$
d) si $t_1(n)$ y $t_2(n)$ son los tiempos de dos implementaciones de un mismo algoritmo, se verifica que $\exists\ c \in \mathbb{R}^+ \wedge \exists\ n_0 \in \mathbb{N} \mid t_1(n) \leq c\ t_2(n)\ \forall\ n \geq n_0$

(*) También era válidas: n^2, n^3, 2^n y n^n

[2.5 puntos] Resuelve la siguiente ecuación de recurrencia a través del método de sustitución, concluyendo con el/los orden/es de complejidad correspondiente/s:

$$T(n) = \begin{cases} c1 & si\ n \leq 2 \\ T(n-1) + T(n-2) + T(ndiv3) + c2 & si\ n > 2 \end{cases}$$

$T(n) = T(n-1)+T(n-2)+T(ndiv3)+c_2 = T(n-2)+T(n-3)+T((n-1)div3) + c_2 +$

$+ T(n-3)+T(n-4)+T((n-2)div3) + c_2 + T(ndiv3-1) + T(ndiv3-2)+T(ndiv3^2)+c_2+c_2=$

…

En la expresión anterior aumenta el número de términos según avanzamos hacia la base, siendo difícil establecer una fórmula general. Para solventar este problema podemos acotar T(n) con las ecuaciones $T_1(n)$ y $T_2(n)$ siguientes:

Acotación superior.-

$$T_1(n) = \begin{cases} c1 & si\ n \leq 2 \\ 3T_1(n-1) + c2 & si\ n > 2 \end{cases}$$

$$T_1(n) = 3T_1(n-1) + c2 = 3[3T_1(n-2) + c2] + c2 = 3^2 T_1(n-2) + 3c2 + c2 =$$

$$= 3^2[3T_1(n-3) + c2] + 3c2 + c2 = 3^3 T_1(n-3) + 3^2 c2 + 3c2 + c2 = \cdots = 3^i T_1(n-i) + \sum_{j=0}^{i-1} 3^j\, c2$$

El caso base se alcanza cuando n-i≤2, por lo que $i \cong n$, de ese modo

$$= 3^n T_1(2) + \sum_{j=0}^{n-1} 3^j\, c2 \in \theta(3^n)$$

Acotación inferior.-

$$T_2(n) = \begin{cases} c1 & si\ n \leq 2 \\ 3T_2(ndiv3) + c2 & si\ n > 2 \end{cases}$$

$$T_2(n) = 3T_2(ndiv3) + c2 = 3[3T_2(ndiv3^2) + c2] + c2 = 3^2T_2(ndiv3^2) + 3c2 + c2 =$$

$$= 3^2[3T_2(ndiv3^3) + c2] + 3c2 + c2 = 3^3T_2(ndiv3^3) + 3^2c2 + 3c2 + c2 = \cdots$$

$$\ldots = 3^iT(ndiv3^i) + \sum_{j=0}^{i-1} 3^j\, c2$$

El caso base se alcanza para $i \cong \log_3 n$, por lo que sustituimos i por su valor en la expresión anterior

$$= 3^{logn}T_2(2) + \sum_{j=0}^{logn-1} 3^j\, c2 \in \theta(n)$$

En conclusión, **$T(n) \in \Omega(n)$ y $T(n) \in O(3^n)$.**

[2.5 puntos] Dada la función Cuestion:

```
Funcion Cuestion ( V[1..num] : vector de enteros; num : entero ) retorna ( s : entero )
var k, p, q: entero fvar

para k = 1 hasta num - 1 hacer
        q = AUX ( V, k, num);
        p = V[ k ]
        V[ k ] = V[ q ]
        V[ q ] = p
fpara
retorna 0
ffuncion

Funcion AUX ( V[1..num] : vector de enteros; inicio, fin: entero) retorna ( t : entero )
var i, m: entero fvar
m = inicio;
para i = inicio + 1 hasta fin hacer
     si ( V[ i ] > V[ m ] )  entonces m = i  fsi
fpara
retorna pos
ffuncion
```

donde la llamada inicial a la función es: **Cuestion(V, num)**
¿Cuál es la talla del problema? ¿Tiene mejor y peor caso?
Calcula su complejidad, mostrando los cálculos necesarios para su obtención.

Funcion AUX.-

Talla del problema: número de componentes de la sección de vector a tratar, es decir, fin – inicio + 1, lo renombraremos como num.

En principio existe mejor y peor caso, debido a la sentencia alternativa si (V[i] > V[m]) ya que en función del cumplimiento de esa condición, o no, se realiza la sentencia m=i.

En el caso de que V[i] <= V[m], sería el mejor caso. En ese supuesto, el coste temporal de la función AUX es el siguiente:

$$T_{MC}(num) = 1 + \sum_{i=inicio+1}^{fin} 1 + 1 = 2 + (fin - inicio - 1 + 1) = 1 + fin - inicio + 1 =$$

$$= 1 + num \in \theta(num)$$

En el caso de que V[i] > V[m], sería el peor caso. En ese supuesto el coste temporal de la función AUX es el siguiente:

$$T_{PC}(num) = 1 + \sum_{i=inicio+1}^{fin} 1 + 1 = 2 + (fin - inicio - 1 + 1) = 1 + fin - inicio + 1 =$$

$$= 1 + num \in \theta(num)$$

En conclusión, ambas complejidades coinciden.

Funcion Cuestión.-

Talla del problema: número de componentes del vector, es decir, num.

$$T(n) = \sum_{k=1}^{num-1} \big(3 + (num - k)\big) = \sum_{k=1}^{num-1} 3 + \sum_{k=1}^{num-1} (num - k) = 3(num - 1) + \sum_{k=1}^{num-1} k =$$

$$= 3num - 3 + \frac{(1 + num - 1)}{2}(num - 1) = 3num + \frac{1}{2}num^2 - \frac{1}{2}num \in \theta(num^2)$$

[2 puntos] Dada la función Cuestion:

```
Q = { 1 ≤ inicio ≤ fin+1 ≤ num + 1 }
Función Cuestion (V[1..num]:vector de enteros; inicio, fin : entero)  retorna ( r : entero )
  var k, p : entero fvar
  p = 1
  si inicio ≥ fin entonces retorna inicio * fin
  si no   k = 1
          mientras k <= 100 hacer
                p = p + V[inicio] * k
                k = k +1
          fmientras
          retorna p * Cuestion (V, inicio + 1, fin – 1 )
  fsi
ffunción
```

donde la llamada inicial a la función es: **Cuestion(V, 1, num)**
¿Cuál es la talla del problema? ¿Tiene mejor y peor caso?
Calcula su complejidad, mostrando los cálculos necesarios para su obtención.

Talla del problema: número de componentes de la sección de vector a tratar, es decir, fin – inicio + 1, lo renombraremos como num.

No existe mejor y peor caso.

La sentencia alternativa (inicio ≥ fin) cuestiona el tamaño del problema, de tal modo que no determina mejor y peor caso. Si inicio ≥ fin, es decir, si la talla del problema es menor o igual que 1, es irrelevante de cara a la eficiencia, pues esta es una propiedad de carácter asintótico.

Para problemas de talla mayor que 1 (inicio < fin), se realiza una llamada recursiva en la que la talla del problema se reduce en dos unidades y previo a la llamada recursiva tiene lugar un bucle *mientras* cuya variable (k) toma un valor inicial de 1 . El número de iteraciones de ese bucle es 100 iteraciones. Siempre se realizan el mismo número de iteraciones, 100, independientemente de la talla, por tanto su complejidad es constante. Dado que en cada iteración del bucle se realizan 2 sentencias de coste unitario, el coste final de esta franja de código es

$$1 + \sum_{i=1}^{100} 2 = 1 + 2(100 - 1 + 1) = 1 + 200 \in \theta(1)$$

Por tanto, la ecuación de recurrencia es

$$T(n) = \begin{cases} c1 & si\ n \leq 1 \\ T(n-2) + c2 & si\ n > 1 \end{cases}$$

expandiendo la recurrencia resulta

$$T(n) = T(n-2) + c2 = T(n-4) + c2 + c2 = T(n-6) + c2 + c2 + c2 =$$

$$= T(n-6) + 3c2 = \cdots = T(n - 2 * i) + ic2$$

El caso base se produce cuando n-2i ≤ 1, por tanto n/2 ≤ i.

Así tenemos que

$$\cong T(0) + \frac{n}{2}c2 = c1 + \frac{n}{2}c2 \in \theta(n)$$

Curso Académico 2016 – 2017

1.- [1 punto] Indicar de las siguientes afirmaciones cuáles son ciertas y cuáles son falsas:

a) Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ entonces f(n) domina asintóticamente a g(n) y no al revés
b) O(n) $\subset$ O(log n)
c) f(n) = n y g(n) = c n con c $\in$ IR+. Se verifica que f(n) $\in$ O(g(n)) y g(n) $\in$ O(f(n))
d) f(n) = n y g(n) = n^2. Se verifica que f(n) $\in$ O(g(n)) y g(n) $\notin$ O(f(n)).

CIERTAS: a, c, d **FALSAS:** b

2.- [2 puntos] Resuelve la siguiente ecuación de recurrencia a través del método de sustitución, concluyendo con el/los orden/es de complejidad correspondiente/s:

$$T(n) = \begin{cases} c1 & si\ n \le 5 \\ T(n-5) + T(n-1) + T(ndiv5) + c2 & si\ n > 5 \end{cases}$$

$$T(n) = T(n-5) + T(n-1) + T(ndiv5) + c2 =$$
$$= T(n-10) + T(n-6) + T((n-5)div5) + c2 + T(n-6) + T(n-2) + T((n-1)div5) +$$
$$+ c2 + T(ndiv5-5) + T(ndiv5-1) + T(ndiv5^2) + c2 + c2 = \cdots$$

En la expresión anterior aumenta el número de términos según avanzamos hacia la base, siendo difícil establecer alguna fórmula general. Para solventar este problema podemos acotar T(n) con las ecuaciones $T_1(n)$ y $T_2(n)$ siguientes:

Acotación superior.-

$$T_1(n) = \begin{cases} c1 & si\ n \le 5 \\ 3T_1(n-1) + c2 & si\ n > 5 \end{cases}$$

Acotación inferior.-

$$T_2(n) = \begin{cases} c1 & si\ n \le 5 \\ 3T_2(ndiv5) + c2 & si\ n > 5 \end{cases}$$

A través del método de sustitución (o expansión de la recurrencia) solucionaremos ambas ecuaciones de recurrencia:

$$T_1(n) = 3T_1(n-1) + c2 = 3[3T_1(n-2) + c2] + c2 = 3^2T_1(n-2) + 3c2 + c2 = \cdots$$
$$= 3^iT_1(n-i) + \sum_{j=0}^{i-1} 3^j\, c2$$

El caso base se alcanza para $i \cong n$ por lo que

$$T_1(n) \cong 3^n c1 + \sum_{j=0}^{n-1} 3^j\, c2 \in \theta(3^n)$$

$$T_2(n) = 3T_2(ndiv5) + c2 = 3[3T_2(ndiv5^2) + c2] + c2 = 3^2T_2(ndiv5^2) + 3c2 + c2 = \cdots$$
$$= 3^iT_2(ndiv5^i) + \sum_{j=0}^{i-1} 3^j\, c2$$

El caso base se alcanza para $i \cong \log_5 n$ por lo que $T_2(n) \in \theta(3^{\log_5 n}) = \theta(n^{\log_5 3}) = \theta(n^{0.68})$.

En conclusión, $T(n) \in \Omega(n^{0.68})$ y $T(n) \in O(3^n)$.

3.- [2 puntos] Dada la función Cuestion3:

```
Función Cuestion ( A[1..n] : vector de enteros; p, q : entero ) retorna ( e : entero )
 si ( p = q ó p = q – 1) entonces retorna p * q
 sino
        si (A[ p ] es par) entonces retorna Cuestion (A, p+1, q-1) * Cuestion (A, p+1, q) * 2
              sino retorna (Cuestion (A, p, (p + q) div 2) * Aux(A, p, q)) / ( p + q )
        fsi
 fsi
ffuncion
```

donde **Aux(A, 1, n) ∈ θ(log n)**

Llamada inicial a la función: **Cuestion(A, 1, n).**

¿Cuál es su talla? n = q – p + 1

¿Tiene mejor y peor caso? ☒ SI ☐ NO

Escribe la/s ecuación/es de recurrencia que corresponda/n

Mejor caso.- A[p] es impar

$$T(n) = \begin{cases} c1 & si\ n \leq 2 \\ T(ndiv2) + log\ n\ c2 + c3 & si\ n > 2 \end{cases}$$

Peor caso.- A[p] es par

$$T(n) = \begin{cases} c1 & si\ n \leq 2 \\ T(n-2) + T(n-1) + c2 & si\ n > 2 \end{cases}$$

4.- [5 puntos] Dada la función Cuestión, determinar su complejidad:

```
Función Cuestion ( V[1..n] : vector de enteros; n : entero ) retorna ( r : entero )
        var  siguiente, resultado: entero fvar
        siguiente = 2; resultado = 0;
        mientras  (siguiente ≤ n)  hacer
                resultado = resultado + Aux ( V, 1, siguiente-1, V[siguiente] )
                siguiente = siguiente + 1
        fmientras
        retorna resultado
ffuncion

Función Aux ( V[1..n] : vector de enteros; p, q, x : enteros ) retorna ( p : entero )
        var s, k: entero fvar
        s = 0;
        si p = q entonces  retorna x
        sino si V[ p ] < x entonces          k = 100
                                             mientras k > 0 hacer
                                                   s = s + k;
                                                   k = k div 2;
                                             fmientras
             sino para k = p hasta q hacer
                        s = s + V[ k ]
                  fpara
             fsi
             retorna s
        fsi
ffuncion
```

Función Aux

¿Cuál es su talla? q – p + 1 = n

¿Tiene mejor y peor caso? [X] SI [ ] NO

¿Cuál es el orden de complejidad resultante? Mostrar los cálculos necesarios para su determinación

Función Aux (Mejor caso).- V [p] < x

$$T_{MC}(n) = 1 + \sum_{i=0}^{\log 100} 2 = 1 + 2(\log 100 + 1)\ \epsilon\ \theta(1)$$

Función Aux (Peor caso).- V [p] ≥ x

$$T_{PC}(n) = 1 + \sum_{k=p}^{q} 1 = 1 + (q - p + 1) = 1 + n\ \epsilon\ \theta(n)$$

Función Cuestion

¿Cuál es su talla? n

¿Tiene mejor y peor caso? [X] SI [ ] NO

La función Cuestion no tiene mejor y peor caso por sí misma, pero dado que invoca a la función Aux que sí que presenta mejor y peor caso, la función Cuestion finalmente presenta mejor y peor caso.

¿Cuál es el orden de complejidad resultante? Mostrar los cálculos necesarios para su determinación

Función Cuestion (Mejor caso debido al Mejor caso de la función Aux).-

$$T_{MC}(n) = 2 + \sum_{i=2}^{n}(1+1) = 2 + 2(n-1) = 2n \ \epsilon \ \theta(n)$$

Función Cuestion (Peor caso debido al Peor caso de la función Aux).-

$$T_{PC}(n) = 2 + \sum_{i=2}^{n}(i+1) \ \epsilon \ \theta(n^2)$$

En conclusión, el coste temporal de la función Cuestion es $\Omega(n)$ y $O(n^2)$.

Curso Académico 2015 – 2016

1. CUESTIONES **(1 punto cada una)**

a) ¿Es cierto que si $f(n) \in \Omega(n^2)$ $entonces$ $f(n) \in O(n^2)$. ¿Por qué?

No, porque si elegimos *f(n)=n³*, se cumpliría que $n^3 \in \Omega(n^2)$ (*n²* es cota inferior de *n³*) pero, en cambio NO se cumpliría que $n^3 \in O(n^2)$ porque *n²* NO es cota superior de *n³*.

b) ¿Qué podemos decir si se cumple que $f(n) \in \Omega(n^2)$ y $f(n) \in O(n^2)$? ¿Por qué?

Que *f(n)* es exactamente de orden *n²* porque *n²* es cota inferior y superior de *f(n)*.

c) Demuestra sin utilizar límites que $f(n) = n^2 + 1$ es del mismo orden que $f(n) = 3n^2 + 3$

Veremos que n^2+1 es cota superior de $3n^2+3$ y que $3n^2+3$ es cota superior de n^2+1.
Efectivamente, $\exists\ c \in \mathbf{R}^+ \wedge \exists\ n_0 \in \mathbf{IN} \mid n^2+1 \geq c(3n^2+3)\ \forall\ n \geq n_0$.
Basta elegir c=1/3 y n_0**=1**
También se cumple $\exists\ c \in \mathbf{R}^+ \wedge \exists\ n_0 \in \mathbf{IN} \mid 3n^2+3 \geq c(n^2+1)\ \forall\ n \geq n_0$.
Basta elegir c=1 y n_0**=1**

d) ¿ $log(n)$ y $(\log n)^2$ representan órdenes de complejidad equivalentes? Demostrar la respuesta.

No, porque se cumple que:

$$\lim_{n\to\infty} \frac{(log\ n)^2}{log\ n} = \lim_{n\to\infty} log\ n = \infty$$

Por tanto, $(\log n)^2$ crece más rápidamente que log n, esto es, $(\log n)^2$ domina asintóticamente a log n.

2. Calcular el orden de complejidad del siguiente algoritmo iterativo **(4 puntos)**

```
Funcion Ejercicio1(num: entero, v[1..num]: vector de enteros) retorna entero
  var aux: entero  fvar
  para i desde 2 hasta num  hacer
       aux=v[i]
       j=i;
       mientras ( j > 1 && v[j-1]>aux ) hacer
          v[j]=v[j-1];
          j--;
       fmientras
       v[j]=aux;
  fpara
  retorna 0
ffuncion
```

Talla: num

Existe mejor y peor caso: Sí

Instancia de Mejor caso: los elementos del vector están en orden creciente, por lo que el bucle mientras no se realiza nunca.

$$T_{Mejor\_caso}(num) = \sum_{i=2}^{num} 3 + 1 = 3(num-1) + 1 \in \theta(num)$$

Instancia de Peor caso: v[1]>v[2], v[2]>v[3], v[3]>v[4], …, esto es, los elementos del vector están en orden estrictamente decreciente. El bucle mientras se ejecuta desde 2 hasta i en todas las iteraciones del bucle para, donde i es la variable de dicho bucle.

$$T_{Peor\_caso}(num) = \sum_{i=2}^{num}\left(3 + \sum_{j=2}^{i} 2\right) + 1 = \sum_{i=2}^{num}(3 + 2(i-1)) + 1 =$$

$$= \sum_{i=2}^{num} 3 + \sum_{i=2}^{num} 2(i-1) + 1 = 3(num-1) + 2\sum_{i=1}^{num-1} i + 1 =$$

$$= 3num - 3 + 2\frac{(num-1)(1+num-1)}{2} + 1 \in \theta(num^2)$$

Por lo tanto, $T(n) \in \Omega(num)$ y $T(n) \in O(num^2)$

3. Calcular el orden de complejidad del siguiente algoritmo recursivo **(2 puntos)**

```
Función   Ejercicio2 (n: entero)  retorna (b:entero)
   si   (n=1) O (n=2) entonces retorna n
   sino retorna  2*Ejercicio2(n-1)+3*Ejercicio2(n-2);
   fsi
ffunción
```

Talla: n

Existe mejor y peor caso: No

La ecuación de recurrencia resultante sería:

$$T(n) = \begin{cases} c1 & si\ \ n \leq 2 \\ T(n-1) + T(n-2) + c2 & si\ n > 2 \end{cases}$$

$$\mathrm{T(n)} = \mathrm{T(n-1)} + \mathrm{T(n-2)} + \mathrm{c2} =$$

$$= \mathrm{T(n-2)} + \mathrm{T(n-3)} + \mathrm{c2} + \mathrm{T(n-3)} + \mathrm{T(n-4)} + \mathrm{c2} + \mathrm{c2} = \ldots$$

En la expresión anterior aumenta el número de términos según avanzamos hacia la base, siendo difícil establecer alguna fórmula general. Para solventar este problema podemos acotar T(n) con las ecuaciones T1(n) y T2(n) siguientes:

$$T_1(n) = \begin{cases} c1 & si\ \ n \leq 2 \\ 2T_1(n-2) + c2 & si\ n > 2 \end{cases} \quad \text{y} \quad T_2(n) = \begin{cases} c1 & si\ \ n \leq 2 \\ 2T_2(n-1) + c2 & si\ n > 2 \end{cases}$$

Procedemos a aplicar el método de expansión de la recurrencia para resolver las ecuaciones T1 y T2.

$$\mathrm{T_1(n)} = 2\mathrm{T_1(n-2)} + \mathrm{c2} = 2[2\mathrm{T_1(n-4)} + \mathrm{c2}] + \mathrm{c2} = 2^2\mathrm{T_1(n-4)} + 3\mathrm{c2} =$$

$$= 2^2[2T_1(n-6) + c2] + 3c2 = 2^3T_1(n-6) + 7c2 = \cdots$$

$$= 2^iT_1(n-2i) + \left(2^i - 1\right)c2 = \cdots$$

Alcanzaríamos la base cuando n-2i ≤ 2 y esto sucede cuando i≈n/2. Reemplazamos i por n/2 en la expresión anterior y tendremos

$\cong 2^{n/2}c1 + \left(2^{n/2} - 1\right)c2 \in \theta(2^{n/2})$

$T_2(n) = 2T_2(n-1) + c2 = 2[2T_2(n-2) + c2] + c2 = 2^2T_2(n-2) + 3c2 =$

$= 2^2[2T_2(n-3) + c2] + 3c2 = 2^3T_2(n-3) + 7c2 = \cdots$

$= 2^iT_2(n-i) + \left(2^i - 1\right)c2 = \cdots$

Alcanzaríamos la base cuando n-i ≤ 2 y esto sucede cuando i≈n. Reemplazamos i por n en la expresión anterior y tendremos

$\cong 2^nc1 + (2^n - 1)c2 \in \theta(2^n)$

En conclusión, $T_1(n) \in \theta(2^{n/2})$ y $T_2(n) \in \theta(2^n)$

$T_1(n)$ y $T_2(n)$ no son órdenes equivalentes porque $\lim_{n\to\infty} \frac{2^n}{2^{nDIV2}} = \lim_{n\to\infty} 2^{nDIV2} = \infty$

Por lo tanto, $T(n) \in \Omega\left(2^{n/2}\right)$ y $T(n) \in O(2^n)$