RECURRENCIA

1. Definir la **ecuación de recurrencia** y su **solución** para estos algoritmos. Para cada algoritmo hay que indicar:

   a) Tamaño del problema:
   b) Coste del caso base:
   c) Coste del resto de instrucciones no recursivas:
   d) Número de llamadas recursivas que se realizan en cada pasada:
   e) Tamaño de cada llamada recursiva:
   f) Ecuación de Recurrrencia. Es la suma de:
   - coste no recursivo del algoritmo: b)+c)
   - coste recursivo del algoritmo

   g) Solución de la ecuación :

   Algoritmos Potencia Calculan X^n , sea n potencia de 2.

PotenciaVersion1(X, n)

a) IF n==1 Devolver X
b) ELSE
  1) $mitad \leftarrow n/2$
  2) $mitadPot \leftarrow$ **PotenciaVersion1**(X, $mitad$)
  3) IF n es par
     Devolver $mitadPot \times mitadPot$
  4) ELSE (n impar)
     Devolver $X \times mitadPot \times mitadPot$

Solución:

En primer lugar calcularemos la ecuación de recurrencia asociada al algoritmo. Y después la resolvemos.

- Tamaño del problema: el tamaño es n
- Coste del caso base: línea a) operaciones de tiempo cte: $\mathcal{O}(1)$
- Coste del resto de instrucciones no recursivas:
  línea 1, asignación en línea 2, sentencia condicional en líneas 3-4 : todas son operaciones de tiempo constante: $\mathcal{O}(1)$
- Número de llamadas recursivas que se realizan en cada pasada:
  solo 1 llamada recursiva en la línea 2
- Tamaño de cada llamada recursiva:
  el parámetro es $mitad$ y es igual a $n/2$
- Ecuación de Recurrrencia:

$$T(n) = costeRecursivo + costeNoRecursivo$$

- Coste recursivo del algoritmo: una llamada recursiva de tamaño $n/2$, por tanto su coste será: $T(n/2)$
- Coste no recursivo del algoritmo: máximo de los costes de las instrucciones no recursivas (incluido el caso base): como todas eran de orden constante, el coste no recursivo es de orden $\mathcal{O}(1)$

La ecuación de recurrencia es :

$$T(n) = T(n/2) + 1$$

- Solución de la ecuación : Usamos el Teorema Maestro:

. $a = 1$ llamada recursiva de tamaño $n/2$

. $b = 2$

. $f(n) = 1$

. Calculamos $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$
y comparamos $f(n)$ con 1:

- Como $f(n) = 1 \in \Theta(n^{\log_b a})$, es decir, son iguales, entonces aplicamos Caso 2 ya que $\Theta(f(n)) = \Theta(n^{\log_b a})$

- Y la solución a la recurrencia es:

$$T(n) \in \Theta(n^{\log_b a} \log n) \Rightarrow T(n) \in \Theta(\log n)$$

Por tanto, el orden del algoritmo **PotenciaVersion1**(n) es :

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

PotenciaVersion2(X, n)

a) IF n==1 Devolver X

b) ELSE

1) $mitad \leftarrow n/2$
2) IF n es par
   Devolver **PotenciaVersion2**(X,$mitad$)×**PotenciaVersion2**(X,$mitad$)
3) ELSE (n impar)
   Devolver X×**PotenciaVersion2**(X,$mitad$)×**PotenciaVersion2**(X,$mitad$)

Solución:

En primer lugar calcularemos la ecuación de recurrencia asociada al algoritmo. Y después la resolvemos.

- Tamaño del problema: el tamaño es n
- Coste del caso base: línea a) operaciones de tiempo cte: $\mathcal{O}(1)$
- Coste del resto de instrucciones no recursivas:
  línea 1, operaciones primitivas en comparación if-else,
  operación producto: todas son operaciones de tiempo constante: $\mathcal{O}(1)$
- Número de llamadas recursivas que se realizan en cada pasada:
  Las llamadas recursivas están dentro de un condicional. Por tanto no se van a realizar las cuatro que aparecen en el código. Las posibilidades son:
  - Si se verifica la condición (n es par) : se realizan 2 llamadas recursivas de tamaño $n/2$ cada una.
  - Si NO se verifica la condición (n es impar) : se realizan 2 llamadas recursivas de tamaño $n/2$ cada una.
- Por tanto solo se van a realizar 2 llamadas recursivas de tamaño $n/2$, en cualquier caso.
- Ecuación de Recurrrencia:

$$T(n) = costeRecursivo + costeNoRecursivo$$

  - Coste recursivo del algoritmo: 2 llamadas recursivas de tamaño $n/2$, por tanto su coste será: $T(n/2) + T(n/2)$
  - Coste no recursivo del algoritmo: máximo de los costes de las instrucciones no recursivas (incluido el caso base): $\mathcal{O}(1)$

  La ecuación de recurrencia es :

$$T(n) = 2T(n/2) + 1$$

- Solución de la ecuación : Usamos el Teorema Maestro:

. $a = 2$

. $b = 2$

. $f(n) = 1$

. Calculamos $n^{\log_b a} = n^{\log_2 2} = n^1 = n$
y comparamos $f(n)$ con n:

- Como $f(n) \in \mathcal{O}(n)$, comprobamos las condiciones del caso 1

$\Rightarrow$ Se debe cumplir que:

- $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ para $\epsilon > 0$. Sustituyendo:
- Se debe cumplir que
  $1 \in \mathcal{O}(n^{1-\epsilon})$, para un $\epsilon > 0$
- y es cierto para $\epsilon = 1$,

▪ Podemos aplicar el Caso 1 y por tanto la solución a la recurrencia es:

$$T(n) \in \Theta(n^{\log_2 2}) \Rightarrow T(n) \in \Theta(n)$$

Por tanto, el orden del algoritmo **PotenciaVersion2**(n) es :

$$T(n) \in \Theta(n)$$

PotenciaVersion3(X, n)

a) IF n==1 Devolver X

b) ELSE

Devolver $X\times$ **PotenciaVersion3**$(X,\ n-1)$

Solución:

En primer lugar calcularemos la ecuación de recurrencia asociada al algoritmo. Y después la resolvemos.

- Tamaño del problema: el tamaño es n
- Coste del caso base: línea a) operaciones de tiempo cte: $\mathcal{O}(1)$
- Coste del resto de instrucciones no recursivas:
  Solo los productos, de tiempo constante: $\mathcal{O}(1)$
- Número de llamadas recursivas que se realizan en cada pasada:
  solo 1 llamada recursiva de tamaño $n-1$.
- Ecuación de Recurrrencia:

$$T(n) = costeRecursivo + costeNoRecursivo$$

  - Coste recursivo del algoritmo: una llamada recursiva de tamaño $n-1$, por tanto su coste será: $T(n-1)$
  - Coste no recursivo del algoritmo: máximo de los costes de las instrucciones no recursivas (incluido el caso base): $\mathcal{O}(1)$

  La ecuación de recurrencia es :

$$T(n) = T(n-1) + 1$$

- Solución de la ecuación : No podemos resolverla con el Teorema Maestro porque no se va dividiendo n entre llamadas recursivas, sino que le quitamos una unidad. Para resolver esta ecuación usaremos la Fórmula 2 de los apuntes, según la cual la solución es:

$$T(n) \in \mathcal{O}(n)$$

Por tanto, el orden del algoritmo **PotenciaVersion3**(n) es :

$$T(n) \in \mathcal{O}(n)$$

2. ¿Por qué el algoritmo **PotenciaVersion2** es más costoso que **PotenciaVersion1**?

PotenciaVersion1(X, n)

a) IF n==1 Devolver X

b) ELSE

1) $mitad \leftarrow n/2$
2) $mitadPot \leftarrow$ **PotenciaVersion1**(X, $mitad$)
3) IF n es par
   Devolver $mitadPot \times mitadPot$
4) ELSE (n impar)
   Devolver $X \times mitadPot \times mitadPot$

PotenciaVersion2(X, n)

a) IF n==1 Devolver X

b) ELSE

1) $mitad \leftarrow n/2$
2) IF n es par
   Devolver **PotenciaVersion2**(X,$mitad$)×**PotenciaVersion2**(X,$mitad$)
3) ELSE (n impar)
   Devolver X×**PotenciaVersion2**(X,$mitad$)×**PotenciaVersion2**(X,$mitad$)

Solución:

El primer algoritmo es más eficiente, es de orden $T(n) \in \Theta(\log n)$, mientras que el segundo es de orden $T(n) \in \Theta(n)$. Los dos calculan el mismo valor.

El problema del segundo algoritmo es que se hacen llamadas recursivas innecesarias, en las que además se recalculan los mismos valores.

No es necesario llamar 2 veces al método recursivo, es mejor la solución del primer algoritmo, ya que guarda el valor devuelto por el método en una variable ($mitadPot$) y en las líneas siguientes utiliza ese valor ya calculado, y por tanto solo hace una llamada recursiva.

3. Calcular el tiempo de ejecución en el caso peor de los siguientes algoritmos usando el método maestro para resolver la ecuación de recurrencia.

DivideMatriz(MM, n)

INPUT: **MM**: matriz de enteros , **n**: dimensión $(n \times n)$
OUTPUT: Divide una matriz cuadrada en 4 submatrices

a) A, B, C, D : matriz de enteros $[n/2, n/2]$
b) IF $n < 2$ **return**
c) $mitad \leftarrow n/2$
d) FOR $i := 1$ to $mitad$
  FOR $j := 1$ to $mitad$
  . $A[i][j] \leftarrow MM[i][j]$
  . $B[i][j] \leftarrow MM[i][j + mitad]$
  . $C[i][j] \leftarrow MM[i + mitad][j]$
  . $D[i][j] \leftarrow MM[i + mitad][j + mitad]$
e) **DivideMatriz**$(A, mitad)$
f) **DivideMatriz**$(B, mitad)$
g) **DivideMatriz**$(C, mitad)$
h) **DivideMatriz**$(D, mitad)$

Solución:

En primer lugar calcularemos la ecuación de recurrencia asociada al algoritmo. Y después la resolvemos.

- Tamaño del problema: el tamaño es n
- Coste de instrucciones no recursivas:
  - línea b) caso base: operaciones de tiempo cte: $\mathcal{O}(1)$
  - línea c) tiempo cte: $\mathcal{O}(1)$
  - línea d) Dos bucles for anidados: $\mathcal{O}(n^2)$. Lo demostramos:
    - Las operaciones dentro del segundo bucle son de tiempo constante, su tiempo lo acotamos por c.

  Entonces:

$$T_{bucle} \leq \sum_{i=1}^{n/2} \sum_{j=1}^{n/2} c$$

  Resolvemos los sumatorios desde el más interno al más externo:

  - El más interno: $\sum_{j=1}^{n/2} c \quad = c.n/2$
  - y sustituimos

$$T_{bucle} \leq \sum_{i=1}^{n/2} c.\frac{n}{2}$$

$$T_{bucle} \leq \frac{n}{2}.\frac{n}{2}.c \in \mathcal{O}(n^2)$$

- Número de llamadas recursivas que se realizan:
  4 llamadas recursivas de tamaño $n/2$.
- Ecuación de Recurrrencia:

$$T(n) = costeRecursivo + costeNoRecursivo$$

  - Coste recursivo del algoritmo: 4 llamadas recursivas de tamaño $n/2$, por tanto su coste será: $4T(n/2)$
  - Coste no recursivo del algoritmo: máximo de los costes de las instrucciones no recursivas (incluido el caso base): $\mathcal{O}(n^2)$

  La ecuación de recurrencia es :

$$T(n) = 4T(n/2) + n^2$$

- Solución de la ecuación : Usamos el Teorema Maestro:

. $a = 4$

. $b = 2$

. $f(n) = n^2$

. Calculamos $n^{\log_b a} = n^{\log_2 4} = n^2$
  y comparamos $f(n)$ con n^2:
  - Como $f(n) = n^2 \in \Theta(n^{\log_b a})$, es decir, son iguales, entonces aplicamos Caso 2.

- Y la solución a la recurrencia es:

$$T(n) \in \Theta(n^{\log_b a} \log n) \Rightarrow T(n) \in \Theta(n^2 \log n)$$

Por tanto, el orden del algoritmo **DivideMatriz**(n) es :

$$T(n) \in \Theta(n^2 \log n)$$

BuscaMAX(A, $prim$, ult)

INPUT: **A**: array de enteros $A[prim, .., ult]$
OUTPUT: Máximo elemento del vector

a) IF $prim == ult$ **return** $A[1]$

b) $mitad \leftarrow (prim + ult)/2$
c) $m1 \leftarrow$ **BuscaMAX**$(A, prim, mitad)$
d) $m2 \leftarrow$ **BuscaMAX**$(A, mitad + 1, ult)$
e) IF $m1 > m2$ **return** $m1$
ELSE **return** $m2$

Solución:

En primer lugar calcularemos la ecuación de recurrencia asociada al algoritmo. Y después la resolvemos.

- Tamaño del problema: el tamaño es $ult - prim = n$
- Coste del caso base: línea a) operaciones de tiempo cte: $\mathcal{O}(1)$
- Coste del resto de instrucciones no recursivas:
  líneas b) y e), operaciones primitivas en comparación if-else: todas son operaciones de tiempo constante: $\mathcal{O}(1)$
- Número de llamadas recursivas que se realizan en cada pasada:
  Se realizan 2 llamadas recursivas de tamaño $n/2$.
- Ecuación de Recurrrencia:

$$T(n) = costeRecursivo + costeNoRecursivo$$

  - Coste recursivo del algoritmo: 2 llamadas recursivas de tamaño $n/2$, por tanto su coste será: $T(n/2) + T(n/2)$
  - Coste no recursivo del algoritmo: máximo de los costes de las instrucciones no recursivas (incluido el caso base): $\mathcal{O}(1)$

  La ecuación de recurrencia es :

$$T(n) = 2T(n/2) + 1$$

- Solución de la ecuación : Usamos el Teorema Maestro: Ya resuelta en un ejercicio anterior.

- Se aplica el Caso 1 y la solución a la recurrencia es:

$$T(n) \in \Theta(n)$$

Por tanto, el orden del algoritmo **BuscaMAX**(n) es :

$$T(n) \in \Theta(n)$$

Igual que la versión iterativa.

4. Obtener el orden de complejidad del tiempo de ejecución $T(n)$ en las siguientes recurrencias. Indicar, en cada una de ellas:

   - Número de llamadas recursivas
   - Tamaño de las llamadas recursivas
   - Coste no-recursivo y coste recursivo
   - Orden de complejidad de $T(n)$ (según método maestro):

a) $T(n) = 2T(n/2) + n^3$

. $a = 2$ (2 llamadas recursivas de tamaño $n/2$)

. $b = 2$

. $f(n) = n^3$ (coste no recursivo)

. Calculamos $n^{\log_b a} = n^{\log_2 2} = n^1 = n$
y comparamos $f(n)$ con n

- Como $f(n) = n^3 \in \Omega(n^{\log_2 2})$, es decir $n^3 \in \Omega(n)$, comprobamos el caso 3:

$\Rightarrow$ Se deben cumplir dos condiciones:

i) $f(n) \in \Omega(n^{\log_b a+\epsilon})$
es decir, $n^3 \in \Omega(n^{1+\epsilon})$: cierto para $\epsilon = 1$

ii) Condición de regularidad para $f(n)$.
Para n suficientemente grande,

$$af(n/b) \leq cf(n), \quad c < 1$$

$$af(\frac{n}{b}) = 2\left(\frac{n}{2}\right)^3 \leq c.n^3$$

$$2.\frac{n^3}{8} = \frac{n^3}{4} \leq c.n^3$$

cierto para $c = 1/4 < 1$.

- La solución es: $T(n) \in \Theta(f(n)) \Rightarrow T(n) \in \Theta(n^3)$

b) $T(n) = 7T(n/3) + n^2$

. $a = 7$ (7 llamadas recursivas de tamaño $n/3$)

. $b = 3$

. $f(n) = n^2$ (coste no recursivo)

. Calculamos $n^{\log_b a} = n^{\log_3 7} = n^{1,77}$
y comparamos $f(n)$ con $n^{1,77}$

- Como $f(n) = n^2 \in \Omega(n^{\log_3 7})$, es decir $n^2 \in \Omega(n^{1,77})$, comprobamos el caso 3:

$\Rightarrow$ Se deben cumplir dos condiciones:

i) $f(n) \in \Omega(n^{\log_b a+\epsilon})$
es decir, $n^2 \in \Omega(n^{1,77+\epsilon})$: cierto para $\epsilon = 0{,}2$

ii) Condición de regularidad para $f(n)$.
Para n suficientemente grande,

$$af(n/b) \leq cf(n), \quad c < 1$$

$$af(\frac{n}{b}) = 7\left(\frac{n}{3}\right)^2 \leq c.n^2$$

$$7.\frac{n^2}{9} \leq c.n^2$$

cierto para $c = 7/9 < 1$.

- La solución es: $T(n) \in \Theta(f(n)) \Rightarrow T(n) \in \Theta(n^2)$

c) $T(n) = 7T(n/2) + n^2$

. $a = 7$ (7 llamadas recursivas de tamaño $n/2$)

. $b = 2$

. $f(n) = n^2$ (coste no recursivo)

. Calculamos $n^{\log_b a} = n^{\log_2 7} = n^{2,80}$
y comparamos $f(n)$ con $n^{2,80}$:

- Como $f(n) \in \mathcal{O}(n^{2,80})$, comprobamos si se cumplen las condiciones del caso 1

$\Rightarrow$ Se debe cumplir que:

- $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ para $\epsilon > 0$. Sustituyendo:
- Se debe cumplir que
  $n^2 \in \mathcal{O}(n^{2,80-\epsilon})$, para un $\epsilon > 0$
- y es cierto para por ejemplo $\epsilon = 0{,}5$

▪ Podemos aplicar el Caso 1 y por tanto la solución a la recurrencia es:

$$T(n) \in \Theta(n^{\log_2 7}) \Rightarrow T(n) \in \Theta(n^{2,80})$$

d) $T(n) = 16T(n/4) + n^2$

. $a = 16$ (16 llamadas recursivas de tamaño $n/4$)

. $b = 4$

. $f(n) = n^2$ (coste no recursivo)

. Calculamos $n^{\log_b a} = n^{\log_4 16} = n^2$
y comparamos $f(n)$ con n^2:

- Como $f(n) = n^2 \in \Theta(n^{\log_b a})$, es decir, entonces aplicamos Caso 2.

▪ Y la solución a la recurrencia es:

$$T(n) \in \Theta(n^{\log_b a} \log n) \Rightarrow T(n) \in \Theta(n^2 \log n)$$

5. Tenemos dos algoritmos para resolver un problema. Los tiempos de ejecución de estos algoritmos se describen respectivamente con las recurrencias:

   a) $T(n) = 8T(n/2) + n^2$

   b) $T'(n) = aT'(n/4) + n^2$

   ¿Cuál es el valor máximo que puede tener a para que el segundo algoritmo sea asintóticamente más rápido que el primero?

   Solución:

   - Primero calculamos la solución de la primera recurrencia:

   . $a = 8$

   . $b = 2$

   . $f(n) = n^2$

   . Calculamos $n^{\log_b a} = n^{\log_2 8} = n^3$
   y comparamos $f(n)$ con n^3:

   - Como $f(n) = n^2 \in \mathcal{O}(n^{\log_b a})$, comprobamos el caso 1:
   - Se debe cumplir que $n^2 \in \mathcal{O}(n^{3-\epsilon})$, para un $\epsilon > 0$
     cierto para $\epsilon = 1$

   ▪ Y la solución a la primera recurrencia es:

   $$T(n) \in \Theta(n^{\log_b a}) \Rightarrow T(n) \in \Theta(n^3)$$

   - Ahora vemos las diferentes soluciones que podemos tener para la segunda, dependiendo del valor de a. Necesitamos que la solución de T' sea de un orden menor que T, es decir, menor que n^3.

   . $a = ?$

   . $b = 4$

   . $f(n) = n^2$

   . Calculamos $n^{\log_b a} = n^{\log_4 a} = n^k$

   ▪ Si solucionamos la segunda recurrencia **aplicando el caso 3**:

   - su orden sería

   $$T'(n) \in \Theta(f(n)) \Rightarrow T'(n) \in \Theta(n^2)$$

   - y se cumpliría que $f(n) = n^2 \in \Omega(n^k)$
   - Por lo que $k = \log_4 a$ tendría que ser $k < 2$, y por tanto el máximo valor para a sería:
     $a = 15$

   Y se verifica que el segundo algoritmo es más rápido que el primero, ya que $n^2 \in \mathcal{O}(n^3)$

   ▪ Si solucionamos la segunda recurrencia **aplicando el caso 2**:

- su orden sería

$$T'(n) \in \Theta(f(n).\log n) \Rightarrow T'(n) \in \Theta(n^2.\log n)$$

- y se cumpliría que $f(n) = n^2 \in \Theta(n^k)$
- Por lo que $k = \log_4 a$ tendría que ser $k = 2$, y por tanto el máximo valor para a sería:
  $a = 16$

Y se verifica que el segundo algoritmo es más rápido que el primero, ya que $n^2 \log n \in \mathcal{O}(n^3)$

- Si solucionamos la segunda recurrencia **aplicando el caso 1**:
  - su orden sería

$$T'(n) \in \Theta(n^{\log_b a}) \Rightarrow T'(n) \in \Theta(n^{\log_4 a})$$

  - y se cumpliría que $f(n) = n^2 \in \mathcal{O}(n^k)$
  - Por lo que $k = \log_4 a$ tendría que ser $k > 2$.
  - Como buscamos que el orden del segundo algoritmo sea menor que el primero (que es n^3) entonces:
    $2 < k < 3$
    por tanto el máximo valor para a sería:
    $a = 63$ (ya que no podemos llegar a $\log_4 64 = 3$)
    y en ese caso: $k = \log_4 63 = 2{,}8$

Y se verifica que el segundo algoritmo es más rápido que el primero, ya que $n^{2,8} \in \mathcal{O}(n^3)$