**1. Understanding Algorithms**

**1. Bubble Sort**:

**Definition:**

Compare adjacent elements and swap them if they are in the wrong order. Repeat this process until the list is sorted.

**Time Complexity**:

* + **Best Case**: O(n) with optimization.
  + **Average Case**: O(n²).
  + **Worst Case**: O(n²).

**2. Insertion Sort**:

**Definition**:

Build a sorted portion of the list one element at a time. Take each new element and insert it into its correct position within the sorted portion.

**Time Complexity**:

* + **Best Case**: O(n) – When the list is already sorted.
  + **Average Case**: O(n²) – When the list is unsorted.
  + **Worst Case**: O(n²) – When the list is in reverse order.

**3. Quick Sort**:

**Definition**:

Select a pivot element from the list, partition the other elements into two sub-arrays according to whether they are less than or greater than the pivot. Recursively apply the same process to the sub-arrays.

**Time Complexity**:

* + **Best Case**: O(n log n).
  + **Average Case**: O(n log n).
  + **Worst Case**: O(n²).

**4. Merge Sort**:

**Definition**:

Divide the list into halves, sort each half recursively, and then merge the sorted halves back together.

**Time Complexity**:

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

**2. Analysis**

**Bubble Sort**:

* **Time Complexity**:
  + **Best Case**: O(n) – When the list is already sorted, and we use an optimized version that detects no swaps are needed.
  + **Average Case**: O(n²) – When the list is unsorted, and it has to perform many swaps and comparisons.
  + **Worst Case**: O(n²) – When the list is sorted in reverse order.

**Quick Sort**:

* **Time Complexity**:
  + **Best Case**: O(n log n) – When the pivot divides the list into two nearly equal halves.
  + **Average Case**: O(n log n) – Generally, Quick Sort performs well on average.
  + **Worst Case**: O(n²) – When the pivot is the smallest or largest element, leading to highly unbalanced partitions (e.g., if the list is already sorted).

**Why Quick Sort is Generally Preferred Over Bubble Sort**

* **Efficiency**:
  + **Quick Sort** is much faster on average due to its O(n log n) time complexity compared to Bubble Sort’s O(n²). This makes Quick Sort suitable for larger datasets.
  + **Bubble Sort** is simple but inefficient for large lists because its time complexity grows quadratically with the size of the input.
* **Performance**:
  + Quick Sort generally outperforms Bubble Sort because it reduces the problem size more rapidly by dividing the list into smaller partitions.