Algorithm Analysis
Lab 1 : Implementation of Randomized Quick Sort

AIM : To implement Randomized Quick Sort algorithm and compare its performance with the traditional fixed pivot quick sort.

Description

A randomized algorithm is an algorithm that receives, in addition to its input, a stream of random bits which is used to make random choices. The random bits are assumed to be independent of the input. A salient feature is that repeated runs of a randomized algorithm with fixed input data will, in general, not produce the same result. While randomized approach lacks definiteness, it allows to transform deterministic algorithms with bad worst case behaviour into randomized algorithms that perform well with high probability on any input. There are basically two flavours of randomized algorithms, LasVegas and MonteCarlo. LasVegas algorithms always produces correct answer, but it's running time varies for different run. On the other hand, MonteCarlo algorithm has usually fixed running time, but it may produce incorrect output with certain probability. In this laboratory we will implement following randomized algorithms.

Randomized Quick Sort Algorithm

Although the 'traditional' Quicksort algorithm is very efficient in practice, its worst-case running time is rather slow. When sorting n elements, the number of comparisons may be O(n2). The worst case happens if the sizes of the sub-problems are not balanced. The selection of the pivot element determines the sizes of the subproblems. It is thus crucial to select a 'good pivot'.

Algorithm 1 Algo RandomizedQuickSort (A, p, q)

```
if p ≥ q then
        return
else
        r = random select(p, p + 1, ...., q)
        swap(A[p], A[r])
        j = AlgoPartition(A, p, q)
        Algo RandomizedQuickSort(A, p, j − 1)
Algo RandomizedQuickSort(A, j + 1, q)
end if
```

While in traditional Quick Sort algorithm the first or last element is chosen as pivot, the randomized algorithm tries to avoid the worst case behaviour by randomly choosing pivot. Algorithm 1 describes the outline or randomized quick sort algorithm. Similarly to the general Quick Sort algorithm, it partitions the input array A into two partitions and makes two recursive calls to sort them. The difference here, as shown in step 4 and 5 of the algorithm, is in the choice of the pivot. Instead of choosing a fixed pivot, the algorithm select r at random for the index range p to q. The algorithm swaps the selected element at index r with the first element p and the rest of the algorithm works as the traditional Quick Sort.

Algorithm 2 AlgoPartition(A, p, q)

```
for i = (p+1) to q do
        if A[i] < A[p] then
                insert A[i] into bucket L
        else if A[i] > A[p] then
                insert A[i] into bucket U
        end if
end for
copy A[p] into A[|L| + 1]
copy the elements of L into the first |L| entries of A[p..q]
copy the elements of U into the entries of A[(|L| + 2)...q]
return (|L| + 1)
```

The subroutine AlgoPartition, as described in Algorithm 2 distributes the elements of the array around the pivot. All elements which are smaller than the pivot go in the bucket L and larger than the pivot go in bucket U.

Exercise

(A) Run Randomized Quick Sort algorithm multiple time for the following data.

(i) [4, 3, 6, 8, 5, 9, 20, 50, 12, 30, 432, 12, 69, 29, 40, 50, 10, 4, 23, 34, 45, 45, 24, 89, 799, 45, 80, 50, 10, 30, 43,54,65,76,87, 98, 21, 32, 43, 78, 98, 57, 29, 91, 34, 54, 64, 19, 82, 87, 65]

Does the algorithm produce same output in every run? Does the algorithm take the same time in every run? (To calculate running time you may count the number of time swapping operation carried out.)

(B) Randomized Quick Sort is a _____ (Las Vegas / Monte Carlo) algorithm.

(C) What is the purpose of using randomized quick sort over standard quick sort?

1. *so as to avoid worst case time complexity*
2. *so as to avoid worst case space complexity*
3. *to improve accuracy of output*
4. *to improve average case time complexity*

(D) State true or false and justify:

1. *Maximum Number of Comparisons for the Randomized Quicksort is O(n^2)*
2. *Expected(Average) Number of Comparisons for the Randomized Quicksort is O(nlogn)*
3. *If two persons A, B run the same code of Randomized Quicksort over the same input, then A and B may get different counts of comparisons.*
4. *Output of Randomized Quicksort is always correct.*