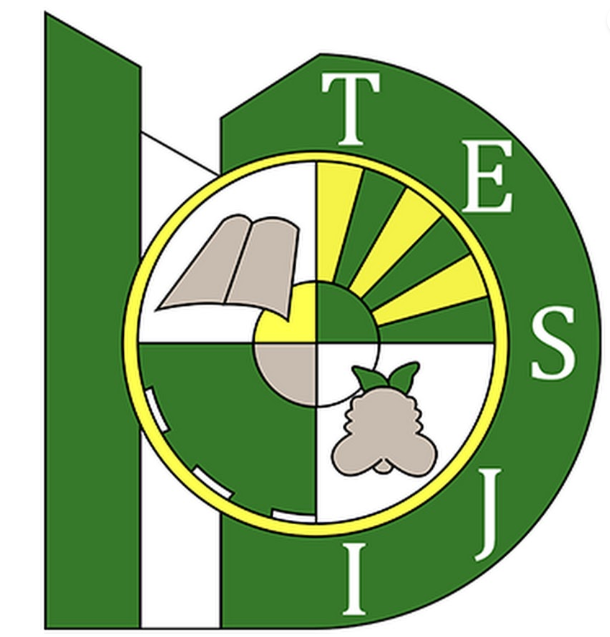
**Tecnológico de Estudios Superiores de Jilotepec**



**Alumna:**

Gabriela de Jesús Rueda N.L: 08.

**Profesor:** Anselmo Martínez Montalvo.

**Materia:** Apoyo Formación Profesional (Programación).

Nombre del trabajo:

**Investigación de los Métodos de Ordenamiento.**

**Grupo:** 3101

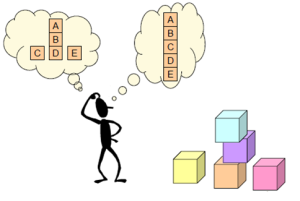
**INTRODUCCIÓN**

La función de los algoritmos de ordenamiento es ordenar y acomodar los vectores que el usuario introduce de manera aleatoria. Es decir, para que se puedan ordenar una cantidad exacta de números guardados en un vector o matriz, se pueden usar distintos métodos o bien algoritmos los cuales tienen características diferentes y complejas.

Existe desde el método mas simple hasta el mas complejo.

En el caso de datos numéricos, el propósito de los métodos de ordenamiento es ordenar los valores dentro del vector dependiendo a un criterio dado (de ordenamiento claramente).

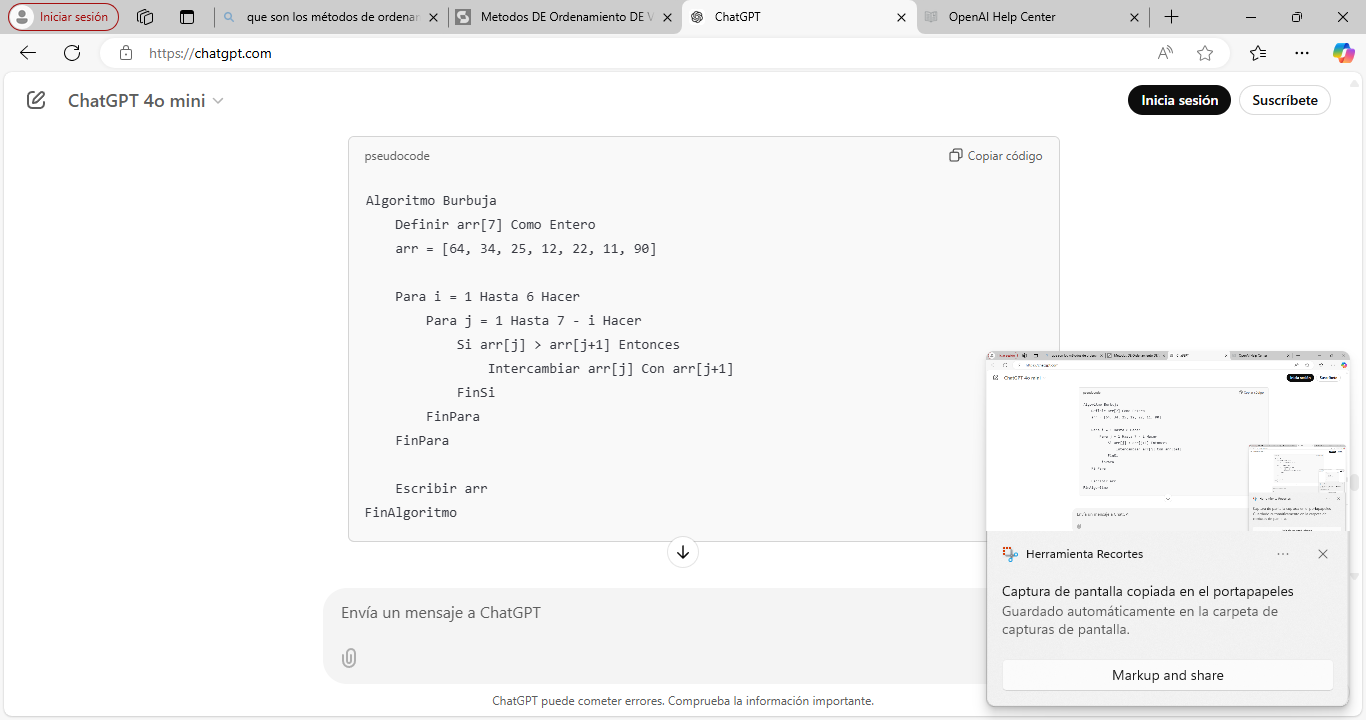
El **ordenamiento** es un proceso esencial en informática que organiza elementos en estructuras de datos como arreglos o vectores, ya sea de forma ascendente o descendente. Es clave en aplicaciones como búsqueda, visualización de datos y optimización. Existen diversos métodos de ordenamiento, que varían en **eficiencia** y **complejidad**. Los **métodos simples** como **burbuja (Bubble Sort), inserción** y **selección (Selection Sort)** son fáciles de entender pero poco eficientes para grandes volúmenes de datos. Por otro lado, algoritmos más avanzados como **rápido (Quick Sort), fusión (Merge Sort)** y **montículos (Heap Sort)** son más eficientes para manejar grandes cantidades de datos.



1. **Método de la burbuja (Bubble Sort)**

**Algoritmo:**

El algoritmo de ordenamiento por burbuja compara elementos adyacentes en un vector y los intercambia si están en el orden incorrecto. Este proceso se repite a lo largo de todo el vector varias veces hasta que no se realizan intercambios.

****

**Ventajas:**

* Simplicidad: El algoritmo es fácil de entender e implementar.
* Buen rendimiento para listas pequeñas.

**Desventajas:**

* Ineficiencia: Tiene una complejidad de O(n²), lo que lo hace lento para vectores grandes.
* Requiere múltiples comparaciones innecesarias.

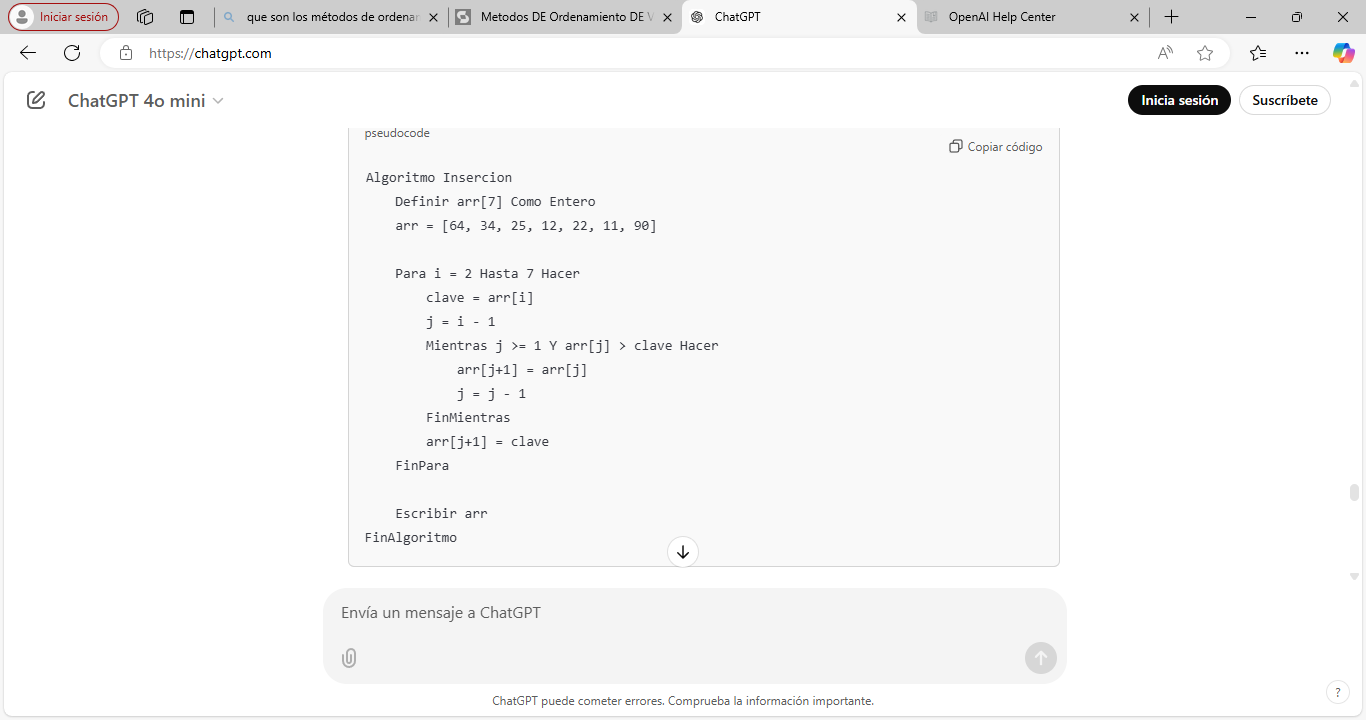
**Conclusión:**

* El algoritmo de la burbuja es útil en situaciones donde la simplicidad es más importante que la eficiencia. Es más adecuado para listas pequeñas o listas casi ordenadas.

1. **Método de inserción (Insertion Sort)**

**Algoritmo:**

El algoritmo de inserción construye el arreglo ordenado elemento por elemento, insertando cada elemento en su posición correcta dentro de la parte ordenada del arreglo.

****

**Ventajas:**

* Simplicidad: Al igual que el algoritmo de burbuja, es fácil de entender e implementar.
* Eficiente para listas pequeñas o parcialmente ordenadas.

**Desventajas:**

* Ineficiencia para grandes vectores: Su complejidad por lo que no es adecuado para grandes volúmenes de datos.
* Comparaciones y movimientos innecesarios.

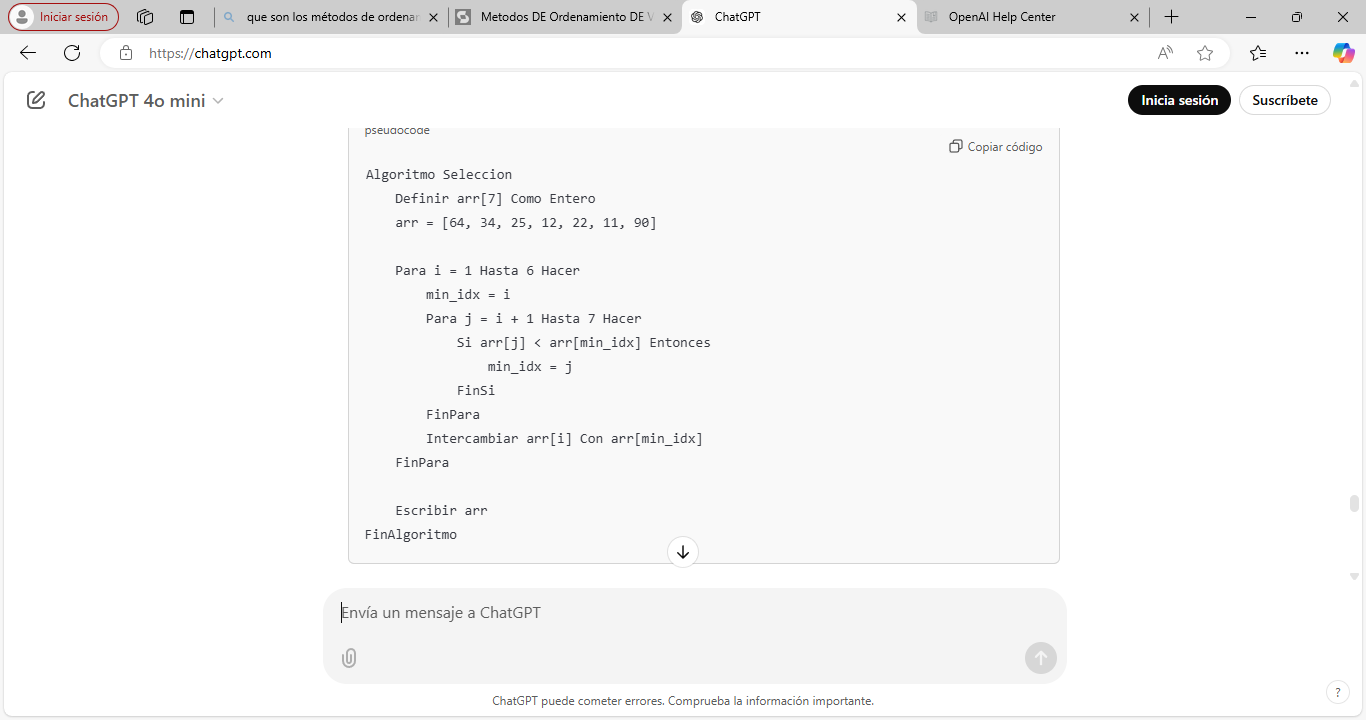
**Conclusión:**

El algoritmo de inserción es muy eficiente en listas pequeñas o cuando los datos ya están parcialmente ordenados, pero no es adecuado para grandes cantidades de datos.

1. **Método de selección (Selection Sort)**

**Algoritmo:**

Este algoritmo busca el valor mínimo (o máximo) en el vector y lo coloca en la posición correcta. Luego, repite este proceso para el resto del vector.

****

**Ventajas:**

* Simplicidad es decir es fácil de implementar y entender.
* Intercambios mínimos: Solo realiza un intercambio por iteración.
* No requiere memoria adicional: Es un algoritmo in situ.

**Desventajas:**

* Ineficiencia: Su complejidad, lo que lo hace ineficaz para grandes volúmenes de datos.
* No es estable: Puede cambiar el orden de los elementos iguales.

**Conclusión:**

El algoritmo de selección es sencillo y tiene menos intercambios que el de burbuja, pero su eficiencia es limitada en vectores grandes y no es estable.

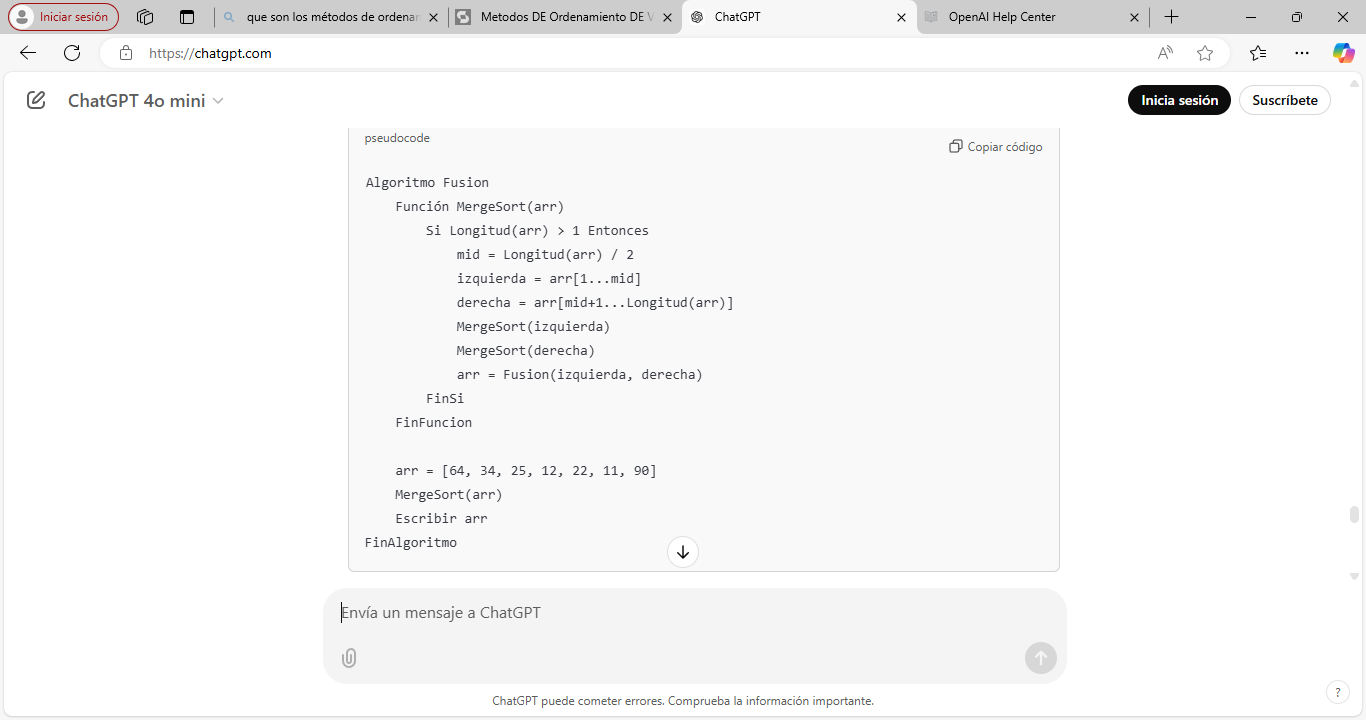
1. **Método de fusión (Merge Sort)**

**Algoritmo:**

Dividir: Divide el arreglo en dos mitades.

Conquistar: Recursivamente ordena ambas mitades.

Fusionar: Combina las dos mitades ordenadas en un solo arreglo ordenado.

****

**Ventajas:**

* Eficiencia: Tiene una complejidad, lo que lo hace mucho más eficiente que los algoritmos de O(n²).
* Estabilidad: Mantiene el orden relativo de los elementos iguales.
* Siempre tiene el mismo rendimiento: Independientemente de si la lista está ordenada o no.

**Desventajas:**

* Uso de memoria adicional: Requiere espacio adicional para almacenar los vectores divididos.
* Complejidad de implementación: Es más complejo de implementar que los métodos de O(n²).

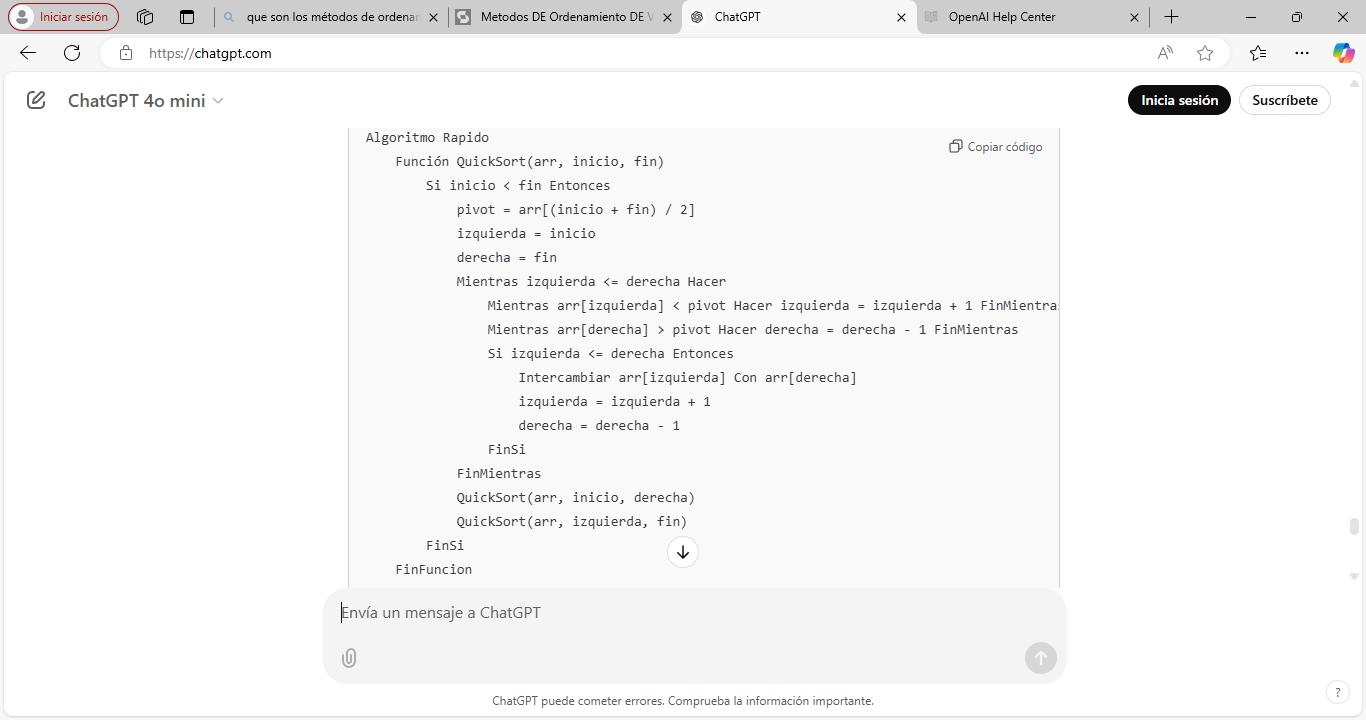
**Conclusión:**

El algoritmo de fusión es ideal para grandes vectores debido a su eficiencia. Aunque necesita más memoria, su rendimiento constante lo convierte en una excelente opción en muchos casos.

1. **Método rápido (Quick Sort)**

**Algoritmo:**

El ordenamiento rápido es un algoritmo basado en la técnica de "divide y vencerás". Elige un elemento como pivote y divide el arreglo en dos subarreglos: uno con los elementos menores que el pivote y otro con los elementos mayores. Luego, recursivamente, ordena los subarreglos.

****

**Ventajas:**

* Eficiencia: Tiene una complejidad promedio de O(n log n), lo que lo hace rápido para vectores grandes.
* Rendimiento consistente en la mayoría de los casos.

**Desventajas:**

* No es estable: Puede alterar el orden relativo de los elementos iguales.

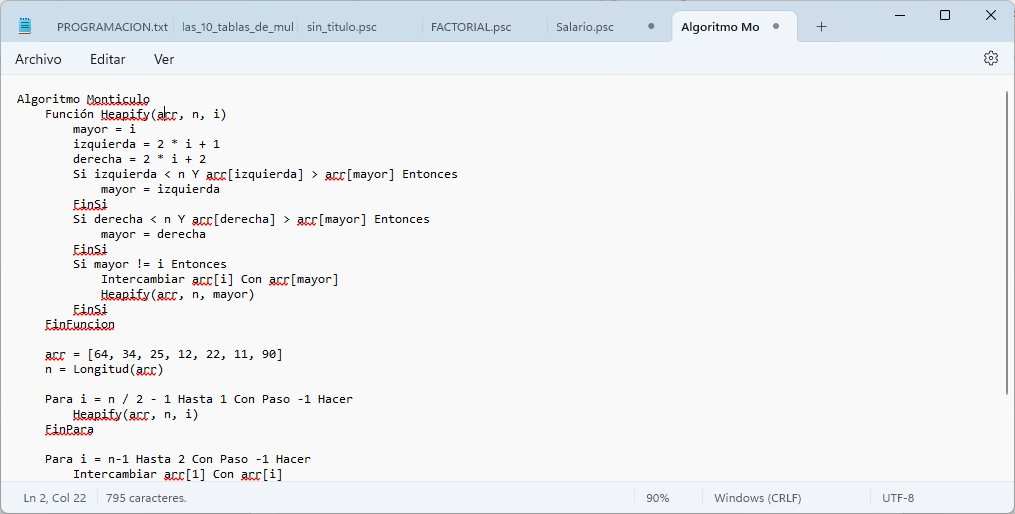
**Conclusión:**

El algoritmo rápido es uno de los más eficientes y populares en la práctica, pero depende de una buena elección de pivote. A pesar de su mala complejidad en el peor caso, su rendimiento promedio es excepcional.

1. **Método de ordenamiento por montículos (Heap Sort.**

**Algoritmo:**

El algoritmo **Heap Sort** se basa en aprovechar las características de una estructura de datos llamada **heap binario** para ordenar un arreglo de manera eficiente. Primero, transforma el arreglo en un **heap máximo** (un tipo especial de árbol binario en el que el valor de cada nodo es mayor que el de sus hijos). Luego, el proceso de ordenación comienza con la extracción del **elemento máximo** del heap (que siempre está en la raíz), colocándolo en la última posición del arreglo.



**Ventajas:**

* Eficiencia: Tiene una complejidad de O.
* No requiere memoria adicional: Es un algoritmo in situ.
* Buen rendimiento para vectores grandes.

**Desventajas:**

* No es estable: Al igual que otros algoritmos no estables, puede cambiar el orden de los elementos iguales.
* Complejidad en la implementación: Es más difícil de implementar que los métodos más simples.

**Conclusión:**

El algoritmo de montículo es muy eficiente para grandes vectores, aunque tiene la desventaja de no ser estable. Es adecuado cuando la eficiencia es más importante que la estabilidad.