@BritoAlv

Insertion Sort

Dado un array de tamaño N de **int**, ordenarlo de menor a mayor

1. La idea es recorrer el array entero desde la posición 1 hasta la posición $N-1$ (última), **garantizando que al acabar la iteración i, el array desde la posición 0 hasta la i esté ordenado.** Notar que esto implica que al **comenzar la iteración en la posición i el array está ordenado desde la posición 0 hasta la posición $i-1$.**

2. Por lo que el algoritmo consiste en dos partes, un **for** para recorrer el array desde la posición 1 hasta la $N-1$, y en cada iteración del **for** un algoritmo que ubique al número i en la posición correspondiente entre 0 a $i-1$, de forma que el arreglo desde 0 a i quede ordenado.

3. El primer **for** recorre $N-1$ elementos $\rightarrow O(n)$, para cada uno de esos elementos aplicamos otro algoritmo, este puede ser:

4. **Swaps**, ir cambiando hasta llegar a la posición correcta, sería así ejemplo: $2,5,6,3 \rightarrow 2,5,3,6 \rightarrow 2,3,5,6$. Este método de **Swaps** realiza dos operaciones: **Comparar e intercambiar, compara si no es esta la posición, intercambia y vuelve a comparar, hasta llegar a la posición, la cantidad de comparaciones es a lo más una unidad mayor que la cantidad de intercambios, usando este método de swap.** El costo por defecto de la operación de comparar e intercambiar es constante, pero y si no lo fuera, una manera de reducir el número de comparaciones es usar binary search para hallar la posición donde va el elemento i y realizar los swaps correspondientes.

5. **Binary Search**, ir dividiendo el arreglo en mitades, hasta encontrar la posición donde va el número. ***Teniendo esta posición hacemos los swaps, o shift the array one position to the right***.

Observaciones

1. Para cada i insertion sort, realiza lo siguiente, inserta el número i en un array de tamaño i, a través de swaps, independientementde de si se use binary search o no, el tendrá que realizar los swaps para insertar al número i, esto se realiza en tiempo lineal, **la complejidad del siguiente problema es lineal: dado un arreglo de tamaño n+1, insertar el número en la posición n en la**

posición i, el peor caso de esto es cuando $i = 0$ que tendría que realizar n swaps, por lo que su complejidad es linear. La utilidad de binary search es para reducir el número de comparaciones, que sería eficiente cuando realizar comparaciones tenga un costo mayor. Por lo que concluyo que la complejidad es cuadrática. $O(n * (n + \log_2 n))$ que es $O(n^2)$.

2. Insertion Sort es ideal para arreglos almost-sorted porque su tiempo de ejecución radica en cuantas veces haya que realizar los **swap** para cada elemento. **De esto se saca que en un array ordenado (ideal) su complejidad es $O(n)$ porque no hay que hacer swaps y en uno ordenado pero decrecientemente su complejidad es $O(n^2)$ porque hay que hacer todos los swaps.**

3. Los algoritmos de sorting, son **abstracciones** ellos deben funcionar con cualquier estructura que tenga definida que significa que un elemento sea mayor que otro.