Santiago Ochoa Castaño
Universidad Eafit
Medellín, Colombia
sochoac1@eafit.edu.co

Miguel Ángel Zapata Jiménez
Universidad Eafit
Medellín, Colombia
mazapataj@eafit.edu.co

Taller Nro. 5

Punto 1:

COMPLEJIDAD ASINTÓTICA:

```
public static int[] insertionSort (int[] nums){
   int temp=0; //T2(n)=c_1
   for(int i = 1; i < nums.length;i++){//T3(n)=c_2+c_3*n
      for(int j=i-1;j>=0 && nums[j+1] < nums[j];j--){//T4(n)=(c_4+c5(n-1))*n
         temp=nums[j+1];
         nums[j+1]=nums[j];
         nums[j]=temp;     //T5(n)=c_6(n-1)*n
      }
   }
   return nums;//T6(n)=c_7
}
T(n)=c_1+c_2+c_3*n+(c_4+c_5(n-1))n+(c_6(n-1))n+ c_7
```

ECUACIONES:

- **Ecuación de Complejidad:**

$$T(n) = c_5\, n^2 + c_6\, n^2 + c_3\, n + c_4\, n - c_5\, n - c_6\, n + c_1 + c_2 + c_7$$

- **Notación O:**

$$O(C_5n^2 + C_6n^2 + C_3n + C_4n - C_5n - C_6n + C_1 + C_2 + C_7)$$

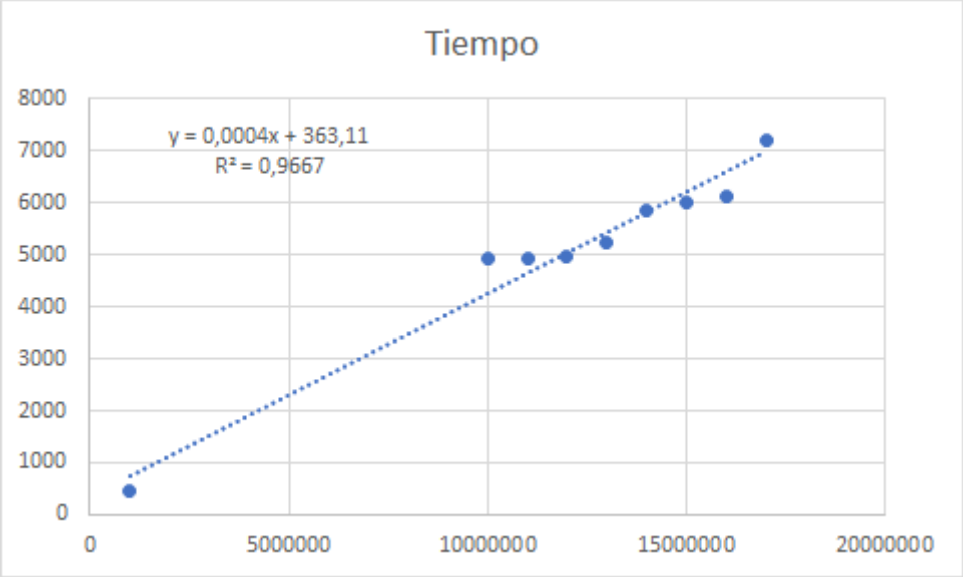
$$O(C_5n^2 + C_6n^2 + C_3n + C_4n - C_5n - C_6n)$$

$$O(n(C_4 - C_5 - C_6) + n^2(C_5 + C_6))$$

$$O(n^2(C_5 + C_6))$$

$$O(n^2)$$

GRAFICA:



¿Este algoritmo es apropiado para ordenar grandes volúmenes de datos?

El algoritmo itera sobre los índices del arreglo de forma que compara si el elemento insertado es menor que cada elemento en el arreglo a su izquierda. Para el peor de los casos, tendría que llegar hasta (n-1) (con n definida como el tamaño del arreglo) donde la notación se da como $O(n^2)$. Por lo que a la hora de ordenar grandes volúmenes de datos tiende a parecerse a una ecuación cuadrática que a nivel de complejidad no es la favorable. Sin embargo, no se puede decir que $O(n^2)$ satisface todos los casos, porque para el mejor de los casos se ejecuta a un tiempo O(n).

Punto 2:

COMPLEJIDAD ASINTÓTICA DEL ALGORITMO:

```
public class Suma {
   public static int sum(int[] a){
      int acum = 0; // C1
      for (int i = 0; i < a.length; i++){   // C2 + C3*n
         acum = acum + a[i]; // C4*n
      }
      return acum; // C5
   } // Ecuacion asintotica T(n) = C1 + C2 + C5 + (C3 + C4)*n
   // Ecuacion en notacion O: T(n) = O(n)
}
```

En este caso en concreto la n representa el tamaño del arreglo que le es ingresado al algoritmo. Existe la presencia de 5 constantes representadas con la letra C, las cuales nos hablan de operaciones en tiempo constante ligadas a unos ciclos “for” y otras operaciones presentes en el algoritmo.

ECUACIONES:

- **Ecuación de Complejidad:**

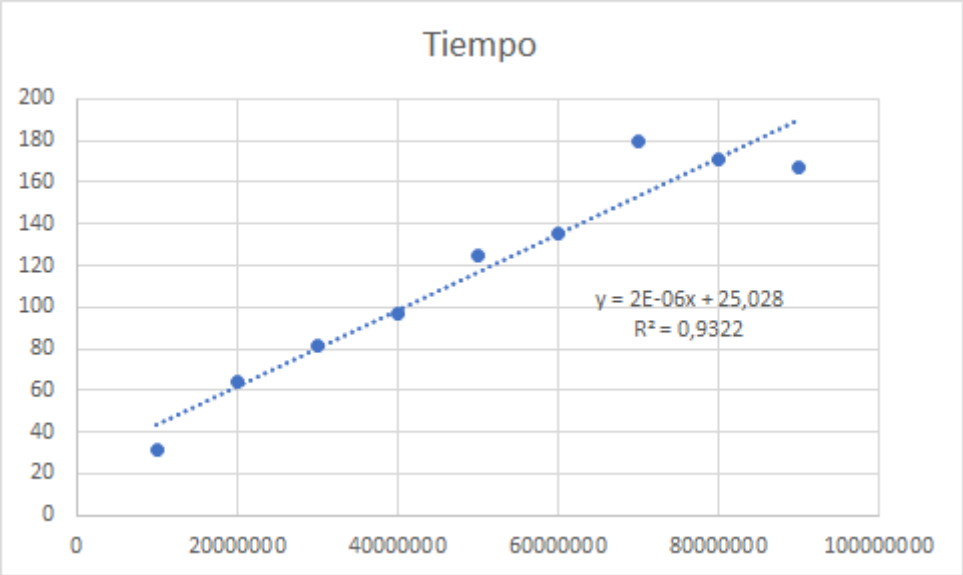
$$T(n) = C_3\, n + C_4\, n + C_1 + C_2 + C_5$$

- **Notación O:**

$$O(C_3 n + C_4 n + C_1 + C_2 + C_5)$$
$$O(C_3 n + C_4 n)$$
$$O(n(C_3 + C_4))$$
$$O(n)$$

Y de acuerdo a la notación O se comprende que T(n) = O(n) lo que nos indica que el tiempo de ejecución de este algoritmo tiene un crecimiento lineal siendo una complejidad adecuada. Esto permite realizar operaciones con mayor eficacia.

GRAFICA:



Como se puede observar en la gráfica el tiempo de ejecución de los datos crece de manera lineal siendo una complejidad adecuada. Al comparar la gráfica con la ecuación de complejidad se puede observar claramente que se habla de una tendencia lineal por lo que se puede apreciar que las ejecuciones no tardan un tiempo muy grande a la hora de procesar los datos que se le son asignados.

¿Existe una diferencia significativa, en el tiempo de ejecución, entre la implementación con recursión y la implementación con ciclos? ¿Por qué?

```
public class recursión
{
   public static int sumarValores(int array[], int posArray) {
      int tam = posArray;  //c1
      int rta;
      if (tam == 0) {
         return array[tam]; //c2
```

```
        } else {
            rta = (array[tam]) + sumarValores(array, tam - 1); //T(n) = c3 + T(n-1) / T(n) = O(n)
        }
        return rta;
    }

}
```

Ecuación de Complejidad:

$$T(n) = c_3\, n + c_1$$

Notación O:

$$O(C_3 n + C_1)$$

$$O(C_3 n)$$

$$O(n)$$

Si este algoritmo se implementara de manera de recursiva su complejidad seria de una tendencia lineal, es decir, O(n) y si lo comparamos con el caso de bucles podríamos apreciar que no existe un cambio significativo en el tiempo de ejecución. Por lo que para este caso específico podemos concluir que ambos métodos poseen una complejidad igual en notación O(n) por lo que sus tiempos siempre tendrán un crecimiento lineal.