a. Implement circular queue using arrays :::

```
package com.circularqueue.main;

public class Circularque
{
    private int data;
    private int front;
    private int rear;
    private   int arr[];
    private int size;

          public  Circularque (int size)
          {

           front=-1;
           rear=-1;
           this.size=size;
           arr=new int [this.size];

          }


          public void Enqueue(int data)
          {
           if(rear==(size-1) && front==0)
           {
                System.out.println("Queue is full");
                return;
           }
           if(front==rear+1)
           {
              System.out.println("Queue is full");
              return;
           }

           if(rear==size-1)
           {
                rear=0;
                arr[rear]=data;
                return;
           }
           rear++;
```

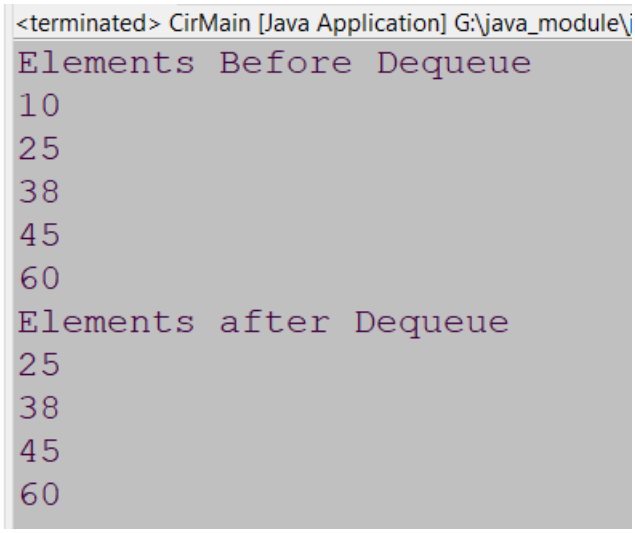
```
            if(front==-1)
            {
                 front++;//front=0
            }
            arr[rear]=data;


           }
           public void display()
           {
if(rear>front)
{
            for(int i=front;i<=rear; i++)//- - - - -
            {

                 System.out.println(arr[i]);


            }
     }
else
{

     for(int i=front;i<size;i++)
     {
     System.out.println(arr[i]);
     }
     for(int i=0;i<front;i++)
     {
     System.out.println(arr[i]);
     }
}
}

           public void Dequeue()
           {

            if(front==-1)
            {
                 System.out.println("Queue is empty");
                 return;
            }
            arr[front]=0;
```

```
            front++;

           }

}
```

Main Method ///////////////

```
package com.circularqueue.main;

public class CirMain {

    public static void main(String[] args) {

        Circularque ref= new Circularque (5);
        ref.Enqueue(10);//arr[0]=36
        ref.Enqueue(25);//arr[1]=100
        ref.Enqueue(38);//arr[2]=38
        ref.Enqueue(45);//arr[3]=45
        ref.Enqueue(60);//arr[4]=60

    System.out.println("Elements Before Dequeue");

        ref.display();// rear=4 front=0
    ref.Dequeue();
    System.out.println("Elements after Dequeue");
    ref.display();

    }

}
```

Output ///////////////



b. Perform quick sort to arrange given set of elements

```
package com.dsa.java;

import java.util.Arrays;

class Quicksort {


  static int partition(int array[], int low, int
high) {


    int pivot = array[high];


    int i = (low - 1);

    for (int j = low; j < high; j++) {
      if (array[j] <= pivot) {


        i++;


        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
```

```
      }

    }


    int temp = array[i + 1];
    array[i + 1] = array[high];
    array[high] = temp;


    return (i + 1);
  }

 static void quickSort(int array[], int low, int
high) {
    if (low < high) {

      // find pivot element such that
      // elements smaller than pivot are on the left
      // elements greater than pivot are on the right
      int pi = partition(array, low, high);

      // recursive call on the left of pivot
      quickSort(array, low, pi - 1);

      // recursive call on the right of pivot
      quickSort(array, pi + 1, high);
    }
  }
}
```

Main //////////////////////////

```
package com.dsa.java;

import java.util.Arrays;

public class SortMain {

    public static void main(String[] args) {
```

```
        int[] data = { 8, 7, 2, 1, 0, 9, 6 };
        System.out.println("Unsorted Array");
        System.out.println(Arrays.toString(data));

        int size = data.length;

        // call quicksort() on array data
        Quicksort.quickSort(data, 0, size - 1);

        System.out.println("Sorted Array in Ascending
Order ");
        System.out.println(Arrays.toString(data));
    }

}
```

Output //////////////////