4. Sort a given set of *n* integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of *n*> 5000 and record the time taken to sort. Plot a graph of the time taken versus *n*on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divideand-conquer method works along with its time complexity analysis: worst case, average case and best case.

```
import java.util.Scanner;
import java.util.Random;
import java.io.*;
public class QuickSortDemo
{
    static int size; // To read size of input elements
    public static void main(String [] args)throws IOException
    {
       Scanner in = new Scanner(System.in);
       System.out.println("Enter the number of elements to sorted: ( >5000):");
       size =  in.nextInt();
       //Declare an array of dimension 'size'
       int inputArr [] = new int[size];
       genRandomNumbers(inputArr);
       //Sort the randomly generated numbers for average case and best case complexity
       long startTime = System.nanoTime();
       quickSort(inputArr,0,size-1);
       long estimatedTime = System.nanoTime() - startTime;
       PrintWriter outA = new PrintWriter(new File("qsort.txt"));
       for(int i=0;i<inputArr.length;i++)
       {
        outA.println(inputArr[i]);
       }
       outA.close();
       System.out.println("The time complexity for best case and average case is " +
    (estimatedTime/1000000.0)+ " ms");

       //Sort the sorted numbers for worst case complexity
        startTime = System.nanoTime();
        quickSort(inputArr,0,size-1);
        estimatedTime = System.nanoTime() - startTime;
        System.out.println("The time complexity for worst case is " +
    (estimatedTime/1000000.0)+ " ms");

  }
```

```
// Method to generate the Random Numbers
public static void genRandomNumbers(int inputArr[]) throws IOException
  {
   int  number, count=0;
   Random rand = new Random();
   PrintWriter out = new PrintWriter(new File("Random.txt"));
   while(count<size)
   {
    number=rand.nextInt(size)+1;
    out.println(number);
    inputArr[count]=number;
    count++;
   }
   out.close();
   System.out.println("The total numbers generated: " + count );

  }
 //Method for QuickSort
 public static void quickSort(int a[],int low,int high)
 {
    int j;
    if(low<high)
    {
     j= partition(a,low,high);
     quickSort(a,low,j-1);
     quickSort(a,j+1,high);
    }
 }
 //Method for Partition
 public static int partition(int a[],int low,int high)
 {
    int i,j,temp,pivot;
    pivot = a[low];
    i = low + 1;
    j = high;

    while(true)
    {
     while(i<high && a[i]<=pivot)
        i++;
     while(a[j]>pivot)
        j--;
     if(i<j)
     {
       temp = a[i];
       a[i] = a[j];
```

```
        a[j] = temp;
      }
      else
      {
        temp = a[low];
        a[low] = a[j];
        a[j] = temp;
        return j;
      }
    }// End of While
  } // End of Main Method  Partition method
} // End of class
```