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 $X \times mitadPot \times mitadPot$

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,$mitad$)

PotenciaVersion3(X, n)

a) IF n==1 Devolver X
b) ELSE
     Devolver $X\times$ **PotenciaVersion3**(X, $n-1$)

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

3. Calcular el tiempo de ejecución en el caso peor de los siguientes algoritmos usando el método maestro para resolver la ecuación de recurrencia. En cada uno de ellos, hay que indicar:

   a) Tamaño del problema
   b) Número de llamadas recursivas en cada pasada (a)
   c) Tamaño de las llamadas recursivas (n/b)
   d) Coste no-recursivo del algoritmo
   e) Ecuación de Recurrencia
   f) Solución (según fórmula)

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**$(D, mitad)$

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** $m2$

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) $T(n) = 2T(n/2) + n^3$
   b) $T(n) = 7T(n/3) + n^2$
   c) $T(n) = 7T(n/2) + n^2$
   d) $T(n) = 16T(n/4) + n^2$

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?

6. Obtener el árbol de recursión y la solución de las siguientes recurrencias:

   a) $T(n) = T(n/4) + T(n/2) + n^2$
   b) $T(n) = T(n/3) + T(2n/3) + n$