Lab:-1 :- Implementation of Randomized Quick sort.

* Aim :- To implement Randomized Quick sort algorithm & compare it's performace with the traditional fixed quick sort.

* Program:-

```
import random.

global count_var
count_var = 0

def Randomized_QuickSort (a, p, r):
    # a = array; p = startindex ; r = endindex.
    q = Randomized_Partision (a, p, r)
    Randomized Quick sort (a, p, q-1)
    Randomized_Quick_Sort (a, q+1, r)
    pass

def Randomized_Partision (a, p, r):
    i = random.randint ( p, r)
    a[i], a[r] = a[r], a[i] # swap.
    return Partition (a, p, r)

def Partition (array, low, high):
    pivot = array [high]
    i = low-1
```

```
    for j in range( low, high):
        global count_var
        count_var += 1
        if array[j] <= pivot:
            i = i+1
            array[i], array[j] = array[j], array[i]
    array[i+1], array[high] = array[high], array[i+1]
    return i+1

→ if __name__ == "__main__":
    l = [9, 7, 5, 6, 48, 12, 74, 3, 12, 14, 36, 56, 74, 96]
    RandomizedQuickSort(l, 0, len(l)-1)
    print("No. of comparision : ", count_var)
    print(l)
    ## end
```

* Input / Output :-

I/P :- Unsorted array

O/P :- Always sorted array

* Time complexity :- O(nlogn)

* Exercise :-

(A) Yes, algorithm produced same output in every run.

→ No, it doesn't takes same Run time in each case (not taking same no. of steps i.e. count_var is different)

(B) Las Vegas.

(C) To avoid worst case time complexity

(D) 1)

2)

3) True

→ It is because the random function generator is different for both person A & B. i.e. random number generated for both person may be different, it also depends on programming language used & compiler used.

4) True

→ It is las Vegas type of algorithm so.