5. Sort a given set of n integer elements using Merge 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 non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide and 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 MergeSortDemo
{
  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 best case, average case and worst case
      complexity
     long startTime = System.nanoTime();
     mergeSort(inputArr,0,size-1);
     long estimatedTime = System.nanoTime() - startTime;
     PrintWriter outA = new PrintWriter(new File("msort.txt"));
     for(int i=0;i<inputArr.length;i++)
     {
       outA.println(inputArr[i]);
     }
     outA.close();
     System.out.println("The time complexity for best,average and worst case is " +
(estimatedTime/1000000.0)+ " ms");
   }
  // Method for random number generation
  public static void genRandomNumbers(int inputArr[]) throws IOException
  {
     int  number, count=0;
     Random rand = new Random();
     PrintWriter out = new PrintWriter(new File("Mrandom.txt"));
     while(count<size)
    {
                number=rand.nextInt(size)+1;
```

```
                    out.println(number);
                    out.print(" ");
                    inputArr[count]=number;
                    count++;
    }
   out.close();
   System.out.println("The total numbers generated: " + count );
  }
 // Method to
  public static void merge(int a[],int low,int mid,int high)
  {
                  int i = low;
                  int j = mid+1;
                  int k = low;

                  int c [] = new int[1000000];
                  while(i<=mid && j<=high)
                  {
                   if(a[i] < a[j])
                   {
                          c[k] = a[i];
                          i = i+1;
                          k = k+1;
                   }
                   else
                   {
                          c[k] = a[j];
                          j = j+1;
                          k = k+1;
                   }
                  }
                  while(i<=mid)
                  {
                   c[k++] = a[i++];
                  }
                  while(j<=high)
                  {
                   c[k++] = a[j++];
                  }
                  for(i=low;i<=high;i++)
                  {
                   a[i] = c[i];
                  }
  }
 public static void mergeSort(int a[],int low,int high)
   {
     int mid;
     if(low<high)
     {
      mid = (low+high)/2;
      mergeSort(a,low,mid);
      mergeSort(a,mid+1,high);
      merge(a,low,mid,high);
     }
   }
}
```

Time Complexity for QuickSort algorithm

Best Case	**O(nlogn)**
Average Case	**O(nlogn)**
Worst Case	**O(n^2)**