Ejercicio 01

1. Calcule el T(n) del siguiente código:

```
public static void tiempo (int n) {
  int i, j;
  for (i = 1; i <= n/2; i++) {
    for (j = n; j >= i; j--) {
      algo_de_O(1);
    }
  }
}
```

2. Calcule el O(n) por definición.

Resolución

Cálculo de T(n)

En este caso tenemos un algoritmo iterativo con dos *for* anidados sujetos a un parámetro de entrada n. Lo primero que vamos a hacer, para ser organizados en nuestro planteo, es entender el algoritmo como si todo lo que estuviera dentro de las llaves de cada estructura de control (en este caso, los *for*) no nos importara, comenzando por las de mayor jerarquía. Entonces, nos quedaría algo así:

```
public static void tiempo (int n) {
  int i, j;
  for (i = 1; i <= n/2; i++) {
    algo_que_todavia_no_nos_importa();
  }
}
```

Como podemos observar, tenemos un *for* recorrido con un índice i que va desde 1 hasta $n/2$. Lo que nos interesa es cuántas veces vamos a ejecutar el código que tenemos dentro de ese *for*. Para obtener este dato debemos averiguar la cantidad de elementos que hay entre 1 y $n/2$ inclusive. Recordemos como calcular la cantidad de elementos en un intervalo:

$$\#elementos = Mayor\_elemento - Menor\_elemento + 1$$

En este caso, nuestro primer elemento este el numero 1, y el último la mitad del parámetro n, es decir, $n/2$. Por lo tanto:

$$\#elementos = n/2 - 1 + 1 = n/2$$

Nuestra primera sumatoria va a tener n/2 elementos, por lo tanto, el índice de la misma irá desde 1 hasta n/2 inclusive. En este caso particular este análisis puede resultar trivial, pero son importantes estas aclaraciones para saber cómo encarar casos más complicados donde la cantidad de veces que se ejecuta el código o el rango de valores que puede adoptar el índice no están tan claros. Volviendo al ejercicio en sí, nuestro T(n), por ahora, quedaría asi:

$$T(n) = \sum_{i=1}^{n/2} algo\_que\_todavia\_no\_nos\_importa()$$

Comencemos a trabajar ahora con el *for* que es parte de ese código que, hasta este momento, no nos importaba analizar. Tomaremos al código como si fuera el siguiente:

```
public static void tiempo (int n) {
  int i, j;
  for (i = 1; i <= n/2; i++) {
    for (j = n; j >= i; j--) {
      otro_algo_que_todavia_no_nos_importa();
    }
  }
}
```

Tenemos un índice j que inicialmente adopta el valor de n y disminuye hasta alcanzar el valor de i inclusive. Por lo tanto, volvemos a utilizar nuestra ecuación para la cantidad de elementos del intervalo:

$$\#elementos = n - i + 1$$

Observamos que la cantidad de veces que ejecutamos el algoritmo es la misma independientemente de cómo se mueva j, es decir, sea desde n hasta i o a la inversa. Como a nosotros solo nos importa la cantidad de veces que se ejecuta el código, nos es indistinto plantearlo de una u otra forma, por lo que vamos a adoptar la que más nos convenga, en este caso, movernos desde i hasta n, a la inversa. De esta forma, el planteo de T(n) adicionando este último *for* nos quedaría así:

$$T(n) = \sum_{i=1}^{n/2} \sum_{j=i}^{n} otro\_algo\_que\_todavia\_no\_nos\_importa()$$

Por último, nos informan que la parte del código que nos falta analizar es de O(1), lo que significa que lo trabajamos como una constante. Nuestro T(n) queda así:

$$T(n) = \sum_{i=1}^{n/2} \sum_{j=i}^{n} 1$$

Nos resta ahora operar algebráicamente esta expresión para que T(n) quede en una forma sencilla para calcular su tiempo de ejecución. Entonces, lo primero que vamos a hacer es resolver la sumatoria con índice j que va desde i hasta n. Como dentro de la expresión no tenemos a la variable j, vamos a sumar 1 tantas veces como elementos tenga el intervalo de i hasta n, que calculamos previamente. Por lo tanto:

$$T(n) = \sum_{i=1}^{n/2} \sum_{j=i}^{n} 1$$

$$T(n) = \sum_{i=1}^{n/2} (n - i + 1)$$

Distribuimos la sumatoria en cada uno de sus términos para poder trabajar. Recordemos que la sumatoria puede distribuirse sólo con las sumas y las restas.

$$T(n) = \sum_{i=1}^{n/2} n - \sum_{i=1}^{n/2} i + \sum_{i=1}^{n/2} 1$$

Si analizamos cada sumatoria individualmente, observamos que en la primera y la tercera no tenemos al índice dentro de los valores a sumar, por lo tanto, es lo mismo que trabajar con constantes, por lo que operamos análogamente a como hicimos previamente.

$$T(n) = \left(n * \frac{n}{2}\right) - \sum_{i=1}^{n/2} i + \left(1 * \frac{n}{2}\right)$$

En este punto vale la pena hacer una aclaración: si bien es innecesario en este ejemplo encerrar entre paréntesis las expresiones que corresponden a cada sumatoria, en ejercicios más complejos algebráicamente esta costumbre puede salvarnos de errores que, aunque simples, pueden significar la desaprobación de un examen.

Siguiendo con el ejercicio, nos queda resolver la sumatoria de los *n/2* primeros números. Tengamos en mente algo importante: no importa que el límite superior de la sumatoria sea *n*, *n/2*, $\sqrt{n^3}$, sigue siendo un nÃºmero y debemos tratarlo como a cualquier otro. Por lo tanto, recordando la equivalencia de la suma de los primeros n números:

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

Apliquemos esto en nuestro ejercicio:

$$T(n) = \left(n * \frac{n}{2}\right) - \left(\frac{\left(\frac{n}{2}\right) * \left(\frac{n}{2}+1\right)}{2}\right) + \left(1 * \frac{n}{2}\right)$$

Ya tenemos todo en función de n. Ahora, vamos a acomodarlo para poder calcular posteriormente el orden de dicha función.

$$T(n) = \left(n * \frac{n}{2}\right) - \left(\frac{\left(\frac{n}{2}\right) * \left(\frac{n}{2}+1\right)}{2}\right) + (1 * \frac{n}{2})$$

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

$$T(n) = \frac{n^2}{2} - \left(\frac{n^2}{8} + \frac{n}{4}\right) + \frac{n}{2}$$

$$T(n) = \frac{n^2}{2} - \frac{n^2}{8} - \frac{n}{4} + \frac{n}{2}$$

$$T(n) = \frac{4n^2}{8} - \frac{n^2}{8} - \frac{n}{4} + \frac{2n}{4}$$

$$T(n) = \frac{3}{8}n^2 + \frac{1}{4}n$$

De esta forma, concluimos con el cálculo del T(n) del algoritmo planteado.

Cálculo de O(n) por definición

A partir del T(n) calculado en el inciso anterior vamos a averiguar el O(n) utilizando la definición. En resumidas palabras, esta definición nos dice lo siguiente:

$$T(n) \leq c * f(n)$$

La definición de Big-O nos dice que el orden de ejecución de un algoritmo es una función que, multiplicada por una constante, es mayor o igual al T(n) del algoritmo. Concretamente, es una función que sirve de cota superior al tiempo de ejecución, es decir, representa "lo peor" que puede pasar con el algoritmo al ejecutarse.

Como en este caso nuestro T(n) vamos a aplicar la definición de Big-O para polinomios, es decir, si T(n) es un polinomio de grado k, entonces $O(n^k)$. Esto podemos demostrarlo con el siguiente procedimiento.

1. Ordenamos nuestro T(n) de manera descendente tomando como criterio el grado de cada miembro.

$$T(n) = \frac{3}{8}n^2 + \frac{1}{4}n$$

2. Tomamos al término de mayor grado para tener como referencia, en este caso, n^2.

3. Separamos los términos y aplicamos la definición de Big-O vista anteriormente. Para la parte de la derecha vamos a utilizar el valor absoluto del coeficiente y el n de mayor grado que individualizamos en el paso anterior. O, más sencillo:

$$\frac{3}{8}n^2 \leq \left|\frac{3}{8}\right| n^2$$

$$\frac{1}{4}n \leq \left|\frac{1}{4}\right| n^2$$

Antes de seguir, detengámonos un instante para entender por qué hacemos esto. n es un número natural, de forma tal que el resultado de elevarlo a cualquier potencia es un número positivo. Por

otro lado, como nuestra referencia es el término de mayor grado, este número es estrictamente mayor que cualquier otro de menor grado. Para finalizar, consideramos el valor absoluto del coeficiente, lo que nos evita problemas en la ecuación por cambios de signo. Es decir, tomando el de mayor grado y el valor absoluto del coeficiente nos garantizamos que la relación **siempre** se cumple.

4. Sumamos cada término de la relación.

$$\frac{3}{8}n^2 + \frac{1}{4}n \leq \left|\frac{3}{8}\right| n^2 + \left|\frac{1}{4}\right| n^2$$

   Tengamos en cuenta que si alguno de los coeficientes es negativo quedará como una resta del lado izquierdo ya que sumar algo negativo es, en otras palabras, restar. No sucederá de la misma forma del lado derecho visto que siempre trabajamos con valores absolutos.

5. Observamos que, a la derecha, nuestra referencia es factor común en esta suma, por lo que podemos agrupar los coeficientes.

$$\frac{3}{8}n^2 + \frac{1}{4}n \leq \left(\left|\frac{3}{8}\right| + \left|\frac{1}{4}\right|\right) n^2$$
$$\frac{3}{8}n^2 + \frac{1}{4}n \leq \left(\left|\frac{3}{8}\right| + \left|\frac{2}{8}\right|\right) n^2$$
$$\underbrace{\frac{3}{8}n^2 + \frac{1}{4}n}_{\mathbf{T(n)}} \leq \underbrace{\frac{5}{8}}_{\mathbf{c}} \underbrace{n^2}_{\mathbf{f(n)}}$$

Como puede apreciarse, logramos llegar a una expresión idéntica a la definición, de donde podemos separar el f(n) y concluir que el algoritmo es de O(n^2). Fin de la demostración.

Dudas, correcciones o sugerencias: **daniffig@gmail.com**