**Understand Sorting Algorithms:-**

**Bubble Sort**

**Description**: Bubble Sort is a simple comparison-based sorting algorithm where each pair of adjacent elements is compared, and the elements are swapped if they are in the wrong order. This process repeats until the array is sorted.

**Algorithm**:

1. Start at the beginning of the array.
2. Compare each pair of adjacent elements.
3. If the elements are in the wrong order, swap them.
4. Move to the next pair of elements.
5. Repeat the process until no swaps are needed.

**Time Complexity**:

* Best Case: O(n) (when the array is already sorted)
* Average Case: O(n²)
* Worst Case: O(n²)

**Space Complexity**: O(1) (in-place sorting)

**Pros**: Simple to understand and implement. **Cons**: Inefficient for large datasets due to its quadratic time complexity.

**Insertion Sort**

**Description**: Insertion Sort builds the sorted array one element at a time by repeatedly picking the next element and inserting it into its correct position among the previously sorted elements.

**Algorithm**:

1. Start with the second element (assume the first element is already sorted).
2. Compare the current element with the elements in the sorted portion.
3. Shift the sorted elements to the right to make space for the current element.
4. Insert the current element in the correct position.
5. Repeat until the entire array is sorted.

**Time Complexity**:

* Best Case: O(n) (when the array is already sorted)
* Average Case: O(n²)
* Worst Case: O(n²)

**Space Complexity**: O(1) (in-place sorting)

**Pros**: Simple to understand and implement, performs well with small or nearly sorted datasets. **Cons**: Inefficient for large datasets due to its quadratic time complexity.

**Quick Sort**

**Description**: Quick Sort is a divide-and-conquer algorithm that selects a 'pivot' element and partitions the array into two subarrays: elements less than the pivot and elements greater than the pivot. It then recursively sorts the subarrays.

**Algorithm**:

1. Choose a pivot element from the array.
2. Partition the array into two subarrays: elements less than the pivot and elements greater than the pivot.
3. Recursively apply the same steps to the subarrays.
4. Combine the sorted subarrays and the pivot to form the final sorted array.

**Time Complexity**:

* Best Case: O(n log n)
* Average Case: O(n log n)
* Worst Case: O(n²) (when the pivot is the smallest or largest element repeatedly)

**Space Complexity**: O(log n) (due to recursion stack)

**Pros**: Efficient for large datasets, average time complexity is O(n log n). **Cons**: Worst-case time complexity is O(n²), but this can be mitigated with good pivot selection strategies (e.g., random pivot).

**Merge Sort**

**Description**: Merge Sort is a divide-and-conquer algorithm that splits the array into two halves, recursively sorts each half, and then merges the sorted halves.

**Algorithm**:

1. Divide the array into two halves.
2. Recursively sort each half.
3. Merge the sorted halves to form the final sorted array.

**Time Complexity**:

* Best Case: O(n log n)
* Average Case: O(n log n)
* Worst Case: O(n log n)

**Space Complexity**: O(n) (requires additional space for the temporary arrays)

**Pros**: Stable sort (maintains the relative order of equal elements), consistently O(n log n) time complexity. **Cons**: Requires additional space, which can be a disadvantage for large datasets.

**Summary**

| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** | **Space Complexity** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- | --- | --- |
| Bubble Sort | O(n) | O(n²) | O(n²) | O(1) | Simple to implement | Inefficient for large datasets |
| Insertion Sort | O(n) | O(n²) | O(n²) | O(1) | Simple, good for small data | Inefficient for large datasets |
| Quick Sort | O(n log n) | O(n log n) | O(n²) | O(log n) | Efficient for large datasets | Worst case is O(n²) |
| Merge Sort | O(n log n) | O(n log n) | O(n log n) | O(n) | Stable, consistent O(n log n) | Requires additional space |

In practice, **Quick Sort** and **Merge Sort** are often preferred for their efficiency with larger datasets. Quick Sort is generally faster in practice but can degrade to O(n²) in the worst case, which can be mitigated with good pivot selection. Merge Sort is stable and has consistent performance but requires additional memory. **Bubble Sort** and **Insertion Sort** are simpler and can be useful for smaller datasets or nearly sorted data but are not efficient for larger datasets.