Sorting Techniques:

1. Bubble Sort: Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

Time Complexity: O(N2)

Auxiliary Space: O(1)

Best case complexity is O(N)

Worst case time complexity is O(N2)

2. Insertion Sort: The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed in the correct position in the sorted part.

Time Complexity: O(N^2)

Auxiliary Space: O(1)

The worst-case time complexity of the Insertion sort is O(N^2)

The average case time complexity of the Insertion sort is O(N^2)

The time complexity of the best case is O(N).

3. Selection Sort: is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.

The time complexity of Selection Sort is O(N2)

Auxiliary Space: O(1)

Best-case: O(n2)

Average-case: O(n2),

Worst-case: O(n2)

4. Quick Sort: is a sorting algorithm based on the Divide and Conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

Time Complexity:

Best Case: Ω (N log (N))

Average Case: θ (N log (N))

Worst Case: O(N2)

Auxiliary Space: O(1)

5. Merge Sort: sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.

Time Complexity: O(N log(N))

The time complexity of Merge Sort isθ(Nlog(N)) in all 3 cases (worst, average, and best)

Auxiliary Space: O(N)

6. Heap Sort: where we first find the minimum element and place the minimum element at the beginning. Repeat the same process for the remaining elements.

Time Complexity: O(N log N)

Auxiliary Space: O(log n)

worst case will be O(n(log(n)).

best case would be O(n).

average runtime of O(n(log(n))

7. Counting Sort: is a non-comparison-based sorting algorithm.The basic idea behind Counting Sort is to count the frequency of each distinct element in the input array and use that information to place the elements in their correct sorted positions.

Time Complexity: O(N+M), where N and M are the size of inputArray[] and countArray[] respectively.

Worst-case: O(N+M).

Average-case: O(N+M).

Best-case: O(N+M).

Auxiliary Space: O(N+M), where N and M are the space taken by outputArray[] and countArray[] respectively.

8.Shell sort: is mainly a variation of Insertion Sort.

Time Complexity: Time complexity of the above implementation of Shell sort is O(n2).

The worst-case complexity for shell sort : O(n2)

best case complexity is: Ω(n log(n))

Average Case Complexity:θ(n log(n)2).

The space complexity of the shell sort is O(1).

9. Radix Sort:Radix Sort is a linear sorting algorithm that sorts elements by processing them digit by digit. It is an efficient sorting algorithm for integers or strings with fixed-size keys.

Time Complexity: O(d * (n + b)), where d is the number of digits, n is the number of elements, and b is the base of the number system being used.

Space Complexity: Radix sort also has a space complexity of O(n + b), where n is the number of elements and b is the base of the number system.

Worst case time complexity: O(logb(mx)(n+b)); Only one element which has significantly large number of digits

Best case time complexity: All elements have the same number of digits

Average case time complexity: O(d(n+b))