8. Algoritmos de ordenamiento: Ordenamiento por inserción

Carlos Zerón Martínez

Universidad Nacional Autónoma de México

zeron@ciencias.unam.mx

Martes 12 de Enero de 2021

Introducción

- Muchas cosas en la vida real requieren procesarse de forma ordenada, hay muchas estrategias para hacerlo, dependiendo de cuántos objetos tenemos que ordenar
- El ordenamiento es una de las tareas en el mundo computacional hechas con mayor frecuencia
- Por ejemplo, es posible que necesitemos ordenar registros en una base de datos para poder buscar en ella de forma más eficiente o como formando parte de un algoritmo para resolver otro problema.

Insertion Sort (Ordenamiento por inserción)

Uno de los algoritmos de ordenamiento más simples

Para ordenar n elementos se llevan a cabo $n-1$ fases, en cada una de ellas mueve el elemento en la posición i hacia la izquierda hasta encontrar su posición correcta entre los primeros $i-1$ elementos.

Ejemplo: $A = [5, 2, 4, 6, 1, 3]$

- final de primera fase: $A = [\mathbf{2}, \mathbf{5}, 4, 6, 1, 3]$
- final de segunda fase: $A = [\mathbf{2}, \mathbf{4}, \mathbf{5}, 6, 1, 3]$
- final de tercera fase: $A = [\mathbf{2}, \mathbf{4}, \mathbf{5}, \mathbf{6}, 1, 3]$
- final de cuarta fase: $A = [\mathbf{1}, \mathbf{2}, \mathbf{4}, \mathbf{5}, \mathbf{6}, 3]$
- final de quinta fase: $A = [\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}]$

Insertion Sort (Ordenamiento por inserción)

```
public class InsertionSort {

    public static void sort(Comparable[] array) {
        int n = array.length;
        for (int i = 1; i < n; i++) {
            Comparable x = array[i];
            int j = i - 1;
            while (j >= 0 && array[j].compareTo(x) > 0) {
                array[j + 1] = array[j];
                j--;
            }
            array[j + 1] = x;
        }
    }

}
```

Insertion Sort (Ordenamiento por inserción)

- Cada fase se representa con el ciclo externo donde la variable i cambia su valor
- En el ciclo interno se busca el lugar que le corresponde al elemento x dentro de la parte ordenada (los $i-1$ que lo anteceden)
- Al encontrar la posición j donde el elemento es menor o igual a x (lo contrario de la condición del ciclo interno) se pone en la posición $j+1$ al elemento x
- El peor caso se tiene cuando el arreglo está ordenado en forma decreciente o no creciente, pues el ciclo interno tendría que repetirse $i-1$ veces siempre. Puesto que i toma valores desde 1 hasta $n-1$, entonces el tiempo de ejecución de los dos ciclos es proporcional a la suma de los primeros $n-1$ números naturales, lo cual es $O(n^2)$ y este es el tiempo de ejecución total.