PART A

(PART A : TO BE REFFERED BY STUDENTS)

Experiment No.03

- **Aim:**

Implementation of quick sort.

- **Prerequisite:**

C programming

- **Outcome:**

After successful completion of this experiment students will be,

Apply divide and conquer strategy to solve problems.

- **Theory:**

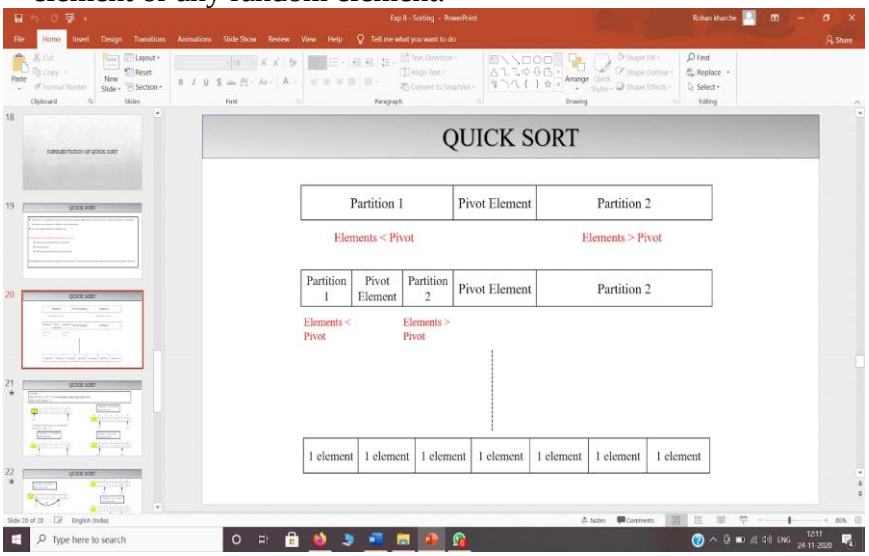
Quick sort:

Quicksort is an algorithm based on divide and conquer approach in which the array is split into subarrays and these sub-arrays are recursively called to sort the elements. It is also called partition-exchange sort.

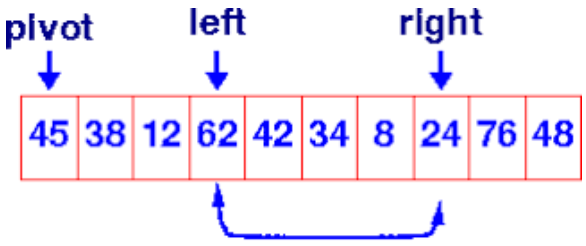
This algorithm divides the list into three main parts:

- Elements less than the Pivot element
- Pivot element
- Elements greater than the pivot element

Pivot element can be any element from the array, it can be the first element, the last element or any random element.



Example:



Algorithm:

Step 1: Repeat the steps 2 to 10 while low < high
Step 2: Select Pivot = a[low]
Pivot location p = low
Step 3: i = low and j = high
Step 4: Repeat the steps 5 to 7 while i < j
Step 4: Increment index i till a[i] < pivot
Step 5: Decrement index j till a[j] > pivot
Step 6: Swap a[i] with a[j]
[End of Loop]
Step 7: if i > j then
Swap a[p] with a[j]
Step 8: call quicksort(a, low, j-1)
Step 9: Call quicksort(a, j+1, high)
[End of Loop]
Step 10: Stop

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

Roll. No. A56	Name: ANAS MUBEEN EDROOS
Class: DSE-MTRX	Batch: DSE
Date of Experiment: 27/2/2021	Date of Submission: 6/3/2021
Grade:	

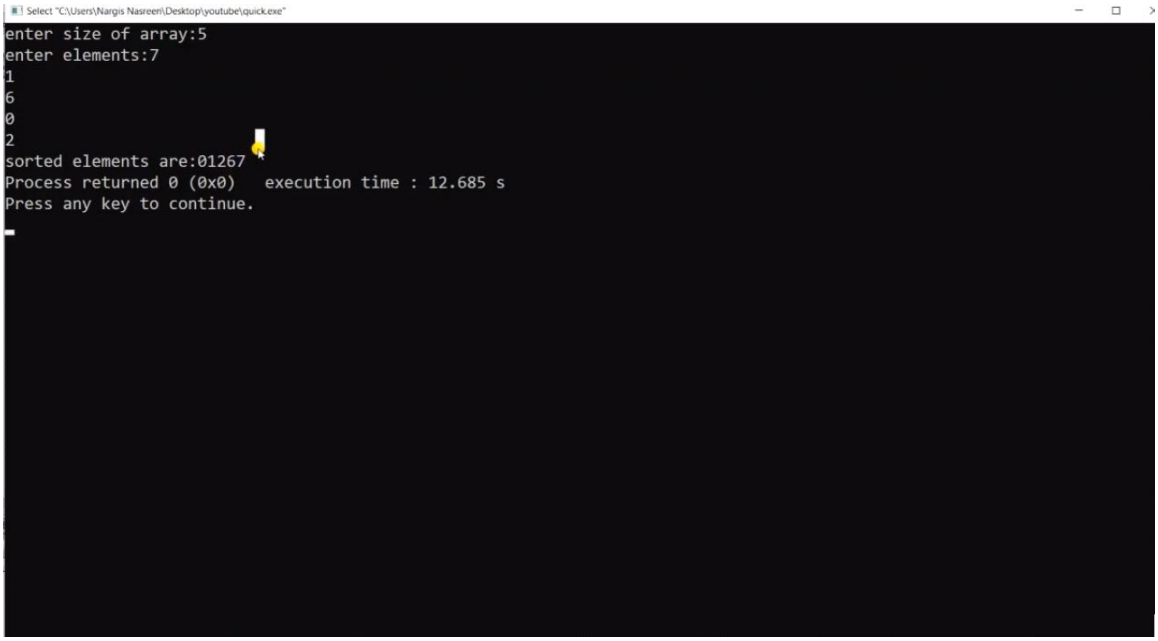
B1. Software Code written by student:

Program:-

```
//ANAS EDROOS
//Roll No 56
#include <bits/stdc++.h>
using namespace std;

void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)
    {
                if (arr[j] < pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
                int pi = partition(arr, low, high);
         quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void printArray(int arr[], int size)
{
```

```
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int arr[] = {1,6,0,2};
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

B2. Output:



B3. Observations and learning:

We observe how implemation of quick sort are done.

B4. Conclusion:

We have understood the implementation of quick sort.

B5. Question of Curiosity

1) what is the time complexity of quick sort?

soln. To sort an array of n distinct elements, quicksort takes O(n log n) time in

expectation, averaged over all n! permutations of n elements with equal probability.

Best-case performance

Worst-case space complexity

Worst-case performance

Average performance

2)Explain internal and external sorting.

soln. Internal sorting: If the input data is such that it can be adjusted in the main memory at once, it is called internal sorting.

External sorting: If the input data is such that it cannot be adjusted in the memory entirely at once, it needs to be stored in a hard disk, floppy disk, or any other storage device. This is called external sorting.

3) Quick Sort follows Divide and Conquer Strategy

a)true b) false

soln.

a) True

Explanation: In quick sort, the array is divided into sub-arrays and then it is sorted (divide-and-conquer strategy).