SECTION-IV

SORTING AND HASHING

Objective and properties of different sorting algorithms:

important properties of different sorting techniques including their complexity, stability and memory constraints. Before understanding this article, you should understand basics of different sorting techniques (See : Sorting Techniques).

Time complexity Analysis –

We have discussed the best, average and worst case complexity of different sorting techniques with possible scenarios.

Comparison based sorting –

In comparison based sorting, elements of an array are compared with each other to find the sorted array.

- **Bubble sort and Insertion sort –**
  Average and worst case time complexity: n^2
  Best case time complexity: n when array is already sorted.
  Worst case: when the array is reverse sorted.

- **Selection sort –**
  Best, average and worst case time complexity: n^2 which is independent of distribution of data.

- **Merge sort –**
  Best, average and worst case time complexity: nlogn which is independent of distribution of data.

- **Heap sort –**
  Best, average and worst case time complexity: nlogn which is independent of distribution of data.

- **Quick sort –**
  It is a divide and conquer approach with recurrence relation:

$T(n) = T(k) + T(n-k-1) + cn$

- Worst case: when the array is sorted or reverse sorted, the partition algorithm divides the array in two subarrays with 0 and n-1 elements. Therefore,

$T(n) = T(0) + T(n-1) + cn$

Solving this we get, $T(n) = O(n^2)$

- Best case and Average case: On an average, the partition algorithm divides the array in two subarrays with equal size. Therefore,

$T(n) = 2T(n/2) + cn$

Solving this we get, $T(n) = O(nlogn)$

Non-comparison based sorting –
In non-comparison based sorting, elements of array are not compared with each other to find the sorted array.

- **Radix sort –**
  Best, average and worst case time complexity: nk where k is the maximum number of digits in elements of array.

- **Count sort –**
  Best, average and worst case time complexity: n+k where k is the size of count array.

- **Bucket sort –**
  Best and average time complexity: n+k where k is the number of buckets.
  Worst case time complexity: n^2 if all elements belong to same bucket.

In-place/Outplace technique –
A sorting technique is inplace if it does not use any extra memory to sort the array.
Among the comparison based techniques discussed, only merge sort is outplaced technique as it requires an extra array to merge the sorted subarrays.
Among the non-comparison based techniques discussed, all are outplaced techniques. Counting sort uses a counting array and bucket sort uses a hash table for sorting the array.
Online/Offline technique –
A sorting technique is considered Online if it can accept new data while the procedure is ongoing i.e. complete data is not required to start the sorting operation.
Among the comparison based techniques discussed, only Insertion Sort qualifies for this because of the underlying

algorithm it uses i.e. it processes the array (not just elements) from left to right and if new elements are added to the right, it doesn't impact the ongoing operation.

Stable/Unstable technique –

A sorting technique is stable if it does not change the order of elements with the same value.

Out of comparison based techniques, bubble sort, insertion sort and merge sort are stable techniques. Selection sort is unstable as it may change the order of elements with the same value. For example, consider the array 4, 4, 1, 3.

In the first iteration, the minimum element found is 1 and it is swapped with 4 at 0th position. Therefore, the order of 4 with respect to 4 at the 1st position will change. Similarly, quick sort and heap sort are also unstable.

Out of non-comparison based techniques, Counting sort and Bucket sort are stable sorting techniques whereas radix sort stability depends on the underlying algorithm used for sorting.

Analysis of sorting techniques :

- When the array is almost sorted, insertion sort can be preferred.
- When order of input is not known, merge sort is preferred as it has worst case time complexity of nlogn and it is stable as well.
- When the array is sorted, insertion and bubble sort gives complexity of n but quick sort gives complexity of n^2.

Que – 1. Which sorting algorithm will take the least time when all elements of input array are identical? Consider typical implementations of sorting algorithms.

(A) Insertion Sort
(B) Heap Sort
(C) Merge Sort
(D) Selection Sort

Solution: As discussed, insertion sort will have the complexity of n when the input array is already sorted.

Que – 2. Consider the Quicksort algorithm. Suppose there is a procedure for finding a pivot element which splits the list into two sub-lists each of which contains at least one-fifth of the elements. Let T(n) be the number of comparisons required to sort n elements. Then, (GATE-CS-2012)

(A) $T(n) <= 2T(n/5) + n$
(B) $T(n) <= T(n/5) + T(4n/5) + n$
(C) $T(n) <= 2T(4n/5) + n$
(D) $T(n) <= 2T(n/2) + n$

Solution: The complexity of quick sort can be written as:

$T(n) = T(k) + T(n-k-1) + cn$

As given in question, one list contains 1/5th of total elements. Therefore, another list will have 4/5 of total elements. Putting values, we get:
$T(n) = T(n/5) + T(4n/5) + cn$, which matches option (B).

Time and Space Complexity Comparison Table :

Sorting Algorithm	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case

Bubble Sort	**Ω(N)**	**Θ(N2)**	**O(N2)**	**O(1)**
Selection Sort	**Ω(N2)**	**Θ(N2)**	**O(N2)**	**O(1)**
Insertion Sort	**Ω(N)**	**Θ(N2)**	**O(N2)**	**O(1)**
Merge Sort	**Ω(N log N)**	**Θ(N log N)**	**O(N log N)**	**O(N)**
Heap Sort	**Ω(N log N)**	**Θ(N log N)**	**O(N log N)**	**O(1)**
Quick Sort	**Ω(N log N)**	**Θ(N log N)**	**O(N2)**	**O(log N)**
Radix Sort	**Ω(N k)**	**Θ(N k)**	**O(N k)**	**O(N + k)**