

Tres problemas con soluciones sobre análisis de orden de tiempo ejecución para el peor caso

Problema 1

Considere la siguiente función en C++, definido sobre un arreglo de enteros de tamaño n:

```
bool F (int * A, unsigned int n)
{
    bool res = true;
    int i, j;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            if (i<j)
                res = res && (A[j]<A[i]);
    return res;
}
```

- a) ¿Qué calcula/retorna (conceptualmente) la función F, dado un arreglo de enteros de tamaño n?
- b) Calcule el orden (O) de tiempo de ejecución para el peor caso de la función F. El cálculo puede ser realizado esquemáticamente sobre el código mismo.
- c) El problema que resuelve F, ¿podría resolverse en un menor orden de tiempo de ejecución en el peor caso?. Justifique en caso negativo y en caso afirmativo escriba la función, indicando el orden (O).

Problema 2

Considere la siguiente función en C++, definida sobre un arreglo de enteros de tamaño n:

```
bool F (int * A, unsigned int n)
{
    bool res = true;
    int i, j;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            if (i!=j)
                res = res && (A[i]!=A[j]);
    return res;
}
```

- a) ¿Qué calcula/retorna (conceptualmente) la función **F**, dado un arreglo de enteros de tamaño n?
- b) Calcule el orden (O) de tiempo de ejecución para el peor caso de la función F. El cálculo puede ser realizado esquemáticamente sobre el código mismo.
- c) Si se sabe que el arreglo A sólo puede contener valores enteros en el rango [0 : n-1], el problema que resuelve F ¿podría resolverse en un menor orden de tiempo de ejecución en el peor caso?. Justifique en caso negativo y en caso afirmativo escriba la función, indicando el orden (O).

Problema 3

Considere la siguiente función en C++, definida sobre dos arreglos de enteros de tamaño n :

```
bool F (int * A, int * B, int n)
{
    bool res = true;
    int i, j;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            if (i+j == n-1)
                res = res && (A[i]==B[j]);
    return res;
}
```

- ¿Qué calcula/retorna (conceptualmente) la función F , dado dos arreglos de enteros de tamaño n ?
- Calcule el orden (O) de tiempo de ejecución para el peor caso de la función F . El cálculo puede ser realizado esquemáticamente sobre el código mismo.
- El problema que resuelve F , ¿podría resolverse en un menor orden de tiempo de ejecución en el peor caso?. Justifique en caso negativo y en caso afirmativo escriba la función, indicando el orden.

Solución del Problema 1:

Considere la siguiente función en C++, definido sobre un arreglo de enteros de tamaño n :

```

bool F (int * A, unsigned int n)
{
    bool res = true;           O(1)
    int i, j;                  O(1)
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            if (i<j)           O(1)
                res = res && (A[j]<A[i]);  O(1)
    return res;               O(1)
}

```

$\left. \begin{array}{l} O(1) \\ O(1) \\ \left. \left. \begin{array}{l} O(1) \\ O(1) \end{array} \right\} O(n) \right\} O(n^2) \\ O(1) \end{array} \right\} O(n^2)$

- a) ¿Qué calcula/retorna (conceptualmente) la función F, dado un arreglo de enteros de tamaño n ?
Retorna true si y sólo si el arreglo está ordenado estrictamente (sin elementos repetidos) de mayor a menor.
- b) Calcule el orden (O) de tiempo de ejecución para el peor caso de la función F. El cálculo puede ser realizado esquemáticamente sobre el código mismo.
Es $O(n^2)$. Resuelto sobre el código de la función F.
- c) El problema que resuelve F, ¿podría resolverse en un menor orden de tiempo de ejecución en el peor caso?. Justifique en caso negativo y en caso afirmativo escriba la función, indicando el orden (O).
SI, en $O(n)$ peor caso.

```

bool F (int * A, unsigned int n)
{
    int i;                     O(1)
    for (i=1; i<n && A[i-1]>A[i]; i++);  O(n)
    return (i==n);            O(1)
}

```

$\left. \begin{array}{l} O(1) \\ O(n) \\ O(1) \end{array} \right\} O(n)$

Solución del Problema 2:

Considere la siguiente función en C++, definida sobre un arreglo de enteros de tamaño n :

```
bool F (int * A, unsigned int n)
{
    bool res = true;           O(1)
    int i, j;                 O(1)
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            if (i!=j)
                res = res && (A[i]!=A[j]);
    return res;
}
```

Diagrama de complejidad:

- $O(1)$ para `bool res = true;`
- $O(1)$ para `int i, j;`
- $O(n)$ para el bucle `for (i=0; i<n; i++)`
- $O(n)$ para el bucle `for (j=0; j<n; j++)`
- $O(1)$ para `if (i!=j)`
- $O(1)$ para `res = res && (A[i]!=A[j]);`
- $O(1)$ para `return res;`

El producto de los bucles `for` resulta en $O(n^2)$, lo que define la complejidad total de la función.

- a) ¿Qué calcula/retorna (conceptualmente) la función **F**, dado un arreglo de enteros de tamaño n ?
Retorna true si y sólo todos los elementos del arreglo son distintos.
- b) Calcule el orden (O) de tiempo de ejecución para el peor caso de la función **F**. El cálculo puede ser realizado esquemáticamente sobre el código mismo.
Es $O(n^2)$. Resuelto sobre el código de la función **F.**
- c) Si se sabe que el arreglo **A** sólo puede contener valores enteros en el rango $[0 : n-1]$, el problema que resuelve **F** ¿podría resolverse en un menor orden de tiempo de ejecución en el peor caso?. Justifique en caso negativo y en caso afirmativo escriba la función, indicando el orden (O).
SI, en $O(n)$ peor caso.

```
bool F (int * A, unsigned int n)
{
    int i;
    int * Pertenece = new int[n];

    for (i=0; i<n; i++) Pertenece[i] = 0;
    for (i=0; i<n && Pertenece[A[i]]==0; i++)
        Pertenece[A[i]] = 1;
    delete [] Pertenece;
    return (i==n);
}
```

Diagrama de complejidad:

- $O(1)$ para `int i;`
- $O(1)$ para `int * Pertenece = new int[n];`
- $O(n)$ para el bucle `for (i=0; i<n; i++) Pertenece[i] = 0;`
- $O(n)$ para el bucle `for (i=0; i<n && Pertenece[A[i]]==0; i++)`
- $O(n)$ para `Pertenece[A[i]] = 1;`
- $O(1)$ para `delete [] Pertenece;`
- $O(1)$ para `return (i==n);`

El producto de los bucles resulta en $O(n)$, lo que define la complejidad total de la función.

Solución del Problema 3:

Considere la siguiente función en C++, definida sobre dos arreglos de enteros de tamaño n:

```

bool F (int * A, int * B, int n)
(1) {  bool res = true;           | O(1)
(2)    int i, j;                 | O(1)
(3)    for (i=0; i<n; i++)
(4)        for (j=0; j<n; j++)
(5)            if (i+j == n-1)   | O(1)  n*O(1)=O(n)
(6)                res = res&&(A[i]==B[j]); | O(1)
(7)    return res;              | O(1)
}

```

$n * O(n) = \underline{O(n^2)}$

a) ¿Qué calcula/retorna (conceptualmente) la función F, dado dos arreglos de enteros de tamaño n?

La función verifica que los arreglos sean uno la inversión del otro.

Si A es:

2	3	7	9	11
---	---	---	---	----

y B es:

11	9	7	3	2
----	---	---	---	---

retorna true; en cualquier otro caso retorna false

b) Calcule el orden (O) de tiempo de ejecución para el peor caso de la función F. El cálculo puede ser realizado esquemáticamente sobre el código mismo . $O(n^2)$

c) El problema que resuelve F, ¿podría resolverse en un menor orden de tiempo de ejecución en el peor caso?. Justifique en caso negativo y en caso afirmativo escriba la función, indicando el orden.

```

bool F (int * A, int * B, int n)
(1) {  bool res = true;           | O(1)
(2)    int i;                     | O(1)
(3)    for (i=0; i<n; i++)
(4)        res = res && (A[i]==B[n-1-i]); | O(1)
(5)    return res;              | O(1)
(6) }

```

$n * O(1) = O(n)$
 $O(\max(1, 1, n, 1)) = \underline{O(n)}$