Q3) What is the running time of Quick sort when all elements of array A have the same value?

Ans) The running time of Quick sort when all elements of array A have the same value will be equivalent to the worst running of Quick sort since no matter what point pivot is picked, Quick sort will have to go through all the values in A. And since all values are the same, each recursive call will lead to unbalanced partitioning.

Thus the recurrence will be

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

The above recurrence has the solution (using substitutio

$$T(n) = T(n-1) + O(n)$$
$$T(n-1) = T(n-2) + O(n-1)$$
$$T(n-2) = T(n-3) + O(n-2)$$
$$\vdots \qquad \vdots \qquad \vdots$$
$$T(1) = T(0) + O(1)$$

Adding all the above eq^{ns}, we'll get

$$T(n) = O(n) + O(n-1) + O(n-2) + \cdots + O(1)$$
$$= O(n(n+1)/2) = O(n^2).$$

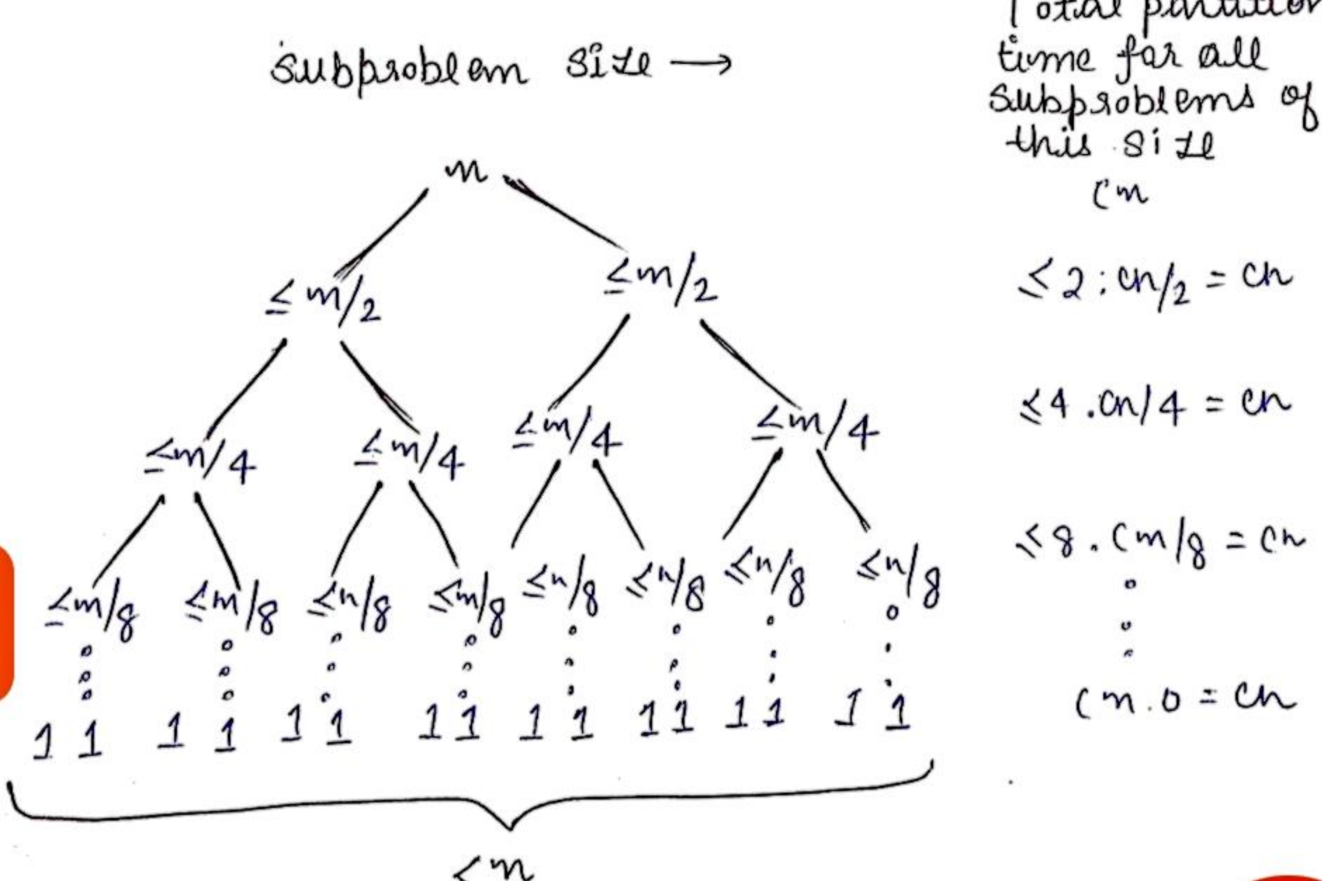
$$\therefore T(n) = O(n^2)$$

Hence the running time of quick sort in this case is

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

Q4) Show that quicksorts best case running time is $O(n \log n)$ with example.

Ans) Quicksort's best case occurs when the partitions are as evenly balanced as possible: their sizes either are equal or are within 1 of each other. The former case occurs if the subarray has an odd number of elements and the pivot its right in the middle after partitioning and each partition has $(n-1)/2$ elements. The letter case occurs if the subarray has an even number n of elements and one partition has $n/2$ elements with the other having $n/2 - 1$, In either of these cases, each partition has atmost $n/2$ elements, and the tree of subproblem sizes looks a lot like tree of subproblem sizes for merge sort, with the partitioning times looking like the merging times:



Q2) Write the recurrence relation for Binary search, merge sort and quicksort.

Ans) • for Binary Search:

Input: Sorted Array A of size n, an element x to be searched.

Approach: check whether $A[n/2] = x$. if $x > A[n/2]$ then prune the lower half of the array. $A[1, \dots, n/2]$ Otherwise, prune the upper half array. Therefore, Pruning happens at every iterations. After each iteration the problem size (array size under consideration) reduces by half.

Recurrence relation:

$$T(n) = T(n/2) + 1$$

1^{st} step: $T(n) = T(n/2) + 1$

2^{nd} step: $T(n/2) = T(n/4) + 1$

3^{rd} step: $T(n/4) = T(n/8) + 1$

4^{th} step: $T(n/8) = T(n/16) + 1$

⋮

K^{th} step: $T(n/2^{K-1}) = T(n/2^K) + 1$

By Adding all the equations we get,

$$T(n) = T(n/2^K) + K(1) \text{ ----- final eq}^n$$

$\Rightarrow n/2^K = 1 \Rightarrow n = 2^K.$

$\Rightarrow K = \log_2 n.$

Put $K = \log(n)$ in final eq^n

$$T(n) = T(1) + \log(n)$$

$$T(n) = O(\log n) \quad \{(\text{Taking dominant Polynomial})\}$$

- For Merge Sort:

Approach: Divide the array into two equal sub arrays and sort each sub array recursively. Do the subdivision operation Recursively till the array size becomes one. Trivially, the problem size and one is sorted when the recursion Bottom out two sub problems. of size one are combined to get a sorting sequence of size two further two sub problems of size two (each one is sorted) are combined to get a sorting sequence of size four, and soon. We shall see the detailed description below. To combine 2 sorted arrays of size $n/2$ each, we need $n/2 + n/2 + 1 = n - 1$ comparisons in the worst case.

Recurrence Relation:

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

By using substitution method:

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

$$= 2[2T(n/2^2) + n/2 - 1] + n - 1$$

$$\Rightarrow 2^k T(n/2^k) + n - 2^k + n - 2^{k+1} + n - 1.$$

when $n = 2^k$, $T(n) = 2^k T(1) + n + \text{-----} + n - [2^{k-1} + \text{--} + 2^0]$

Note that:

$$2^{k-1} + \text{-----} + 2^0 = \frac{2^{k-1+1} - 1}{2 - 1} = 2^k - 1 = n - 1.$$

Also, $T(1) = 0$ as there is no comparison required if $n = 1$, Therefore, $T(n) = n\log_2(n) - n + 1$

$$= O(n\log_2 n).$$

$$T(n) = O(n\log_2 n)$$

• For Quick sort:

Approach: like merge sort, Quicksort is a Divide and conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quicksort that pick pivot in different ways.

(1) Always pick first element as pivot.
(2) Always pick last element as pivot.
(3) pick a random element as pivot.
(4) Pick median as pivot.

The key process in quick sort is partition. Target of partitions is given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Recurrence Relation:

$$T(n) = 2T(n/2) + n$$
$$T(0) = T(1) = 0 \text{ (base case)}$$

using substitution method

$$T(n) = 2T(n/2) + n$$
$$\Rightarrow \left(T(n)/n\right) = 1 + \left(T(n/2)/T(n/2)\right)$$
$$\Rightarrow \left(T(n/2)/n/2\right) = 1 + \left(T(n/4)/(n/4)\right)$$
$$\ldots \qquad \ldots$$
$$\Rightarrow \left(\frac{T(n/(n/2))}{T(n/n/2)}\right) = 1 + \left(\frac{T(n/n)}{(n/n)}\right) = 1 + \frac{T(1)}{(1)}$$

same as $\frac{T(2)}{(2)} = 1 + \frac{T(1)}{(1)} \Rightarrow \frac{T(n)}{n} = 1 + 1 + 1 - \cdots \log(n)$ times

$$T(n)/n = \log(n)$$