Lab 5:
Quick sort

Aim: Implement quick sort to the given list of numbers. Display the corresponding list in each pass.

Theory:

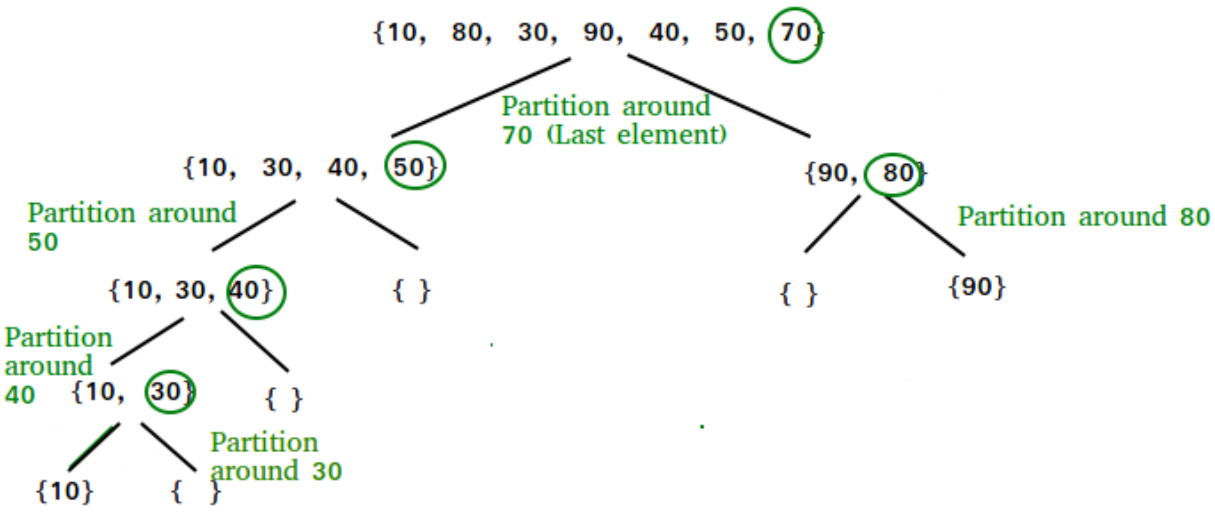
Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target 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 elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Pseudo Code for recursive QuickSort function :

```
/* low  --> Starting index,  high  --> Ending index */

quickSort(arr[], low, high)

{

   if (low < high)

   {

      /* pi is partitioning index, arr[pi] is now

         at right place */

      pi = partition(arr, low, high);


      quickSort(arr, low, pi - 1);  // Before pi

      quickSort(arr, pi + 1, high); // After pi

   }

}
```



Partition Algorithm

There can be many ways to do partition, following pseudo code adopts the method given in CLRS book. The logic is simple, we start from the leftmost element and keep track of index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
   if (low < high)
   {
      /* pi is partitioning index, arr[pi] is now
         at right place */
      pi = partition(arr, low, high);

      quickSort(arr, low, pi - 1);  // Before pi
      quickSort(arr, pi + 1, high); // After pi
   }
}
```

Pseudo code for partition()

```
/* This function takes last element as pivot, places
  the pivot element at its correct position in sorted
```

```
   array, and places all smaller (smaller than pivot)
  to left of pivot and all greater elements to right
  of pivot */
partition (arr[], low, high)
{
   // pivot (Element to be placed at right position)
   pivot = arr[high];

   i = (low - 1)  // Index of smaller element

   for (j = low; j <= high- 1; j++)
   {
      // If current element is smaller than the pivot
      if (arr[j] < pivot)
      {
         i++;    // increment index of smaller element
         swap arr[i] and arr[j]
      }
   }
   swap arr[i + 1] and arr[high])
   return (i + 1)
}
```

Illustration of partition() :

```
arr[] = {10, 80, 30, 90, 40, 50, 70}
Indexes:  0   1   2   3   4   5   6


low = 0, high =  6, pivot = arr[h] = 70
Initialize index of smaller element, i = -1

Traverse elements from j = low to high-1
j = 0 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
i = 0
```

arr[] = {10, 80, 30, 90, 40, 50, 70} // No change as i and j
// are same

j = 1 : Since arr[j] > pivot, do nothing
// No change in i and arr[]

j = 2 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
i = 1
arr[] = {10, 30, 80, 90, 40, 50, 70} // We swap 80 and 30

j = 3 : Since arr[j] > pivot, do nothing
// No change in i and arr[]

j = 4 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
i = 2
arr[] = {10, 30, 40, 90, 80, 50, 70} // 80 and 40 Swapped
j = 5 : Since arr[j] <= pivot, do i++ and swap arr[i] with arr[j]
i = 3
arr[] = {10, 30, 40, 50, 80, 90, 70} // 90 and 50 Swapped

We come out of loop because j is now equal to high-1.
Finally we place pivot at correct position by swapping arr[i+1] and arr[high] (or pivot)
arr[] = {10, 30, 40, 50, 70, 90, 80} // 80 and 70 Swapped

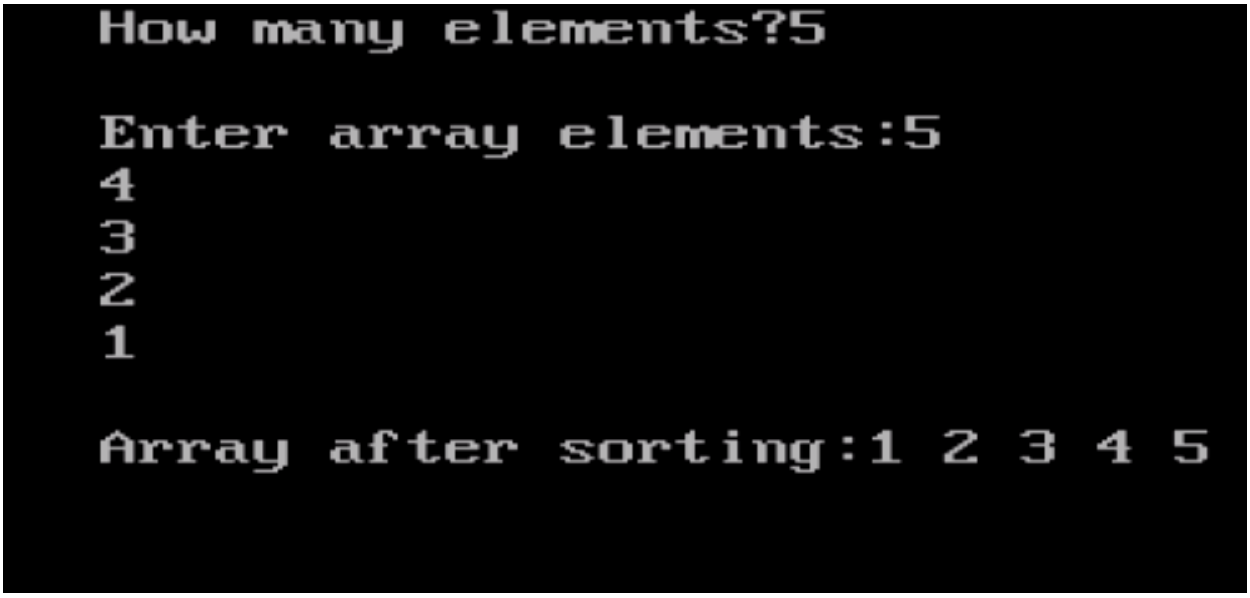
Now 70 is at its correct place. All elements smaller than 70 are before it and all elements greater than 70 are after it.

Program:

```
#include<iostream.h>

#include<conio.h>


void quick_sort(int[],int,int);

int partition(int[],int,int);


int main()

{

   int a[50],n,i;

   cout<<"How many elements?";

   cin>>n;

   cout<<"\nEnter array elements:";


   for(i=0;i<n;i++)

           cin>>a[i];


   quick_sort(a,0,n-1);

   cout<<"\nArray after sorting:";


   for(i=0;i<n;i++)

           cout<<a[i]<<" ";


   return 0;

}
```

```
void quick_sort(int a[],int l,int u)
{
   int j;
   if(l<u)
   {
         j=partition(a,l,u);
         quick_sort(a,l,j-1);
         quick_sort(a,j+1,u);
   }
}

int partition(int a[],int l,int u)
{
   int v,i,j,temp;
   v=a[l];
   i=l;
   j=u+1;

   do
   {
         do
            i++;

         while(a[i]<v&&i<=u);
```

```
        do
          j--;
        while(v<a[j]);

        if(i<j)
        {
          temp=a[i];
          a[i]=a[j];
          a[j]=temp;
        }
    }while(i<j);

    a[l]=a[j];
    a[j]=v;

    return(j);
}
```



Conclusion:

In this way we have successfully studied how to sort elements using quick sort by partitioning.