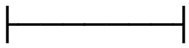
EJEMPLOS DE EFICIENCIA EN BUCLES Y FUNCIONES

Usando la notación O(·), dar el peor caso de tiempo de ejecución de las siguientes funciones, expresándolo en función de n:

```
1:    void ejemplo1(int n)
2:    {
3:       int i, j, k;
4:
5:       for (i = 0; i < n; i++)
6:          for (j = 0; j < n; j++)
7:          {
8:             C[i][j] = 0;
9:             for (k = 0; k < n; k++)
10:              C[i][j] = += A[j][k] * B[k][j];
11:         }
12:    }
```

Se puede considerar la instrucción 10 como constante, que más la condición, inicialización e incremento del bucle en 9 daría O(1) para el cuerpo de dicho bucle. Puesto que el bucle se repite n veces, tendríamos que el bucle tiene un orden de O(n). La instrucción en 8 también es O(1), que junto con la condición, inicialización e incremento del bucle en 6 y más el bucle en 9 daría O(1) + O(n) = O(n) para el cuerpo del bucle en 6. Puesto que el bucle en 6 se repite n veces, tendríamos $O(n^2)$. Y puesto que el bucle en 5 también se repite n veces, tenemos finalmente un orden de $O(n^3)$.



```
1:    void ejemplo2(int n)
2:    {
3:       int i, j, k;
4:
5:       for (i = 1; i < n; i++)
6:          for (j = i+1; j <= n; j++)
7:             for (k = 1; k <= j; k++)
8:                Global += k*i;
9:    }
```

Lo que nos interesa en este caso es ver cuántas veces se repite la instrucción en 8. Se puede resolver de varias formas. Vamos a ver 2:

1. Si nos fijamos en las cabeceras de los bucles, podemos ver que teniendo en cuenta los tres bucles (primero los dos internos y luego el más externo), la instrucción en 8 se repite el siguiente número de veces:

$$\sum_{i=1}^{n-1}\sum_{j=1}^{n-i} j+i$$

Teniendo en cuenta la sumatoria interna y separando en $\sum_{j=1}^{n-i} j+\sum_{j=1}^{n-i} i$, tendríamos:

$$\sum_{j=1}^{n-i} j=\frac{(n-i)(n-i+1)}{2}=\frac{n^2-2ni+n+i^2-i}{2},\ \sum_{j=1}^{n-i} i=(n-i)i=ni-i^2, con\ lo\ que$$

$$\sum_{j=1}^{n-i} j+\sum_{j=1}^{n-i} i=\frac{n^2-2ni+n+i^2-i+2ni-2i^2}{2}=\frac{n^2+n-i-i^2}{2}$$

Finalmente tendríamos que:

$$\sum_{i=1}^{n-1}\frac{n^2+n-i-i^2}{2}=\frac{1}{2}\left(\sum_{i=1}^{n-1}n^2+\sum_{i=1}^{n-1}n-\sum_{i=1}^{n-1}i-\sum_{i=1}^{n-1}i^2\right)=\frac{1}{2}\left((n^3-n^2)+(n^2-n)-(\frac{n^2-n}{2})-(\frac{2n^3-3n^2+n}{6})\right)=$$

$$=\frac{1}{2}\left(\frac{6n^3-6n^2+6n^2-6n-3n^2+3n-2n^3+3n^2-n}{6}\right)=\frac{1}{2}\left(\frac{4n^3-4n}{6}\right)=\frac{n^3-n}{3}$$

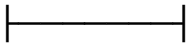
Ejemplo: Para *n=4* se puede comprobar que la instrucción en 8 se ejecuta 20 veces. Teniendo en cuenta la fórmula saldría también *(64-4)/3 = 20*.

2. Si tenemos en cuenta un ejemplo (n=4) y vemos el número de veces que se ejecuta la instrucción en 8 se puede obtener la siguiente progresión:

```
2 + 3 + 4 +
    3 + 4 +
      + 4
```

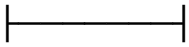
Se puede ver que la progresión es la siguiente: 1*2 + 2*3 + 3*4. Teniendo en cuenta un n genérico tendríamos: 1*2 + 2*3 + … + *(n*-1*)*n*. Y teniendo en cuenta que en el primer bucle la variable i toma valores desde 1 hasta n-1, tendríamos la siguiente sumatoria:

$$\sum_{i=1}^{n-1}i(i+1)=\sum_{i=1}^{n-1}i^2+\sum_{i=1}^{n-1}i=\left(\frac{2n^3-3n^2+n}{6}\right)+\left(\frac{n^2-n}{2}\right)=\frac{2n^3-3n^2+n+3n^2-3n}{6}=\frac{n^3-n}{3}$$



```
1:    void ejemplo3(int n)
2:    {
3:       int i, j, x, y;
4:
5:       for (i = 1; i <= n; i++)
6:          if (i % 2 == 1)
7:          {
8:             for (j = i; j <= n; j++)
9:                x++;
10:            for (j = 0; j < i; j++)
11:               y++;
12:         }
13:   }
```

En este caso, se puede ver que la instrucción en 9 se ejecuta *n-i+1* veces. En el caso de la instrucción en 11, se puede ver que se ejecuta i veces. En total, para el trozo de código entre las líneas 7 y 12 tenemos *n-i+1+i = n+1* instrucciones, por lo que es de orden O(n). Teniendo en cuenta los valores de i para el bucle en 5 y la condición en 6, podemos ver que el cuerpo de la instrucción condicional se ejecuta *n/2* veces (división entera). Por lo que, si el cuerpo de la condicional es de orden O(n), tendríamos $n^2/2$ y finalmente el orden sería $O(n^2)$.

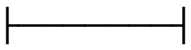


```
1:    int ejemplo4(int n)
2:    {
3:       if (n <= 1)
4:          return 1;
5:       else
6:          return (ejemplo4(n - 1) + ejemplo4(n - 1));
7:    }
```

La eficiencia de una función recursiva se obtiene planteando la función de tiempo de manera recursiva y resolviendo dicha recurrencia. En estos casos, tenemos que ver qué se ejecuta antes, en medio y después de las llamadas recursivas para sumárselo a los tiempos de las llamadas recursivas. Tenemos una comparación en 3 más dos restas y una suma en 6. El orden de estas instrucciones es O(1). De esta manera y teniendo en cuenta las llamadas recursivas se nos plantea la siguiente recurrencia:

$$T(n) = 2T(n-1) + 1,$$

cuya solución es $T(n) = c_1 2^n + c_2 1^n$. Teniendo en cuenta que la función de tiempo tiene que ser positiva, el orden sería $O(2^n)$.



```
1:    int ejemplo5(int n)
2:    {
3:       if (n == 1)
4:          return n;
5:       else
6:          return (ejemplo5(n/2) + 1);
7:    }
```

Como en el caso anterior, la eficiencia de una función recursiva se obtiene planteando la función de tiempo de manera recursiva y resolviendo dicha recurrencia. Viendo qué se ejecuta antes, en medio y después de la llamada recursiva tenemos, una comparación en 3 más una división y una suma en 6. El orden de estas instrucciones es O(1). De esta manera y teniendo en cuenta la llamada recursiva se nos plantea la siguiente recurrencia:

$$T(n) = T(\frac{n}{2}) + 1,$$

cuya solución es $T(n) = c_1 n^0 + c_2 \lg(n) n^0$. Teniendo en cuenta que la función de tiempo tiene que ser positiva y creciente, el orden sería $O(\lg(n))$.