Programación III
TEMA 8: Tiempo de Ejecución
Práctica nº 8 - A

Importante: todas las referencias a logaritmos se refieren a logaritmo en base 2, por tanto, de ser necesario utilice el cambio de base para convertir los logaritmos a base 2 (explicado en: http://es.wikipedia.org/wiki/Logaritmo#Cambio_de_base). Si quiero cambiar de base a en base b:

$\log_a(x) = \log_b(x) / \log_b(a) \Leftrightarrow a \neq 1, b \neq 1$

1. En este ejercicio puede utilizar el graficador de funciones que se encuentra online en http://fooplot.com/. Dado que el graficador solo trabaja con logaritmos en base 10 utilice el cambio de base para convertir los logaritmos en base 2. **La consigna del ejercicio es ordenar los tiempos de ejecución según sus velocidades de crecimiento**.

   a. $T_1(n) = n * log(n)$
   b. $T_2(n) = (1/3)\, n$
   c. $T_3(n) = 2n + n^2$
   d. $T_4(n) = (3/2)\, n$
   e. $T_5(n) = (log(n))^2$
   f. $T_6(n) = log(n)$
   g. $T_7(n) = n + log(n)$
   h. $T_8(n) = n^{1/2}$
   i. $T_9(n) = 3^n$

2. Sean algoritmo1 y algoritmo2, 2 implementaciones distintas de una tarea cuyo resultado es el mismo. Se sabe además el tiempo de ejecución de estos algoritmos:

   algoritmo1 tiene $T_1(n) = 10n$
   algoritmo2 tiene $T_2(n) = 4n^2$

   2.1. **Grafique** $T_1(n)$ y $T_2(n)$ en el sistema de coordenadas cartesianas
   2.2. **Justifique analíticamente** cuando recomendaría usar el algoritmo 1 y cuando usar el algoritmo 2

3. Determinar si las siguientes sentencias son verdaderas o falsas, **justificando la respuesta.** Su demostración puede hacer uso de la definición de Big-OH ó del límite matemático de funciones.

   3.1. $3^n es\ de\ O(2^n)$
   3.2. $n/log(n)\ es\ de\ O(log(n))$
   3.3. $n + log_2(n)\ es\ de\ O(n)$
   3.4.

   $$\begin{cases} 3n+17, & n < 100 \\ 317, & n >= 100 \end{cases}$$ tiene orden lineal

   3.5. $2^{2n} es\ de\ O(2^n)$

3.6. Si p(n) es un polinomio de grado k, entonces p(n) es $O(n^k)$. Para esta demostración, considere que si $p(n)=a_k n^k + ... + a_1 n + a_0$, utilizar como constante a $M = |a_k| + ... + |a_1| + |a_0|$

3.7.

$$\begin{cases} n^2, & n \leq 100 \\ n, & n > 100 \end{cases}$$ tiene orden cuadrático

3.8. 2^{n+1} *es de* $0(2^n)$

3.9. 2^{2n} *es de* $0(2^n)$

4. Para cada uno de los algoritmos presentados:

a. Expresar en función de n el tiempo de ejecución
b. Establecer el orden de dicha función usando notación big-Oh.

En el caso de ser necesario tenga presente que:

$$\sum_{i=1}^{n} i^4 = \frac{n\,(n+1)\,(6n^3+9n^2+n-1)}{30}$$

$$\sum_{i=1}^{n} i^2 = \frac{n\,(n+1)\,(2n+1)}{6}$$

4.1.

```
public static void uno (int n)  {
            int  i, j, k ;
            int [] [] a, b, c;
            a = new int [n] [n];
            b = new int [n] [n];
            c = new int [n] [n];
            for ( i=1; i<=n-1; i++) {
                  for ( j=i+1; j<=n; j++) {
                              for ( k=1; k<=j; k++)  {
                              c[i][j] = c[i][j]+ a[i][j]*b[i][j];
                              }
                  }
            }
  }
```

4.2.

```
      public static void dos (int n){
```

```
        int i, j, k, sum;
        sum = 0;
        for ( i=1; i<=n; i++) {
            for ( j=1; j <= i*i; j++) {
                for ( k=1; k<= j; k++)  {
                sum = sum + 1;
                }
            }
        }
    }
```

5. Para cada uno de los algoritmos presentados calcule el T(n).

5.1.

```
int c = 1;
while ( c < n ) {
      algo_de_O(1)
      c = 2 * c;
}
```

5.2.

```
int c = n;
while ( c > 1 ) {
      algo_de_O(1)
      c = c / 2
}
```

5.3.

```
int x=1;
for (int i = 1; i < n; i = i+4)
      for (int j = n; j > 1; j = |j/4|)
            for (int k = 1; k < n; k = k*2)
                  x = x+1;
```

6. En complejidad computacional, se dice que el orden de ejecución de un algoritmo es el orden de la cantidad de veces de la instrucción más utilizada del mismo.

6.1. Dado el siguiente algoritmo, determine el valor de la variable suma.

```
private int suma(int n){
      int suma=1;
      for (int i=1;i<=n;i++)
            for (int j=1;j<=i;j++)
                  suma = suma + 1;
      return suma;
}
```

6.2. Determine el T(n) del algoritmo previo.

6.3. Dado el siguiente algoritmo, determine el valor de la variable producto.

```
private int producto(int n){
      int producto=1;
      for (int i=1;i<=n;i++)
            for (int j=1;j<=i;j++)
                  producto= 2*producto
      return producto;
};
```

7. Para cada uno de los siguientes fragmentos de código, determine en forma intuitiva el orden de ejecución:

7.1.

```
for(int i = 0; i< n; i++)
            sum++;
```

7.2.

```
for(int i = 0; i< n; i+=2)
            sum++;
```

7.3.

```
for(int i = 0; i< n; i++)
            for(int j = 0; j< n; j++)
                  sum++;
```

7.4.

```
for(int i = 0; i< n; i++)
            for(int j = 0; j< n*n; j++)
                  sum++;
```