1. **Bubble Sort**:
   * Description: Bubble Sort compares adjacent elements and swaps them if they are in the wrong order. This process is repeated until the entire array is sorted.
   * Time Complexity:
     + Best Case: O(n) (when the array is already sorted, with optimization)
     + Average Case: O(n^2) (requires multiple passes to sort)
     + Worst Case: O(n^2) (when the array is in reverse order)
   * Pros: Simple to implement.
   * Cons: Inefficient for large datasets.
2. **Insertion Sort**:
   * Description: Insertion Sort builds the sorted array one item at a time by inserting each new item into its correct position among the previously sorted items.
   * Time Complexity:
     + Best Case: O(n) (when the array is already sorted)
     + Average Case: O(n^2) (when the array is randomly ordered)
     + Worst Case: O(n^2) (when the array is in reverse order)
   * Pros: Efficient for small datasets or nearly sorted data.
   * Cons: Inefficient for large datasets.
3. **Quick Sort**:
   * Description: Quick Sort divides the array into smaller sub-arrays based on a pivot element, sorting the sub-arrays independently. The process is recursive.
   * Time Complexity:
     + Best Case: O(n log n) (when the pivot divides the array evenly)
     + Average Case: O(n log n) (average case performance)
     + Worst Case: O(n^2) (when the pivot is the smallest or largest element, which is rare with good pivot selection)
   * Pros: Very efficient for large datasets; generally faster than other O(n log n) algorithms.
   * Cons: Performance can degrade if not implemented with a good pivot selection strategy.
4. **Merge Sort**:
   * Description: Merge Sort divides the array into halves, sorts each half, and then merges the sorted halves.
   * Time Complexity:
     + Best Case: O(n log n)
     + Average Case: O(n log n)
     + Worst Case: O(n log n)
   * Pros: Consistently efficient; stable sort.
   * Cons: Requires additional space proportional to the array size.

**Why Quick Sort is Preferred**:

* Efficiency: Quick Sort has a better average-case time complexity (O(n log n)) compared to Bubble Sort’s O(n^2). This makes it much more efficient for large datasets.
* Performance: In practice, Quick Sort often performs faster than Bubble Sort because it makes fewer comparisons and swaps.