1. Quicksort

Descripción: Divide el arreglo en dos partes alrededor de un pivote (elemento que separa elementos más pequeños y más grandes).

Luego ordena recursivamente cada subarreglo.

Ventajas:

-Excelente rendimiento promedio: 𝑂(𝑛log⁡𝑛)

-In-place (requiere poco espacio adicional).

Desventajas:

-Peor caso 𝑂(𝑛2) si el pivote se elige mal.

-Requiere optimización para arreglos pequeños (es mejor usar inserción directa para arreglos de < 10 elementos).

¿Cuándo usarlo?

-Ideal para datos grandes y desordenados.

-Menos eficiente si el arreglo está casi ordenado o tiene muchos elementos iguales (usar alternativas como Heapsort).

2. Inserción Directa

Descripción: Construye un arreglo ordenado uno a uno, colocando cada elemento en su posición correcta.

Ventajas:

-Muy simple de implementar.

-Eficiente para arreglos pequeños o casi ordenados: 𝑂(𝑛)

-O(n) en el mejor caso.

Desventajas:

-Ineficiente en arreglos grandes y desordenados: O(n 2 ).

Cuándo usarlo:

-Arreglos pequeños (<10-20 elementos).

-Arreglos parcialmente ordenados (por ejemplo, cuando se realizan pequeños ajustes en datos previamente ordenados).

3. Heapsort

Descripción: Usa una estructura de arbol binario para ir encontrando el máximo/mínimo y reorganizar el arreglo.

Ventajas:

-Tiempo garantizado O(n log n), independientemente del orden inicial.

-No requiere mucho espacio adicional.

Desventajas:

-No es tan rápido en la práctica como Quicksort (peor acceso a memoria debido a saltos no secuenciales).

Cuándo usarlo:

-Cuando se necesita un tiempo garantizado O(nlogn).

-Adecuado para datos con restricciones de espacio o cuando no se puede garantizar un pivote eficiente.

4. Shellsort

Descripción: Divide el arreglo en subarreglos basados en "saltos" (gaps) decrecientes y aplica inserción directa en cada subarreglo.

Ventajas:

-Más eficiente que la inserción directa en arreglos grandes.

-Adaptable para arreglos parcialmente ordenados.

Desventajas:

-Rendimiento depende de la secuencia de saltos; sin optimización, puede llegar a O(n 2).

-Menos común que otros algoritmos.

Cuándo usarlo:

-Cuando no se desea un algoritmo más complejo pero se necesita algo más eficiente que inserción directa.

-Si el arreglo tiene un tamaño intermedio y cierta desorganización.

5. Burbuja

Descripción: Intercambia elementos adyacentes si están desordenados, repitiendo este proceso varias veces.

Ventajas:

-Simple y fácil de entender.

-Detecta si el arreglo está ordenado temprano (en el mejor caso).

Desventajas:

-Ineficiente para arreglos grandes: O(n 2 ) en promedio y peor caso.

-Generalmente más lento que otros algoritmos 𝑂(𝑛2) como inserción directa o selección directa.

Cuándo usarlo:

-Solo para fines educativos o para arreglos muy pequeños y simples.

6. Selección Directa

Descripción: Encuentra el elemento mínimo (o máximo) en cada iteración y lo mueve a su posición final.

Ventajas:

-Siempre realiza 𝑂(𝑛2) comparaciones, independientemente del orden inicial.

-No requiere espacio adicional (in-place).

Desventajas:

-No aprovecha si el arreglo ya está parcialmente ordenado.

-Más lento en la práctica que inserción directa.

Cuándo usarlo:

-Cuando se minimiza el número de intercambios (por ejemplo, en medios con alto costo de escritura como discos SSD o memorias flash).

7. Binsort (Bucket Sort)

Descripción: Distribuye los elementos en "cubetas" (buckets), los ordena individualmente y los combina.

Ventajas:

Muy eficiente si los datos están uniformemente distribuidos y el rango k es pequeño: O(n+k).

Desventajas:

Ineficiente si los datos no están bien distribuidos (puede degradar a 𝑂(𝑛2)).

Consume más memoria.

Cuándo usarlo:

Cuando se conoce el rango de los datos y están bien distribuidos (por ejemplo, al ordenar números decimales entre 0 y 1).

No es adecuado para datos dispersos o de rango desconocido.

8. Ordenamiento por Cuenta (Counting Sort)

Descripción: Cuenta las ocurrencias de cada valor en un arreglo auxiliar y usa esta información para reconstruir el arreglo ordenado.

Ventajas:

Tiempo lineal O(n+k) si el rango k es pequeño en comparación con 𝑛

No requiere comparaciones.

Desventajas:

Solo funciona con datos discretos como enteros o caracteres.

Consume mucha memoria si el rango k es grande.

Cuándo usarlo:

Para datos discretos con un rango pequeño, como clasificar edades o calificaciones en un examen.

No es eficiente para datos dispersos o no numéricos.

Comparación Final: ¿Cuál usar y cuándo?

Algoritmo Mejor caso de uso

Quicksort Datos grandes y desordenados (generalmente el mejor).

Inserción Directa Datos pequeños o parcialmente ordenados.

Heapsort Datos grandes con necesidad de tiempo garantizado.

Shellsort Datos medianos y parcialmente ordenados.

Burbuja Datos muy pequeños o como herramienta educativa.

Selección Directa Minimizar intercambios, no importa número de comparaciones.

Binsort Datos uniformemente distribuidos en un rango conocido.

Orden por Cuenta Datos discretos con rango pequeño (enteros o caracteres).