1. **Selection Sort**

- **Mejor caso**: $O(n^2)$
  - No tiene una ventaja en el mejor caso, siempre realiza el mismo número de comparaciones y movimientos.
- **Peor caso**: $O(n^2)$
  - Siempre realiza el mismo número de comparaciones y movimientos independientemente del orden de los elementos.

2. **Insertion Sort**

- **Mejor caso**: $O(n)$
  - Ocurre cuando la lista ya está ordenada. Solo se realizan comparaciones sin movimientos.
- **Peor caso**: $O(n^2)$
  - Ocurre cuando la lista está ordenada en orden inverso. Cada elemento debe compararse y moverse a la posición correcta.

3. **Merge Sort**

- **Mejor caso**: $O(n \log n)$
  - Siempre divide y conquista la lista, realizando el mismo número de operaciones independientemente del orden inicial.
- **Peor caso**: $O(n \log n)$
  - La estructura divide y vencerás asegura que el número de operaciones sea consistente.

4. **Heap Sort**

- **Mejor caso**: $O(n \log n)$
  - La estructura del heap asegura que la complejidad sea $O(n \log n)$ en todos los casos.
- **Peor caso**: $O(n \log n)$
  - Similar al mejor caso, la complejidad es consistente debido a la naturaleza del heap.

5. **Quick Sort**

- **Mejor caso**: $O(n \log n)$
  - Ocurre cuando el pivote divide las listas de manera balanceada.
- **Peor caso**: $O(n^2)$
  - Ocurre cuando el pivote es el menor o el mayor elemento repetidamente, dividiendo la lista en una muy desbalanceada.

6. **Counting Sort**

- **Mejor caso**: $O(n + k)$
  - Siempre lineal, ya que cuenta las ocurrencias de cada elemento.

- **Peor caso**: $O(n + k)$
  - La complejidad es consistente si k (rango de los valores) es razonablemente pequeño en comparación con n.

7. Radix Sort

- **Mejor caso**: $O(n \cdot d)$
  - Es lineal respecto al número de elementos y la longitud de los dígitos.
- **Peor caso**: $O(n \cdot d)$
  - Similar al mejor caso, siempre lineal respecto al número de elementos y la longitud de los dígitos.

8. Bucket Sort

- **Mejor caso**: $O(n + k)$
  - Ocurre cuando los elementos están distribuidos uniformemente entre los buckets.
- **Peor caso**: $O(n^2)$
  - Ocurre cuando todos los elementos caen en el mismo bucket, lo que requiere ordenar ese bucket usando un algoritmo de ordenación diferente.

Resumen

Algoritmo	Mejor caso	Peor caso
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n^2)$
Counting Sort	$O(n + k)$	$O(n + k)$
Radix Sort	$O(n \cdot d)$	$O(n \cdot d)$
Bucket Sort	$O(n + k)$	$O(n^2)$

Explicaciones Adicionales

- **Selection Sort**: Siempre selecciona el menor elemento de la lista desordenada y lo coloca en la posición correcta. El número de comparaciones no cambia con el orden inicial de la lista.
- **Insertion Sort**: Muy eficiente para listas que ya están casi ordenadas.
- **Merge Sort**: Divide y conquista, garantizando un tiempo consistente de $O(n \log n)$.
- **Heap Sort**: Utiliza una estructura de heap para mantener el orden.
- **Quick Sort**: La elección del pivote es crucial para su rendimiento.
- **Counting Sort**: Eficiente para listas con un rango limitado de elementos.

- **Radix Sort**: Ordena en varias pasadas basándose en dígitos o caracteres.
- **Bucket Sort**: Eficiente si los elementos están uniformemente distribuidos.