Contenido

[1. Búsqueda Binaria 2](#_Toc182766951)

[2. Bubble Sort 2](#_Toc182766952)

[3. Selection Sort 2](#_Toc182766953)

[4. Insertion Sort 3](#_Toc182766954)

[5. Merge Sort 3](#_Toc182766955)

[6. Quick Sort 4](#_Toc182766956)

[Resumen Complejidad BigO: 4](#_Toc182766957)

# **1. Búsqueda Binaria**

La búsqueda binaria se utiliza en listas ordenadas para encontrar un elemento de manera eficiente, dividiendo repetidamente la lista en mitades.

**Cálculo del Big O**:

* En cada paso, la lista se reduce a la mitad.
* Si la lista tiene n elementos, el número de pasos necesarios es log2(n), ya que seguimos dividiendo hasta que quede un único elemento.

**Complejidad temporal (Big O)**:

* **Mejor caso**: O (1) (si el elemento buscado está en el medio desde el principio).
* **Peor caso**: O (log (n))

# **2. Bubble Sort**

Este algoritmo compara elementos adyacentes y los intercambia si están desordenados. Este proceso se repite hasta que la lista esté ordenada.

**Cálculo del Big O**:

* En el peor caso, hace n−1 pasadas, y en cada pasada compara n−1, n−2, ..., 1 elementos.
* La cantidad total de comparaciones es una suma aritmética: 1+2+...+(n−1) = n(n-1)/2 lo que da O(n2)

**Complejidad temporal (Big O)**:

* **Mejor caso**: O(n) (si la lista ya está ordenada, solo una pasada).
* **Peor caso**: O(n2)
* **Caso promedio**: O(n2)

# **3. Selection Sort**

Este algoritmo encuentra el elemento mínimo (o máximo) en la lista y lo coloca en su posición correcta, repitiendo este proceso para los elementos restantes.

**Cálculo del Big O**:

* Siempre realiza n−1, n−2, ..., 1 comparaciones, similar a Bubble Sort.
* El número total de comparaciones es O(n2).
* Sin embargo, los intercambios son menos frecuentes que en Bubble Sort.

**Complejidad temporal (Big O)**:

* **Mejor caso**: O(n2) (porque siempre recorre toda la lista para buscar el mínimo).
* **Peor caso**: O(n2)
* **Caso promedio**: O(n2)

# **4. Insertion Sort**

Este algoritmo toma un elemento de la lista y lo inserta en la posición correcta dentro de la porción previamente ordenada.

**Cálculo del Big O**:

* En el peor caso (lista ordenada en orden inverso), cada elemento debe compararse con todos los elementos anteriores, lo que da O(n2)
* En el mejor caso (lista ya ordenada), solo realiza n−1 comparaciones, lo que da O(n).

**Complejidad temporal (Big O)**:

* **Mejor caso**: O(n)
* **Peor caso**: O(n2)
* **Caso promedio**: O(n2)

# **5. Merge Sort**

Merge Sort es un algoritmo basado en la técnica "divide y vencerás". Divide la lista en mitades hasta que cada sublista tiene un solo elemento y luego las combina ordenadamente.

**Cálculo del Big O**:

* En cada nivel de recursión, la lista se divide en dos mitades, lo que da O (log n) niveles.
* En cada nivel, se realizan O(n) comparaciones para mezclar.
* Por tanto, el tiempo total es O (nlog n)

**Complejidad temporal (Big O)**:

* **Mejor caso**: O (nlog n)
* **Peor caso**: O (nlog n)
* **Caso promedio**: O (nlog n)

# **6. Quick Sort**

Quick Sort también utiliza "divide y vencerás", seleccionando un pivote y particionando la lista en elementos menores y mayores al pivote.

**Cálculo del Big O**:

* En el peor caso (si el pivote es el menor o mayor elemento en cada partición), el tiempo es O(n2)
* En el caso promedio y mejor caso, la partición es equilibrada, dando O (nlog n)

**Complejidad temporal (Big O)**:

* **Mejor caso**: O (nlog n)
* **Peor caso**: O(n2)
* **Caso promedio**: O (nlog n)

|  |  |  |  |
| --- | --- | --- | --- |
| Algoritmo | Mejor Caso | Peor Caso | Caso Promedio |
| Búsqueda Binaria | O(1) | O(log n) | O(log n) |
| Bubble Sort | O(n) | O(n2) | O(n2) |
| Selection Sort | O(n²) | O(n2) | O(n2) |
| Insertion Sort | O(n) | O(n2) | O(n2) |
| Merge Sort | O(n log n) | O(n log n) | O(n log n) |
| Quick Sort | O(n log n) | O(n2) | O(n log n) |

# Resumen Complejidad BigO: