iii] Analysis of Quick Sort

a) Quick Sort

- This algorithm follows divide and conquer approach.

Divide : Partition the array A[p..r] into two subarrays A[p....q-1] and A[q+1...r] such that each element of A[p....q-1] is less than or equal to A[q] which in turn is less than or equal to each element of A[q+1...r].

Computing index q is the job of partition procedure.

Conquer: Sort the two subarrays A[p..q-1] and A[q+1..r] by recursive calls to quicksort.

Combine : Since array is already sorted, no work is required in combine step.

Algorithm :

i) QUICKSORT Procedure

I/p : A, p, r , A is array, p and r are start & end indexes.

O/p : Sorted sequence A[p...r]

```
QUICKSORT (A, p, r)
{
1.    if (p<r)
      {
2.        q = PARTITION (A, p, r) ;
3.        QUICKSORT (A, p, q-1) ;
4.        QUICKSORT (A, q+1, r)
      }
}
```

To sort an entire array A, initial call is QUICKSORT (A, 1, A.length).

(ii) PARTITION procedure

i/p : A - original array
p, r - indices between which array is rearranged.

o/p : q - index where array would be partition further.

purpose: It rearranges the subarray A[p ... r] in place.

```
PARTITION (A, p, r)
{
1.  x = A[r]                    // last element as pivot
2.  i = p-1
3.  for j = p to r-1
    {
4.      if A[j] ≤ x
        {
5.          i = i+1
6.          A[i] ↔ A[j]
        }
    }
7.  A[i+1] ↔ A[r]               // placing pivot to its correct position.
8.  return i+1
```

(b) Analysis of Quick Sort

- The running time of quicksort depends on whether the partitioning is balanced or unbalanced, which in turn depends on which elements (i.e pivot) are used for partitioning.
- If the partitioning is balanced, then algorithm runs as fast as merge sort : i.e $\Theta(n \log n)$
- If the partitioning is unbalanced, then algorithm runs as slow as insertion sort i.e $\Theta(n^2)$

(2) Worst-Case Partitioning

Conquer - In this, problem is divided into 2 subproblems of size $(n-1)$ and another subproblem with zero (0) element.

Divide - Partition procedure takes $\Theta(n)$ times.

∴ Recurrence for running time is,

$$T(n) = T(n-1) + T(0) + \Theta(n)$$
$$= T(n-1) + \Theta(n)$$

Using Substitution method,

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

Step 1: Guess the solution

$T(n) = O(n^2)$

Exact form

$$T(n) = T(n-1) + n$$
$$T(1) = 1$$
$$T(2) = T(1) + 2 = 1+2$$
$$T(3) = T(2) + 3 = 1+2+3$$

$$\therefore \boxed{T(n) = \frac{n(n+1)}{2}}$$

Step 2: Mathematical Induction

a) To prove: $T(n) \leq c\left[\frac{n(n+1)}{2}\right]$ ——(1)

b) Finding c:

Assumption bound holds for $T(n-1)$

$$\therefore T(n-1) \leq c\left[\frac{(n-1)n}{2}\right] \quad \text{---- (2)}$$

using equation (1) in recurrence equation,

$$T(n) = T(n-1) + n$$
$$= c\left[\frac{n(n-1)}{2}\right] + n$$
$$= \frac{cn^2}{2} - \frac{c}{2}n + n$$

We need to prove,

$$T(n) \leq c\left[\frac{n(n+1)}{2}\right]$$

$$\frac{cn^2}{2} - \frac{c}{2}n + n \leq \frac{c}{2}n^2 + \frac{c}{2}n$$

$$n \leq cn$$

dividing by n,

$$\boxed{c \geq 1}$$

c) Prove the base condition

put $n = 1$ in equation 1

$$T(1) \leq c\left(\frac{n(n+1)}{2}\right)$$

$$\leq c$$

$$\leq 1 \qquad [c = 1]$$

which is satisfied.

$\therefore T(n) = O(n^2)$ for $c = 1$, $n_0 = 1$

(ii) Best Case Partitioning

Divide - Partition procedure takes $\Theta(n)$ times.

Conquer - Problem is divided into 2 subproblems of size $\lfloor n/2 \rfloor$ and $\lceil \frac{n}{2} \rceil - 1$.

∴ Recurrence for the running time is,

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

using Master Theorem,

Step 1: Comparing with standard form

$a = 2 \quad b = 2 \quad f(n) = n$.

$$\log_b a = \log_2 2 = 1$$
$$n^{\log_b a} = n^1 = n$$

Step 2: Identifying case

$$f(n) = n$$
$$= n^{\log_b a}$$
$$= \Theta(n^{\log_b a})$$

By case 2,

$$T(n) = \Theta(n^{\log_b a} \log n)$$

$$\boxed{T(n) = \Theta(n \log n)}$$

(iii) Balanced Partitioning

The average case running time of quicksort is much closer to the best case than the worst case.

Example

A = {5, 4, 6, 1, 3, 2}