Tecnológico de Monterrey

Actividad: Implementación de Algoritmos de Ordenamiento

Leon Daniel Vilchis Arceo A01641413

Álvaro González Martínez A01646343

Gregorio Alejandro Orozco Torres A01641967

Daniel Hernández Gutiérrez A01640706

Lorenzo Orrante Román A01641580

17 de septiembre del 2025

Programación de Estructuras de Datos y Algoritmos Fundamentales

Grupo 604

Bubble Sort:

Es un algoritmo que se encarga de ordenar una lista o array, recorriendo la lista para comparar los elementos adyacentes y los intercambia si el orden no es correcto. En cada iteración el valor más grande se mueve hasta el final de la lista y cuando el algoritmo acaba de iterar, la lista o array se debe de encontrar en orden ascendente. Este tipo de algoritmo puede tener una complejidad temporal de O(n) en el mejor de los casos, que es cuando la lista ya está ordenada, así que regularmente su complejidad será de O(n^2) al realizar comparaciones e intercambios en los bucles anidados. La complejidad espacial del Bubble Sort es de O(1) ya que solo requiere una variable temporal para poder funcionar adecuadamente. Este algoritmo brilla en eficiencia cuando se utiliza en listas o arrays pequeños o unas de tamaño más grande que no necesiten muchos cambios, cuando se intenta implementar en conjuntos de datos muy grandes se vuelve lento, ya que se necesita iterar una gran de cantidad de veces para que todos los datos queden de forma ordenada.

Selection Sort:

Algoritmo de ordenamiento que recorre la lista buscando el elemento más pequeño (o grande) y lo coloca en su posición correcta. Repite este proceso hasta que toda la lista esté ordenada.
Su complejidad en mejor, promedio y peor caso es O(n^2) ya que siempre hace las mismas comparaciones, sin importar si el array ya está casi ordenado, algunos de los algoritmos más eficientes que este son Merge Sort y Quick Sort, ya que utilizan O(n log n) que son más rápidos en listas más grandes. Entre sus ventajas encontramos que es fácil de entender y de usar, no necesita memoria extra y hace pocos intercambios. Sus desventajas son que es muy lento en listas grandes, no aprovecha si el array ya está casi ordenado y es superado por más algoritmos.

Insertion Sort:

El algoritmo se encarga de ordenar la lista yendo elemento por elemento comenzando desde la segunda posición y compara el elemento en el que se encuentra con los que están a la izquierda. Si se encuentra con valores más grandes los desplaza una posición hacia la derecha y así lo va haciendo hasta encontrar la posición correcta del número actual. Esto se repite hasta que todo el array quede ordenado. Su complejidad puede llegar a ser de O(n), sin embargo es más

susceptible a que sea de Orden(n^2) ya que su número de iteraciones depende mucho de lo previamente ordenada que haya estado la lista. Tiene un orden espacial de O(1) ya que no ocupa memoria adicional. Este algoritmo es más eficiente para arreglos pequeños o arreglos que ya casi están ordenados, pero en el caso de listas más grandes o con los números muy aleatorios puede resultar ineficiente y generar una gran cantidad de iteraciones.

Merge Sort:

Algoritmos conocido por dividir y vencer. Consiste en dividir el arreglo entre dos hasta que nos queda un elemento, luego los va uniendo en orden hasta terminar con el arreglo completo. En el mejor y peor caso, además del tiempo promedio, de complejidad es O(nlog n) y en espacio utiliza O(n) porque necesita de arreglos temporales para guardar las mitades. Es eficiente cuando queremos un tiempo consistente y funciona bien en listas enlazadas. Es deficiente cuando tenemos listas muy pequeñas porque tenemos algoritmos como insertion sort que son más rápidos, además de que utiliza más memoria que otros algoritmos.

Quick Sort:

El algoritmo Quick Sort es un método de ordenamiento que organiza un arreglo seleccionando un pivote y separando los elementos en dos grupos: los menores a la izquierda y los mayores a la derecha, aplicando después el mismo proceso a cada grupo de manera recursiva. Su complejidad temporal es de O(n log n) , lo que lo hace muy eficiente en la práctica, mientras que en el peor caso puede degradarse a $O(n^2)$ si los pivotes se eligen mal por ejemplo, en arreglos ya ordenados. La complejidad espacial es de O(log n) debido a la recursión. Quick Sort es especialmente eficiente para arreglos grandes y en la mayoría de los casos reales, pero puede resultar ineficiente cuando los datos están parcialmente ordenados y no se toman medidas como la elección aleatoria del pivote.

Link del video:

https://youtu.be/5rjiHkz7dew