**COMSATS Institute of Information Technology, Islamabad Campus**

**Department of Computer Science**

**Design and Analysis of Algorithms – CSC301**

**BCS – IVA | Quiz #2**



**FA19-BCS-037**

1. **Compare worst case time complexities of Merge and quick sort. (5)**

**Worst Case Time Complexity:**

**Merge Sort:** nlogn

**Justification:** It follows a divide and conquer algorithm in which at every stage an array is split in two halves and sorting is initiated from the lowest level. It does nlogn comparisons in the worst case.

Input array arr[] = [4,0,6,2,5,1,7,3]

/ \

/ \

[4,0,6,2] and [5,1,7,3]

/ \ / \

/ \ / \

[4,0] [6,2] [5,1] [7,3]

| | | |

| | | |

[0,4] [2,6] [1,5] [3,7]

\ / \ /

\ / \ /

[0,2,4,6] [1,3,5,7]

\ /

\ /

[0,1,2,3,4,5,6,7]

**Quick Sort:** n2

**Justification:** The array is not necessarily divided in half because it completely depends on the pivot which is used for partitioning. One part contains elements smaller than the pivot and the other part has elements larger than the pivot and the elements are sorted recursively. However, it takes n2 comparisons in the worst case due to the maximum number of unbalanced partitions.

Subproblem sizes Partitioning time

N cn

/ \

/ \

0 n-1 c(n-1)

/ \

/ \

0 n-2 c(n-2)

/ \

/ \

1. n-3 c(n-3)

\

\

2 2c

/ \

/ \

0 1 0

**2) Best Case of Quick Sort:**

**Consider a scenario where we get balanced Partitions in all cases**.

The sizes are either equal to each other or differ by a maximum of one difference in size i.e.,

* As Quick Sort Heavily Depends on Pivot Selection so to optimize our time complexity, we have two options to achieve best-case i.e.
  + Selecting the Middle element as pivot.
  + Selecting a random value for pivot
* If the array has the size of the odd number of elements. As the Quicksort algorithm depends on Pivot Selection, we can deduce that if we partition the array right in the middle, each partition will have (n-1)/2 elements.
* If an even number of elements exist in an array, then the partition will be n/2 of one sub-array and n/2 -1 of the other sub-array.
* So, **the time complexity would be O (n log2 n)** as there are log n levels and n to divide each array into subproblem.

Subproblem Size

N

<=n/2 <=n/2

<=n/4 <=n/4 <=n/4 <=n/4

<=n/8 <=n/8 <=n/8 <=n/8 <=n/8 <=n/8 <=n/8 <=n/8

**. . . . . . . .**

**. . . . . . . .**

**. . . . . . . .**

1 1 1 1 1 1 1 1

<N

**Total Partitioning time for `all the subproblems of this size.**

**Cn**

**<=2.cn/2=cn**

Log n

**<=4.cn/4=cn**

**<=8.cn/8=cn**

**<n.c=cn**

**3) Solve any recurrence relation using recursion tree method. (5)**

# **Recursion Tree:**

**T(n) = T(n/2) + T(n/2) + O(n).**

**………. n**

**n(root)**

**n/2+n/2= n**

(n/2)

(n/2)

**n/4+n/4+n4+n/4=n**

(n/4)

(n/4)

(n/4)

(n/4)

n/2k

n/2k

**Height of Left and Right sub tree= n/2k**

**Let n/2k =1**

**N=2k**

**Taking log,**

**k = logn**

**T (n) = O (nlogn)**