Análisis de Algoritmos

1. Indicar el orden de complejidad de este algoritmo en el peor caso.

Algoritmo CheckOrder Dado un array A, comprueba si está ordenado de forma ascendente.

CheckOrder(A, n)

INPUT: **A**: array de enteros $A[1,..,n]$
OUTPUT: **True**, si $A[1] \leq A[2] \leq \cdots \leq A[n]$

1. FOR $i := 1$ to n
   . IF $A[i] > A[i+1]$ Devolver $False$
2. Devolver $True$

Solución:

- línea 2: operaciones primitivas de tiempo constante: $\mathcal{O}(1)$
- línea 1: Bucle for : $\mathcal{O}(n)$. Lo demostramos:
  Las operaciones dentro del bucle son de tiempo constante, su tiempo lo acotamos por c. Entonces:

$$T_{bucle} \leq \sum_{i=1}^{n} c \quad = n.c \quad \Rightarrow T_{bucle} \in \mathcal{O}(n)$$

- El orden del algoritmo es del orden del máximo de esas funciones, por tanto:

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

2. Indicar el orden de complejidad de este algoritmo que realiza la multiplicación de dos matrices por el método de "fuerza bruta".

ProductMatrix(A, B, n)

INPUT: $\mathbf{A}$, $\mathbf{B}$: matrices a multiplicar de dimensión $n \times n$
OUTPUT: $\mathbf{C}$: matriz resultante

```
1. FOR i := 1 to n
2.    FOR j := 1 to n
3.       suma ← 0
4.       FOR k := 1 to n
5.          suma ← suma + A[i,k]B[k,j]
6.       C[i,j] ← suma
7. Devolver C
```

Solución:

- Siempre se ejecutan las mismas instrucciones independientemente de la entrada.
- línea 7: operaciones primitivas de tiempo constante: $\mathcal{O}(1)$
- líneas 1-6: Bucles for anidados: $\mathcal{O}(n^3)$. Lo demostramos:
  - Las operaciones dentro del segundo bucle (líneas 3 y 6) son de tiempo constante, su tiempo lo acotamos por c.
  - Las operaciones dentro del tercer bucle (línea 5) son de tiempo constante, su tiempo lo acotamos por d.

  Entonces:

$$T_{bucle} \leq \sum_{i=1}^{n} \sum_{j=1}^{n} \left(c + \sum_{k=1}^{n} d \right)$$

  Resolvemos los sumatorios desde el más interno al más externo:

  - El más interno: $\sum_{k=1}^{n} d \quad = n.d$
  - y sustituimos:

$$T_{bucle} \leq \sum_{i=1}^{n} \sum_{j=1}^{n} (c + n.d)$$

  - Calculamos el siguiente sumatorio: $\sum_{j=1}^{n} (c+n.d) = n(c+n.d)$
  - y sustituimos

$$T_{bucle} \leq \sum_{i=1}^{n} n(c + n.d)$$

$$T_{bucle} \leq n.n(c + n.d) \in \mathcal{O}(n^3)$$

- El orden del algoritmo es del orden del máximo de esas funciones, por tanto:

$$T(n) \in \mathcal{O}(n^3)$$

3. Indicar el orden de complejidad de este algoritmo.

Algoritmo PrefixMedias Dado un array X, la i-ésima media de prefijo A_i es la media de los $(i+1)$ primeros elementos del vector. Por ejemplo:

$$A_i = \frac{X[0] + X[1] + \cdots + X[i]}{i+1}$$

PrefixMedias1(X, n)

INPUT: $\mathbf{X}$: array de enteros
OUTPUT: $\mathbf{A}$: array de medias prefijas: A_i

a) $A \leftarrow$ Incializar (array de n double)
b) FOR $i := 0$ to $n-1$
   1) $s \leftarrow X[0]$
   2) FOR $j := 1$ to i
      $s \leftarrow s + X[j]$
   3) $A[i] \leftarrow s/(i+1)$
c) Devolver A

Solución:

- línea a: Incializa un array de n elementos: $\mathcal{O}(n)$
- línea c: operaciones primitivas de tiempo constante: $\mathcal{O}(1)$
- línea b: Bucles for anidados: $\mathcal{O}(n^2)$. Lo demostramos:
  - Las operaciones dentro del bucle externo (líneas 1 y 3) son de tiempo constante, su tiempo lo acotamos por c.
  - Las operaciones dentro del bucle interno son de tiempo constante, su tiempo lo acotamos por d.

  Entonces:

$$T_{bucle} \leq \sum_{i=0}^{n-1} \left(c + \sum_{j=1}^{i} d \right)$$

  Resolvemos los sumatorios desde el más interno al más externo:

  - El más interno: $\sum_{j=1}^{i} d \quad = i.d$
  - y sustituimos:

$$T_{bucle} \leq \sum_{i=0}^{n-1} (c + i.d)$$

  - Para calcular este sumatorio, lo desdoblamos en dos:
  - en el primero sumamos valores ctes, y en el segundo sumamos en i :

$$T_{bucle} \leq \sum_{i=0}^{n-1} c + \sum_{i=0}^{n-1} i.d$$

- Obtenemos de forma independiente la suma de cada uno y:

$$T_{bucle} \leq n.c + d\sum_{i=0}^{n-1} i$$

- Como esta suma vale

$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

- Entonces:

$$T_{bucle} \leq n.c + d\left(\frac{n(n-1)}{2}\right)$$

$$T_{bucle} \in \mathcal{O}(n^2)$$

- El orden del algoritmo es del orden del máximo de esas funciones, $max\{1, n, n^2\}$ por tanto:

$$T(n) \in \mathcal{O}(n^2)$$

4. Indicar el orden de complejidad para esta nueva versión del algoritmo de cálculo de medias, en la que se elimina el bucle interno.

PrefixMedias2(X, n)

INPUT: **X**: array de enteros
OUTPUT: **A**: array de medias prefijas: A_i

a) $A \leftarrow$ Inicializar (array de n double)
b) $s \leftarrow 0$
c) FOR $i := 0$ to $n-1$
   1) $s \leftarrow s + X[i]$
   2) $A[i] \leftarrow s/(i+1)$
d) Devolver A

Solución:

- línea a: Incializa un array de n elementos: $\mathcal{O}(n)$
- líneas b,d: operaciones primitivas de tiempo constante: $\mathcal{O}(1)$
- línea c: Bucle for : $\mathcal{O}(n)$. Lo demostramos:
  - Las operaciones dentro del bucle son de tiempo constante, lo acotamos por c.

  Entonces:

$$T_{bucle} \leq \sum_{i=0}^{n-1} c$$

  Resolvemos:

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

$$T_{bucle} \in \mathcal{O}(n)$$

- El orden del algoritmo es del orden del máximo de esas funciones, $max\{1, n, n\}$ por tanto:

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

- Mejora la complejidad respecto a la versión anterior evitando recalcular valores (para eso se usaba el bucle interno).

5. Indicar el orden de complejidad de este algoritmo en el peor caso:

Calcular(X, n)

INPUT: **X**: array de enteros
OUTPUT: **A**: array de valores calculados: A_i

a) $A \leftarrow$ Inicializar (array de n double)
b) FOR $i := 0$ to $n-1$
    1) $s \leftarrow X[0]$
    2) FOR $j := 1$ to 8
        $s \leftarrow s + X[j]$
    3) $A[i] \leftarrow s/(i+1)$
c) Devolver A

Solución:

- línea a: Incializa un array de n elementos: $\mathcal{O}(n)$
- línea c: operaciones primitivas de tiempo constante: $\mathcal{O}(1)$
- línea b: Bucles for anidados : $\mathcal{O}(n)$. Lo demostramos:
  - Las operaciones dentro del bucle externo (líneas 1 y 3) son de tiempo constante, su tiempo lo acotamos por c.
  - Las operaciones dentro del bucle interno son de tiempo constante, su tiempo lo acotamos por d.

  Entonces:
  $$T_{bucle} \leq \sum_{i=0}^{n-1} \left(c + \sum_{j=1}^{8} d \right)$$
  Resolvemos los sumatorios desde el más interno al más externo:
  - El más interno: $\sum_{j=1}^{8} d \quad = 8.d$
  - y sustituimos:
  $$T_{bucle} \leq \sum_{i=0}^{n-1} (c + 8.d)$$
  - Solo se suman valores constantes, por tanto:
  $$T_{bucle} \leq n.(c + 8.d)$$
  - Entonces:
  $$T_{bucle} \in \mathcal{O}(n)$$
- El orden del algoritmo es del orden del máximo de esas funciones, $max\{1, n, n\}$ por tanto:
$$T(n) \in \mathcal{O}(n)$$

6. Calcular el número de instrucciones que se realizan en este código en el caso peor:

```
1      int i,j;
2      for( i=0; i<n; i++)
3         for( j=i+1; j<n; j++)
4            if( a[i] + a[j] == 0) contador++;
```

Solución:

- líneas 2,3,4: Bucles for anidados: $\mathcal{O}(n^2)$. Lo demostramos:
  - Las operaciones dentro del bucle interno son de tiempo constante, su tiempo lo acotamos por d.

  Entonces:

  $$T_{bucle} \leq \sum_{i=0}^{n-1}\left(\sum_{j=i+1}^{n-1} d\right)$$

  Resolvemos los sumatorios desde el más interno al más externo:
  - El más interno:

  $$\sum_{j=i+1}^{n-1} d \quad = [(n-1)-(i+1)+1]\, d = (n-1-i)d$$

  - y sustituimos:

  $$T_{bucle} \leq \sum_{i=0}^{n-1}(n-1-i)d$$

  - Para calcular este sumatorio, lo desdoblamos en dos:
  - en el primero sumamos valores ctes, y en el segundo sumamos en i :

  $$T_{bucle} \leq \sum_{i=0}^{n-1}(n-1)d - \sum_{i=0}^{n-1} i.d =$$

  - Obtenemos de forma independiente la suma de cada uno y:

  $$T_{bucle} \leq n.(n-1)d - d\sum_{i=0}^{n-1} i$$

  - Como esta suma vale

  $$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

  - Entonces:

  $$T_{bucle} \leq n.(n-1)d - d\left(\frac{n(n-1)}{2}\right)$$

  $$T_{bucle} \in \mathcal{O}(n^2)$$

- El orden de este código es:

$$T(n) \in \mathcal{O}(n^2)$$

7. Calcular orden de complejidad para este algoritmo:

ParImpar(n: int)

```
INPUT: n: número entero
1. x ← 0
2. y ← 0
3. FOR i := 1 to n
4.   IF esPar(i)
5.     FOR j := i to n
6.         x ← x + 1
7.   ELSE
8.     FOR j := 1 to i − 1
9.         y ← y + 1
End
```

Solución:

- líneas 1,2 : operaciones primitivas de tiempo constante: $\mathcal{O}(1)$
- línea 3: Bucles for anidados, con **sentencia condicional**: $\mathcal{O}(n^2)$.
  Lo demostramos:
  Al haber una sentencia condicional dentro del bucle de la línea 3 el número de operaciones hay que calcularlo en base a dicha sentencia, y será el coste de la parte en la que más operaciones se realicen. Entonces calcularemos dos costes:
  1) Si se verifica la condición de la línea 4, i **es par** , se ejecutaría el bucle de la línea 5 (y las instrucciones de tiempo cte de línea 6), y el coste lo calculamos como:
  $$T_{buclePar} \leq \sum_{i=1}^{n}\left(\sum_{j=i}^{n} c\right)$$
  2) Si la condición de la línea 4 es Falsa, i **es impar** , se ejecutaría el bucle de la línea 8 (y las instrucciones de tiempo cte de línea 9), y el coste lo calculamos como:
  $$T_{bucleImpar} \leq \sum_{i=1}^{n}\left(\sum_{j=1}^{i-1} d\right)$$
  - Calculamos el coste del bucle para el caso 1) cuando i es **par**:
    - El sumatorio más interno: $\sum_{j=i}^{n} c \quad = (n-i+1)c$
    - y sustituimos:
    $$T_{buclePar} \leq \sum_{i=1}^{n}((n-i+1)c)$$
    - Para calcular este sumatorio, lo desdoblamos en dos (uno suma solo constantes y el otro suma valores de i):
    $$T_{buclePar} \leq c\sum_{i=1}^{n}(n+1) - c\sum_{i=1}^{n} i$$

- Obtenemos de forma independiente cada sumando y como

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

- Entonces:

$$T_{buclePar} \leq c.n(n+1) - c\frac{n(n+1)}{2}$$

$$T_{buclePar} \in \mathcal{O}(n^2)$$

- Calculamos el coste del bucle para el caso 2) cuando i es **impar**:
  - El sumatorio más interno: $\sum_{j=1}^{i-1} d \quad = (i-1)d$
  - y sustituimos:

$$T_{bucleImpar} \leq \sum_{i=1}^{n} ((i-1)d)$$

  - Entonces:

$$T_{bucleImpar} \leq d\sum_{i=1}^{n}(i-1)$$

  - Como la suma de esta sucesión es

$$\sum_{i=1}^{n}(i-1) = \frac{n(n-1)}{2}$$

  - Entonces:

$$T_{bucleImpar} \leq d\frac{n(n-1)}{2}$$

$$T_{bucleImpar} \in \mathcal{O}(n^2)$$

Por tanto, vemos que tanto si i es **par** como si i es **impar**, el coste del bucle es n^2:

$$T_{bucle} = max\{T_{buclePar}, T_{bucleImpar}\} = max\{n^2, n^2\}$$

$$T_{bucle} \in \mathcal{O}(n^2)$$

- Y finalmente el orden del algoritmo es del orden del máximo de esas funciones, $max\{1, n^2\}$ por tanto:

$$T(n) \in \mathcal{O}(n^2)$$

8. Calcular $\mathcal{O}$ y Ω para este algoritmo.

Algoritmo LongMaxSecuencia Dado un array X, calcula la máxima longitud de una subsecuencia ordenada (ascendente) dentro del array.

LongMaxSecuencia(X, n)

INPUT: $\mathbf{X}$: array de longitud n

```
1. max ← 0
2. FOR i := 1 to n
3.    cont ← 1
4.    j ← i + 1
5.    WHILE X[i] ≤ X[j] AND j ≤ n
6.       j := j + 1
7.       cont := cont + 1
8.    IF cont > max
9.       max ← cont
End
```

Sugerencia: obtenerlos calculando el **número de veces que se ejecuta el bucle WHILE** en el caso peor (para la notación $\mathcal{O}$) y en el caso mejor (para la notación Ω).

Solución:

NOTACIÓN $\mathcal{O}$

- línea 1 : operaciones primitivas de tiempo constante: $\mathcal{O}(1)$
- línea 2: **bucle for + bucle While anidados** : $\mathcal{O}(n^2)$.
  Lo demostramos:
  Para estudiar el coste del bucle While de la línea 5 tenemos que tener en cuenta, que **en el caso peor, se realizará siempre**, y por tanto su coste podemos deducirlo fácilmente teniendo en cuenta que:
  - El número de veces que se ejecuta el bucle depende del valor de j, que se inicializa a $i+1$ en la línea 4 y como máximo j valdrá n, que es el número de elementos del array. Es decir, j varía desde $i+1$ hasta n, y podemos considerar el bucle While de modo equivalente a un bucle for.
  - Dentro del bucle While, en las líneas 6 y 7, se realizan operaciones de tiempo cte, que acotaremos por c
  - Por tanto el coste del bucle While lo podemos calcular así:

$$T_{bucleWhile} \leq \sum_{j=i+1}^{n} c$$

  - Ya podemos calcular el coste del bucle for, a partir del coste del bucle While en el caso peor, y el coste constante d de las líneas 8 y 9:

$$T_{bucle} \leq \sum_{i=1}^{n} d + \left(\sum_{j=i+1}^{n} c \right)$$

- Calculamos el coste del bucle igual que en ejercicios anteriores:
  - El sumatorio más interno: $\sum_{j=i+1}^{n} c \quad = (n-i)c$
  - y sustituimos:

$$T_{bucle} \leq \sum_{i=1}^{n} d + ((n-i)c)$$

  - Desdoblamos en dos sumatorios (uno suma solo constantes y el otro suma valores de i):

$$T_{bucle} \leq \sum_{i=1}^{n} (d+cn) - c\sum_{i=1}^{n} i$$

  - Obtenemos de forma independiente cada sumando y como

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

  - Entonces:

$$T_{bucle} \leq n(d+cn) - c\frac{n(n+1)}{2}$$

$$T_{bucle} \in \mathcal{O}(n^2)$$

- Y finalmente el orden del algoritmo en el caso peor es del orden del máximo de esas funciones, $max\{1, n^2\}$ por tanto:

$$T(n) \in \mathcal{O}(n^2)$$

NOTACIÓN Ω

- línea 1 : operaciones primitivas de tiempo constante: $\Omega(1)$
- línea 2: **bucle for + bucle While anidados** : $\Omega(n)$.
  Lo demostramos:

  Nuevamente volvemos a estudiar el coste del bucle While de la línea 5 y teniendo en cuenta que analizamos **en el caso mejor, la condición se evaluará a falso y no se ejecutará nunca**, y por tanto el coste de la operación en la línea 5 será solo el de comprobar la condición, que es de tiempo constante, y nunca se ejecutarán las instrucciones de las líneas 6 y 7. Tenemos entonces dentro del bucle for:
  - líneas 3,4: operaciones de tiempo cte.
  - línea 5: bucle While que NO se ejecuta. Comprobar la condición es operación de tiempo cte. No se ejecutan líneas 6 y 7.
  - líneas 8,9: operaciones de tiempo cte.
  - Es decir, dentro del for solo hay operaciones de tiempo constante, que llamaremos c , y por tanto el coste sería:

$$T_{bucle} \geq \sum_{i=1}^{n} c$$

- Ya podemos calcular el coste del bucle for, en el caso mejor:

$$T_{bucle} \geq c.n$$
$$T_{bucle} \in \Omega(n)$$

- Y finalmente el orden del algoritmo en el caso mejor es del orden del máximo de esas funciones, $max\{1, n\}$ por tanto:

$$T(n) \in \Omega(n)$$

El tiempo de ejecución de este algoritmo está en $\Omega(n)$ y en $\mathcal{O}(n^2)$.

9. (Examen Final Junio 2014). **Obtener y demostrar** el orden de complejidad de este algoritmo en el peor caso.

BuscaElementos(M, array, n)

INPUT: $\mathbf{M}$: matriz de enteros de dimensión $n \times n$,
array: array de enteros de dimensión n
OUTPUT:
busca cada elemento de la matriz en el array y devuelve el número de coincidencias.

a) $suma \leftarrow 0$;
b) **FOR** $i \leftarrow 1$ **to** n **do**
c) **FOR** $j \leftarrow 1$ **to** n **do**
d) $valor \leftarrow$ BusquedaBinaria(array, n , M[i][j]) ;
e) IF $valor \geq 0$ THEN $suma \leftarrow suma + 1$ //está
f) Devolver $suma$;

Solución:

- líneas a,f : operaciones primitivas de tiempo constante: $\mathcal{O}(1)$
- línea b: Bucles for anidados, con **llamada a un método dentro del bucle interno**: $\mathcal{O}(n^2 \log n)$.
  Lo demostramos:
  Dentro del **bucle interno** se está invocando a otro método:
  la Búsqueda Binaria, y por tanto en el coste del bucle interno hay que tener en cuenta el coste de ejecutarse este método que es $\mathcal{O}(\log n)$:
  - El coste de la línea d) es : $\mathcal{O}(\log n)$
  - El coste de la línea e) es constante: c
  - Por tanto el coste del bucle interno es : $c + \mathcal{O}(\log n) \in \mathcal{O}(\log n)$

  Calculamos el coste del bucle como:

$$T_{bucle} \leq \sum_{i=1}^{n} \left(\sum_{i=1}^{n} \mathcal{O}(\log n) \right)$$

  Resolviendo, desde el sumatorio más interno:

$$T_{bucle} \leq \sum_{i=1}^{n} (n.\mathcal{O}(\log n))$$

  y por tanto:

$$T_{bucle} \leq n\,(n.\mathcal{O}(\log n)) = \mathcal{O}(n^2).\mathcal{O}(\log n)$$

$$T_{bucle} \in \mathcal{O}(n^2 \log n)$$

- Y finalmente el orden del algoritmo es del orden del máximo de esas funciones, $max\{1, n^2 \log n\}$ por tanto:

$$T(n) \in \mathcal{O}(n^2 \log n)$$

10. (Examen Final Junio 2013). **Obtener y demostrar**, usando la notación asintótica, el orden $\mathcal{O}$ de este algoritmo.

funcion Productos($A[1..n]$)
$//A$ es una array de n elementos

a) $prod \leftarrow 1;$
b) **FOR** $i \leftarrow 1$ **to** n **do**
c) **FOR** $j \leftarrow i+1$ **to** n **do**
d) **FOR** $k \leftarrow 1$ **to** 5 **do**
$prod \leftarrow A[i] * A[j] * prod;$
e) Devolver $prod;$

Solución:

- líneas a) y e): operaciones primitivas de tiempo constante: $\mathcal{O}(1)$
- líneas b), c) y d): Bucles for anidados: $\mathcal{O}(n^2)$. Lo demostramos:
  - Las operaciones dentro del tercer bucle son de tiempo constante, su tiempo lo acotamos por d.

  Entonces:

  $$T_{bucle} \leq \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\sum_{k=1}^{5} d \right)$$

  Resolvemos los sumatorios desde el más interno al más externo:

  - El más interno: $\sum_{k=1}^{5} d \quad = 5.d$
  - y sustituimos:

  $$T_{bucle} \leq \sum_{i=1}^{n} \sum_{j=i+1}^{n} (5.d)$$

  - Calculamos el segundo sumatorio:

  $$\sum_{j=i+1}^{n} (5.d) = (n-i)(5.d)$$

  - y sustituimos

  $$T_{bucle} \leq \sum_{i=1}^{n} (n-i)(5.d)$$

  - Simplificamos:

  $$T_{bucle} \leq 5d \sum_{i=1}^{n} (n-i)$$

  - Como la suma:

  $$\sum_{i=1}^{n} (n-i) = \frac{n(n-1)}{2}$$

  - Entonces:

  $$T_{bucle} \leq 5d.\frac{n(n-1)}{2} \in \mathcal{O}(n^2)$$

- El orden del algoritmo es del orden del máximo de esas funciones, por tanto:

$$T(n) \in \mathcal{O}(n^2)$$