Captura de pantalla de un celular

Descripción generada automáticamenteORDENAMIENTO

Comportamiento en términos de comparaciones:

* **Bubble sort y Insertion sort** tienen un comportamiento cuadrático 𝑂(𝑛2) en el número de comparaciones en el peor caso. Esto significa que, para arreglos grandes, estos algoritmos pueden volverse ineficientes comparados con algoritmos 𝑂(𝑛log𝑛) como Quick sort.
* **Quick sort**, en promedio, realiza 𝑂(𝑛log𝑛) comparaciones. Sin embargo, en el peor caso (cuando la partición no está balanceada), puede realizar hasta 𝑂(𝑛2) comparaciones.

***Mejor caso:*** Los tres algoritmos tienen su mejor caso cuando el arreglo está preordenado o casi preordenado. En este caso, los algoritmos de ordenamiento más simples como Bubble y Insertion sort pueden ser competitivos debido a su menor cantidad de comparaciones constantes.

***Peor caso:*** Los algoritmos **Bubble sort y Insertion sort** muestran su peor caso cuando el arreglo está en orden inverso, lo que requiere el máximo número de comparaciones y permutaciones. Quick sort muestra su peor caso cuando la elección del pivote no divide el arreglo de manera equitativa en cada paso, llevando a una complejidad cuadrática.

*Comportamiento en términos de permutaciones de datos:*

* **Bubble sort y Insertion sort** tienen 𝑂(𝑛2) permutaciones en el peor caso, ya que ambos realizan muchas operaciones de intercambio para ordenar el arreglo.
* **Quick sort** tiene 𝑂(𝑛log𝑛) permutaciones en promedio, pero puede llegar a 𝑂(𝑛2) en el peor caso debido a particiones desbalanceadas que llevan a muchas iteraciones.

***Mejor caso:*** Todos los algoritmos tienen su mejor caso cuando el arreglo está preordenado o casi preordenado, minimizando así el número de permutaciones.

***Peor caso:*** Bubble sort e Insertion sort muestran su peor caso cuando el arreglo está en orden inverso. Quick sort muestra su peor caso cuando el pivote elegido no divide el arreglo equitativamente, resultando en muchas particiones y permutaciones.

*Comportamiento en términos del uso de memoria auxiliar de datos:*

* Todos los algoritmos mencionados pueden operar in situ, es decir, no requieren memoria auxiliar significativa más allá de unas pocas variables temporales.

Elección de algoritmos en máquinas con recursos limitados:

* ***Memoria limitada:*** Dado que todos son algoritmos in situ y no requieren memoria adicional significativa, cualquier algoritmo podría ser utilizado en una máquina con recursos de memoria limitados. Sin embargo, en términos de tiempo de ejecución, **Insertion sort** podría ser preferible debido a su menor complejidad constante en el mejor caso 𝑂(𝑛)
* ***Capacidad de procesamiento limitado:*** En una máquina con capacidad de procesamiento limitado, es crucial minimizar el número de operaciones. Aquí, a pesar de su complejidad cuadrática en el peor caso, **Quick sort** en promedio 𝑂(𝑛log𝑛) podría ser preferido debido a su eficiencia general en el tiempo de ejecución, especialmente para arreglos grandes.

“Si tengo una lista con N cantidad de elementos y la estaba ordenando con un algoritmo de complejidad O(n^2). Mañana tendrá N+1 elemento y tengo la posibilidad de aplicar un algoritmo de ordenación de complejidad O(n):”

* ¿Me conviene cambiar el tipo de algoritmo o sigo con el que estaba?

--- RESPUESTA ---

Conviene cambiar a un algoritmo de ordenación de complejidad O(n), porque la cantidad de iteraciones aumenta de forma lineal y no exponencial como los es O(n^2). Es decir que a medida que crece la cantidad de elementos de la lista su procesamiento será más eficiente con O(n).

--- OTRA RESPUESTA ---

Si se agrega un elemento a la lista, la cantidad de elementos será N + 1. Si se utiliza un algoritmo de complejidad O(n), el tiempo de ejecución será proporcional a N.

En este caso, el tiempo de ejecución del nuevo algoritmo será menor que el tiempo de ejecución del algoritmo anterior.

Por lo tanto, en términos de eficiencia, conviene cambiar al nuevo algoritmo de ordenación con complejidad O(n) cuando se agrega un elemento adicional a la lista.