	PUNE INSTITUTE OF COMPUTER TECHNOLOGY PUNE - 411043	
	Department of Electronics & Telecommunication	
	ASSESMENT YEAR: 2024-2025	CLASS: SE
	SUBJECT: DATA STRUCTURES	
EXPT No:	LAB Ref: SE/2024-25/	Starting date:
	Roll No: 22203	Submission date:
Title:	Demonstrate Quick Sort Virtual lab	
Problem Statement	Working of Quick sort using virtual lab	
Refer lab manual for below		
Prerequisites:		
Objectives:		
Theory:		

Write algorithm and besides it example for each, attach both side ruled pages in case required

What is Quick Sort?

Quick Sort is one of the sorting algorithm in data structures which uses Divide and Conquer strategy (it divides the given array into two smaller sub-arrays and sort them individually) in a recursive way.

In this experiment whenever we mention sorting it means sorting in ascending order.

In quicksort we select an element in the array and keep that element in it's sorted position. We call this element as 'Pivot'. For an element in an array to be in it's sorted position the array should satisfy the condition that all elements less than that element should be in it's left side of the array and all elements greater than that element should be in it's right side of the array.

Partition1 ≤ Pivot ≥ Partition2

After doing this, we will recursively do the same thing for all elements in the left part and all elements in the right part individually(since elements in the left part won't cross the position of pivot after sorting as they are less than pivot,similarly for

elements in right side won't cross the position of pivot after sorting as they are greater than pivot) thus reducing the no. of elements we have to check in further steps.

Steps to Sort an Unsorted Array with Quick Sort Algorithm

- **Pivot Selection and Partition** : Selects an element for partitioning around it and partition the array into two parts such that left part of the pivot contains elements less than pivot and right part contains elements greater than pivot.
- **Recursion in Quick Sort** : Recursively executes above step on both of the partitions(left,right) inividually till they get sorted.
- **Concatenation of the Sub-Arrays** : Concatenate all the sub-arrays according to their indices

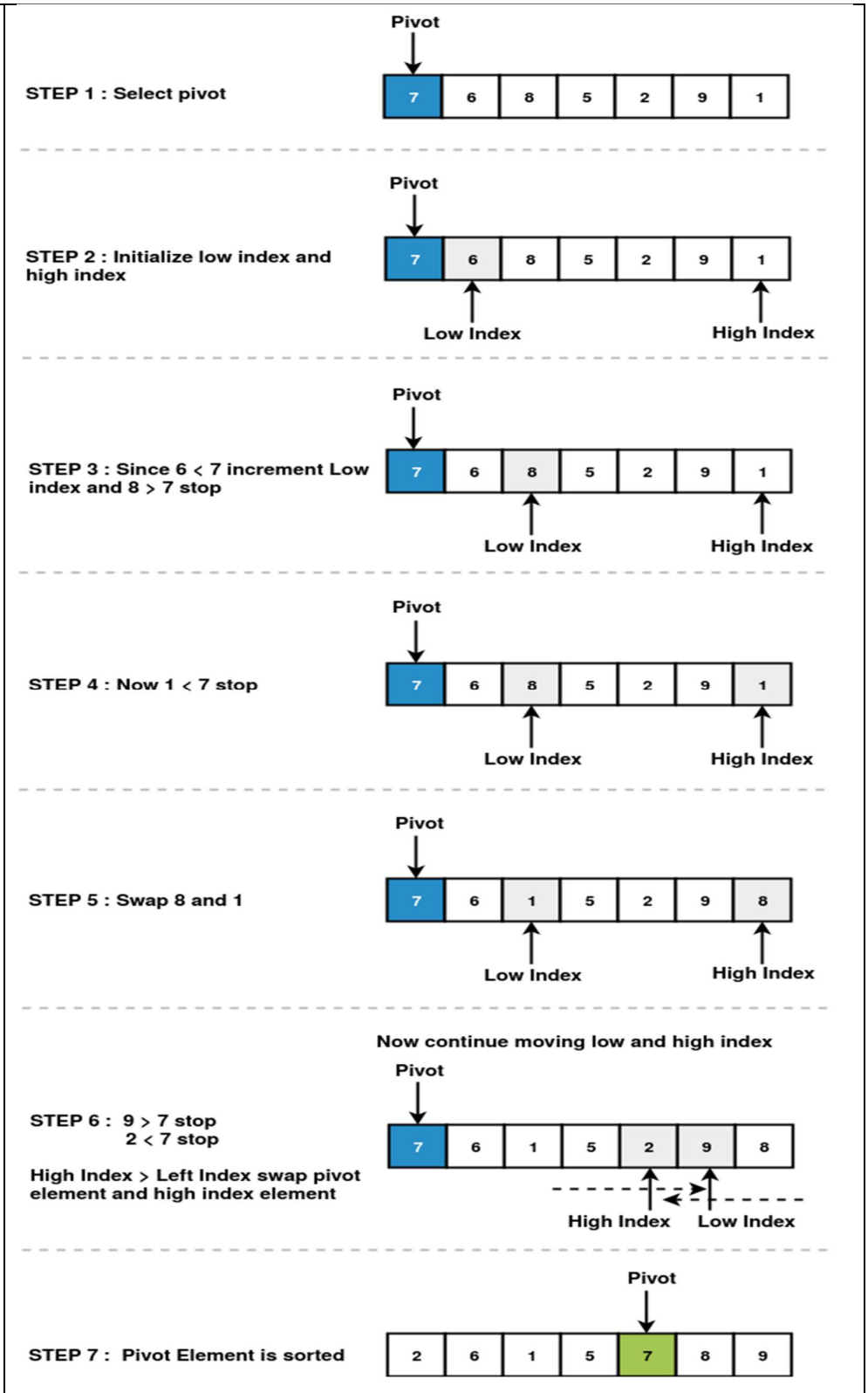
How to Select Pivot?

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 Quick Sort that pick pivot in different ways :

- Always pick first element as pivot
- Always pick last element as pivot
- Pick a random element as pivot
- Pick median as pivot

Pictorial Representation of Pivot Selection



The key process in Quick Sort is partition. Targets 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 and equal elements

(smaller than or equal to x) before x, and put all greater elements (greater than x) after x.

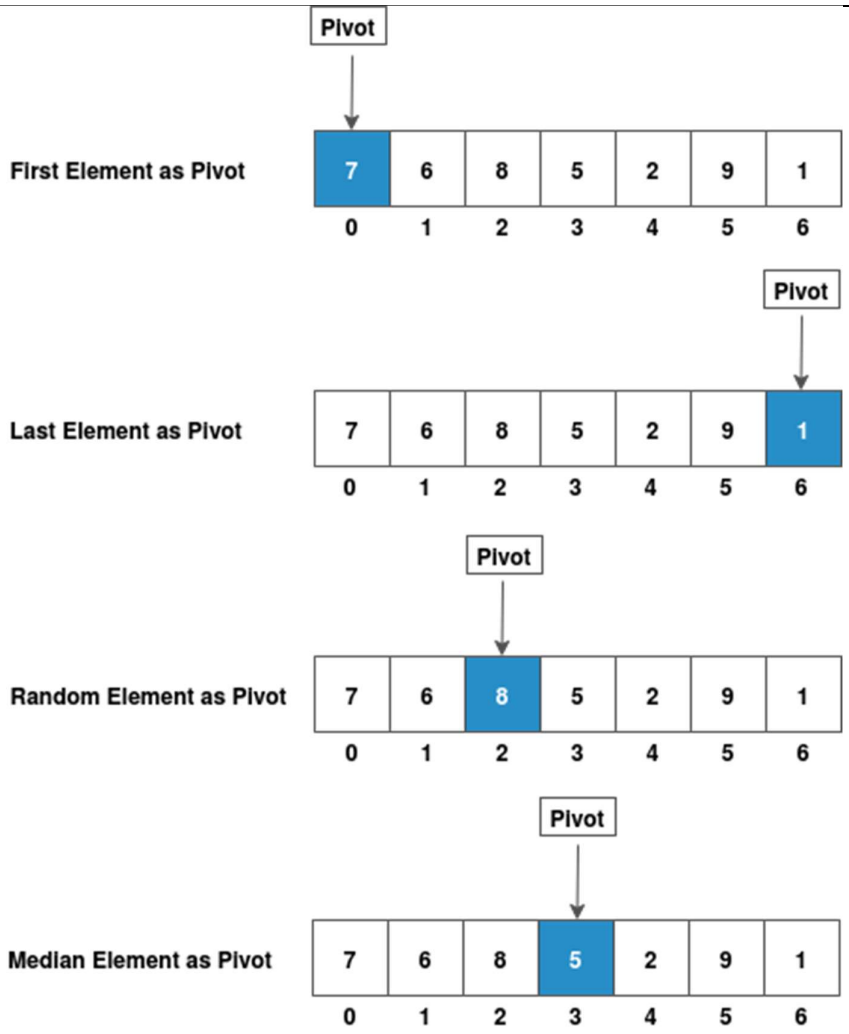
Partitioning array around the pivot means we divide the array into three parts

- **Partition-1** : Contains all elements of that array less than or equal to pivot
- **Partition-2** : Contains only pivot
- **Partition-3** : Contains all elements of that array greater than pivot

Steps to Partition an Unsorted Array

- STEP 1 : First select a pivot in an unsorted array as shown above in the pivot selection section
- STEP 2 : Initialize the low index and high index. Low index represents the first index of an array. High index represents the last index of an array.
- STEP 3 : Start comparing low index element and high index element with the pivot. First start comparing with low index element with pivot element.
    - If low index element is less than the pivot element, increment low index otherwise start comparing high index element with pivot element.
    - If high index element is greater than pivot element, decrement high index otherwise stop.
- STEP 4 : Compare low index with high index.
    - If low index is less than high index swap low index element with the high index element.
    - If low index is greater than high index swap high index element with pivot index element.
- STEP 5 : Repeat the steps 1 to 4 recursively for the sub-arrays until we get a sorted array.

Pictorial Representation of Partitioning Unsorted Array



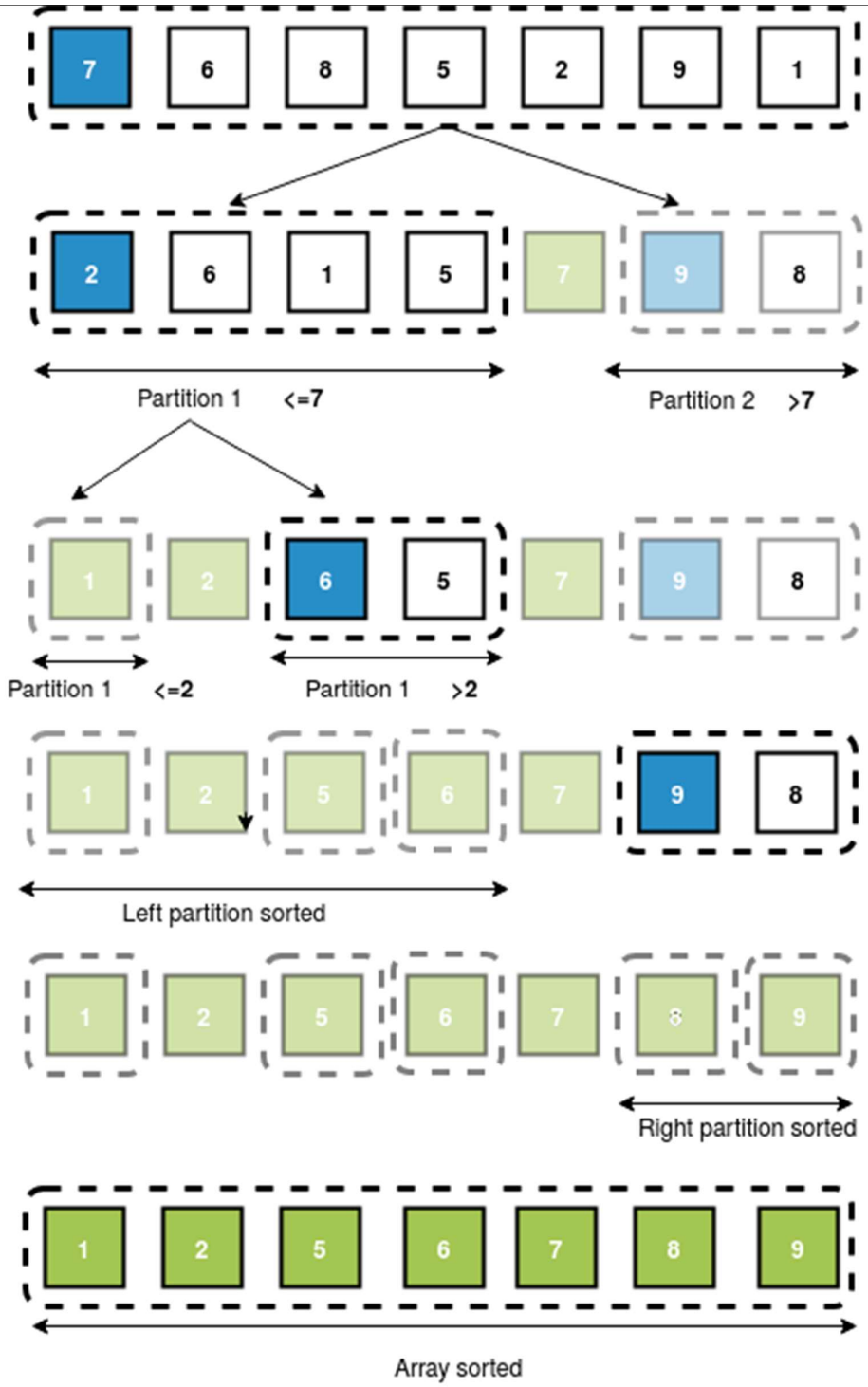
What is Recursion?

On the given array we have to perform partition on that array and recursively on partition-1 and partition-3. But we have to stop at some point i.e., we will stop when the partition is already sorted. Remember that array containing one element is always sorted. So, we proceed till the partition is of length one.

What is Concatenation?

After all sub-arrays(partitions and their partitions and so on) are sorted, we concatenate(place one after the other) all sub-arrays according to their indices and that gives us the sorted-array.

Pictorial Representation of Sorting Array with Quick Sort



Running Time of Quick Sort

The time taken by quick sort to sort an array either in ascending or descending order generally depends upon the input array and partition strategy.

Following are three cases which we come across while sorting an array with quick sort algorithm:

- Best Case
- Worst Case
- Average Case

Best Case Time Complexity Analysis

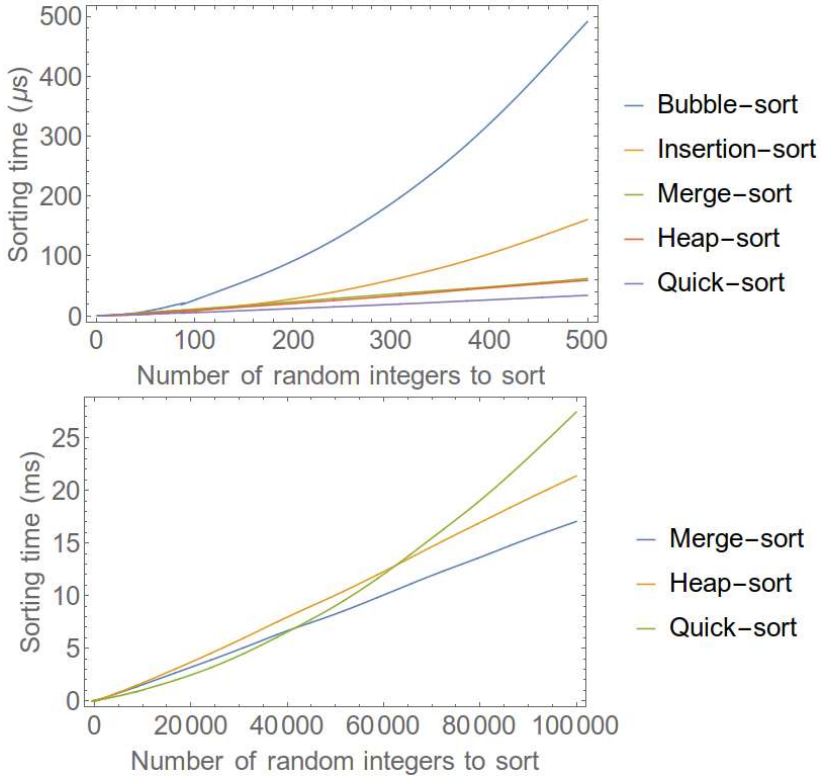
Best case time complexity comes when the no. of levels of recursion is minimum.As the partition divides it into two sub-arrays, if there is a symmetry then both takes same time(both have same no. of elements). Taking the pivot to be the middle element of the sorted array(at each step of recursion), after partition one of the subarray has n/2-1 elemenets while the other has n/2. The divide continues till a single element is left in the array. So, n/(2^i)=1, i=logn(base 2). The depth comes out to be logn. This partition goes on for n elements of the array, hence the best time complexity comes out to be T(n)=nlogn.

Worst Case Time Complexity Analysis

Think of an already sorted array as an input, then the position of pivot does not changes on partition. So, we only have one partition to sort(only left). As this continues the no. of levels of recursion will be O(n)(at each step one element is sorted). This partition goes on for n elements of the array, hence the average time complexity comes out to be T(n)=nlogn.

Average Case Time Complexity Analysis

The average-case running time is also O(nlogn), because the average-case running time cannot be better than the best-case running time. First, let's imagine that we don't always get evenly balanced partitions, but that we always get at worst a 3-to-1 split. That is, imagine that each time we partition, one side gets 3n/4 elements and the other side gets n/4. (To keep the math clean, let's not worry about the pivot.) T(n) = T(n/4) + T(3n/4) T(n)-> time for n elements. On solving this equation we get T(n) = O(nlogn) The left child of each node represents a subproblem size 1/4 as large, and the right child represents a subproblem size 3/4 as large. so, the maximum depth 'i' will go to n/((4/3)^i) = 1 => i = log(n)(base (4/3)). We see that log(n)(base 2) and log(n)(base(4/3)) differ by a

	factor of log(4/3)(base 2), which is a constant. So the average case is taken to be O(nlogn) as well. **Space Complexity Analysis** So, the space complexity for one level of recursion will be O(logn) if median of medians is used for pivot selection and O(1) otherwise. But the no of function calls can go upto O(n) in worst case. So, the call stack memory used can goes upto O(n) because of function call. The space used by median of medians in current step is not needed for further steps. So, overall space complexity of Quicksort is O(n) due to function calls in call stack. **Graph : Time Complexities of Sorting Algorithms** 
ERRORS	**(if any)**
REMEDY	**(if any)**

CONCLUSION:	
REFERENCES: refer lab manual for the same	

Continuous Assessment for DS AY: 2023-24			
RPP (5)	**SPO (5)**	**Total (10)**	**Signature:**
			Assessed By:
Start date	**Submission date**		**Date:**
***Regularity, Punctuality, performance** ***Submission, Presentation, orals**			