**Experiment No 9**

**Name: Veeransh Shah**

**Reg No.: 221070063**

**AIM:**The aim of this experiment is to implement and execute the Insertion Sort and Merge Sort algorithms and subsequently analyze their time complexity.

**THEORY:**

**Insertion Sort:**

Algorithm Overview:

Insertion Sort is a simple sorting algorithm that builds the final sorted array one element at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. However, insertion sort provides several advantages in its simplicity and ease of implementation.

Time Complexity Analysis:

Worst-Case Time Complexity (O(n^2)): The worst-case time complexity occurs when the array is in reverse order. In each iteration, the algorithm compares and shifts elements, leading to quadratic time complexity.

Best-Case Time Complexity (O(n)): The best-case time complexity occurs when the array is already sorted. In this scenario, insertion sort makes minimal comparisons, resulting in linear time complexity.

Average-Case Time Complexity (O(n^2)): On average, insertion sort exhibits quadratic time complexity, making it less suitable for large datasets.

**Merge Sort:**

Algorithm Overview:

Merge Sort is a divide-and-conquer algorithm that recursively divides the input array into two halves, sorts each half, and then merges the sorted halves. It ensures stability and guarantees a time complexity of O(n log n).

Time Complexity Analysis:

Worst-Case Time Complexity (O(n log n)): The worst-case time complexity of merge sort is O(n log n), making it efficient for large datasets. The algorithm consistently divides the array into halves until individual elements, and then merges them in a sorted manner.

Best-Case Time Complexity (O(n log n)): The best-case time complexity is also O(n log n). Unlike other algorithms, merge sort's performance is consistent across different scenarios.

Average-Case Time Complexity (O(n log n)): Merge sort maintains its O(n log n) time complexity on average, making it a reliable choice for sorting.

**ALGORITHM:**

**1.Insertion Sort:**

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.

Step2 - Pick the next element, and store it separately in a key.

Step3 - Now, compare the key with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.

Step 5 - Insert the value.

Step 6 - Repeat until the array is sorted.

**2,Merge Sort:**

Step1:If the array has one element or is empty, it is inherently sorted. Return the array.

Step 2: Split the unsorted array into two equal halves. Determine the middle point of the array.

Step 3: Recursively apply Merge Sort to the left and right halves. For the left half, repeat steps 1-3.For the right half, repeat steps 1-3.

Step4: Create a temporary array to store the merged result. Initialize pointers for the left and right halves. Compare elements from the left and right halves, and merge them into the temporary array in ascending order. Move the pointers accordingly.

Step 5: If there are remaining elements in the left or right halves, copy them into the temporary array.

Step 6: Copy the sorted elements from the temporary array back into the original array.

Step 7: Repeat steps 2-6 for each recursive call.

**CONCLUSION:**

In conclusion, while both Insertion Sort and Merge Sort aim to sort arrays, their time complexity characteristics differ significantly. Insertion Sort, with its quadratic time complexity, is suitable for small datasets, while Merge Sort, with its consistent O(n log n) time complexity, excels in handling larger datasets. The empirical analysis of their time complexity through experimental execution will provide valuable insights into their practical performance characteristics.

ScreenShot:

