**CSCD 304 DAA**

ID: **10661120**

**Q1.** Give the situations in which the quicksort algorithm is in the

1. Best case:   
   The best case occurs when the partition process always picks the middle element as pivot and also when the partitions of the array are as balanced as possible. That is, the partition is called on arrays of equal size or arrays whose sizes are 1 less than the other. The later occurs when we have an odd number of elements(n) and the pivot is greater than (n-1)/2 of those elements, whiles the former occurs when we have an even number of elements(n), and the pivot is greater than n/2 of the elements.

b) Worst case:  
The worst case occurs when the partition process always picks greatest or smallest element as pivot or when pivot of the array is either greater than or lesser than all the elements of the array. When this happens, we have to make calls on arrays of sizes 0 and (n-1).

**Q2.** Provide an analysis (in terms of time complexity) of the best and worst case scenarios of the quicksort algorithm.

Analysis of worst case scenario in terms of time complexity.  
As stated above, the quick sort algorithm is in its worst case scenario when the pivot is greater than or lesser than all its elements in this case, the algorithm will call recursively arrays of sizes (n-1) and 0. Suppose the time taken to call the partition for an array of size n is **c**n, where **c** is a constant. Then the time taken to run the worst case scenario of the algorithm is given by;

C*worst*(n) = **c**n + **c**(n-1) + … + 2**c** = **c**(n + (n-1) + … + 2)  
 = **c**∑ni = 2i  
 =**c**(n(n+1)/2-1)  
 ≈1/2n2 ϵ Θ(n2)  
Hence, in its worst case, the quicksort algorithm behaves like the selection sort algorithm.

Analysis of the best case scenario in terms of time complexity.  
The quick sort algorithm experiences its best case scenario when the algorithm is as balanced as possible. That is, the selected pivot is greater than almost half of the elements in the array. In this case, every call made is on an array of size n/2 or (n-1)/2. So the tree of sub problems looks like the tree of sub problems for merge sort. Consequently, we end up having a time complexity of Θ(nlog2n).

Diagram of best case performance for Quick Sort, with a tree on the left and partitioning times on the right. The tree is labeled "Subproblem size" and the right is labeled "Total partitioning time for all subproblems of this size."
The first level of the tree shows a single node n and corresponding partitioning time of c times n. The second level of the tree shows two nodes, each of less than or equal to 1/2 n, and a partitioning time less than or equal to 2 times c times 1/2 n, the same as c times n. The third level of the tree shows four nodes, each of less than or equal to 1/4 n, and a partitioning time less than or equal to 4 times c times 1/4 n, the same as c times n. The fourth level of the tree shows eight nodes, each of less than ot equal to 1/8 n, and a partitioning time less than or equal to 8 times c times 1/8 n, the same as c times n. Underneath that level, dots are shown to indicate the tree continues like that. A final level is shown with n nodes of 1, and a partitioning time of less than or equal to n times c, the same as c times n.