Name - Raveet Kumar
Roll No. - 1804310039
Branch - Computer Science and Engineering
College - BIET, Jhansi

Experiment - 05

Aim-

We have to implement 2 different algorithms and study the Number of Comparisons for each of them and draw graphs of their best, worst and average case.

- *Min/Max Element*
- *Kth Smallest*

Tools & Language Used-

❏ Java - for coding the algorithm and calculating time
❏ Python - for plotting graphs using matplotlib module.

Code & Analysis-

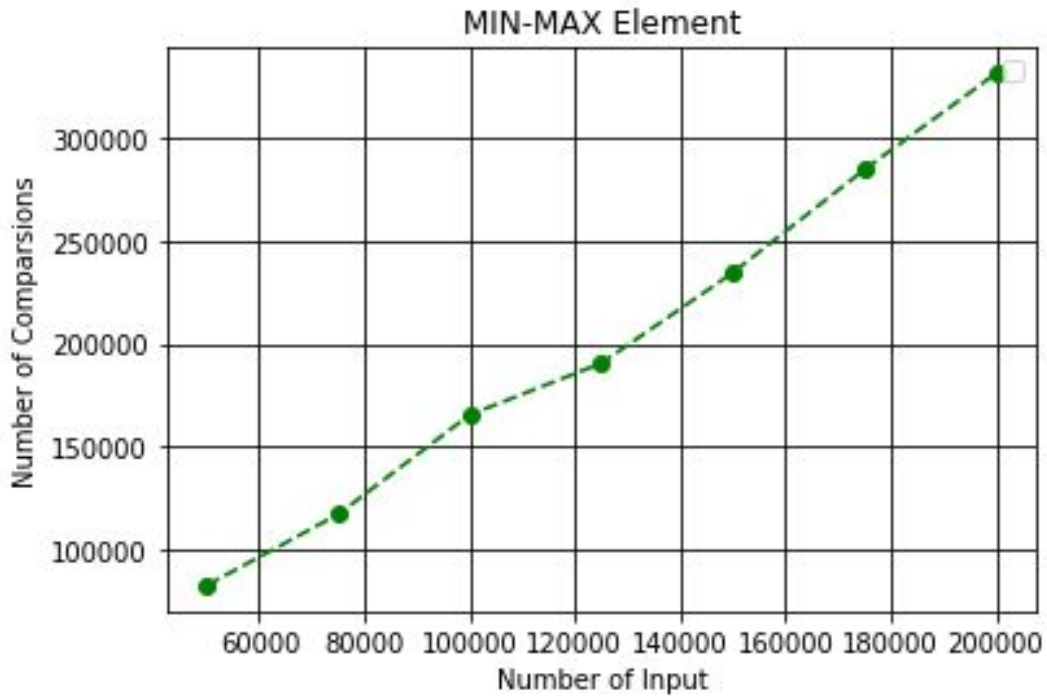
- **Min/Max**(Using Tournament method)
- Code

```
public class MinMax {

    public static void main(String[] args) throws IOException {

        File f=new File("D:\\Algorithm time Complexity
analysis\\min_max_comp_analysis.txt");
        BufferedWriter bw=new BufferedWriter(new FileWriter(f,false),2);

        //50000,75000,100000,125000,150000,175000,200000
        List<Integer>
TestCase=Arrays.asList(50000,75000,100000,125000,150000,175000,200000);
        int[] arr;


        int k=0;
        bw.write("Number_of_Input\t\t\tNumber_of_comparisions\n");
```

```
        while(k < TestCase.size()) {

            int arrSize=TestCase.get(k);
            arr=new int[arrSize];

            Random rand=new Random();    // To Generate Random Numbers...

            for(int i=0;i<arrSize;i++) {
                arr[i]=rand.nextInt(arrSize*10);            //Filling Numbers in
the range of (0, arrSize*10-1) in array of size arrSize
            }


            CountComparisons=0;
            //Avg case.....
            Pair minmax=minMax(arr,0,arr.length-1);
            bw.write(String.format("%d\t\t\t%d\n",arrSize,CountComparisons));

//          System.out.println(minmax.min+" "+minmax.max+" "+CountComparisons);

            k++;
            System.out.println("Success");
        }
        bw.close();

    }

    static int CountComparisons;

    static class Pair{
        int min;
        int max;

        Pair(int min,int max){
            this.min=min;
            this.max=max;
        }
    }

    // By Tournament Method
    private static Pair minMax(int[] arr, int l,int h) {
        if(l==h) {
            return new Pair(arr[l],arr[h]);
        }
        else if(l==h-1) {
            CountComparisons++;
            if(arr[l]>arr[h]) {
                return new Pair(arr[h],arr[l]);
            }else {
                return new Pair(arr[l],arr[h]);
            }
```

```
        }
        int mid=l+(h-l)/2;
        Pair minmaxl=minMax(arr,l,mid);
        Pair minmaxr=minMax(arr,mid+1,h);
        int localmin,localmax;

        CountComparisons+=2;

        if(minmaxl.min < minmaxr.min) {
            localmin=minmaxl.min;
        }else {
            localmin=minmaxr.min;
        }

        if(minmaxl.max > minmaxr.max) {
            localmax=minmaxl.max;
        }else {
            localmax=minmaxr.max;
        }

        return new Pair(localmin,localmax);
    }


}
```

Graph:



- **Kth Smallest Element**
  - Code

```
public class KthSmallest {
    public static void main(String[] args) throws IOException {

        File f=new File("D:\\Algorithm time Complexity
analysis\\kthSmallest_comp_analysis.txt");
        BufferedWriter bw=new BufferedWriter(new FileWriter(f,false),2);

        //50000,75000,100000,125000,150000,175000,200000
        List<Integer>
TestCase=Arrays.asList(50000,75000,100000,125000,150000,175000,200000);
        int[] best;
        int[] worst;
        int[] avg;


        int t=0;
        bw.write("Number_of_Input\t\t\tNumber_of_comparisions\n");

        while(t < TestCase.size()) {


            int arrSize=TestCase.get(t);
            best=new int[arrSize];
            worst=new int[arrSize];
            avg=new int[arrSize];
```

```
            Random rand=new Random();    // To Generate Random Numbers...

            for(int i=0;i<arrSize;i++) {
                avg[i]=rand.nextInt(arrSize*10);            //Filling Numbers in the
range of (0, arrSize*10-1) in array of size arrSize
            }

            for(int i=0;i<arrSize;i++) {
                best[i]=avg[i];
            }
            Arrays.sort(best);                  // To make a sorted array... which we
will use for best case..

            for(int i=0;i<arrSize;i++) {
                worst[i]=best[arrSize-1-i];     // To make reverse order of the Best case
... to Check worst case..
            }

            int k=arrSize/3;

            CountComparisons=0;
            int kthSmallest=kthSmallest(best,0,best.length-1,arrSize/3);

bw.write(String.format("Best_case\t\t\t%d\t\t\t%d\n",arrSize,CountComparisons));
            System.out.println(k+" "+kthSmallest+" "+CountComparisons);

            CountComparisons=0;
            //it will be the worst case
            kthSmallest=kthSmallest(worst,0,worst.length-1,arrSize/3);

bw.write(String.format("Worst_case\t\t\t%d\t\t\t%d\n",arrSize,CountComparisons));
            System.out.println(k+" "+kthSmallest+" "+CountComparisons);

            CountComparisons=0;
            //Avg case.....
            kthSmallest=kthSmallest(avg,0,avg.length-1,arrSize/3);

bw.write(String.format("Avg_case\t\t\t%d\t\t\t%d\n",arrSize,CountComparisons));
            System.out.println(k+" "+kthSmallest+" "+CountComparisons);

            t++;
            System.out.println("Success");
        }
        bw.close();

    }

    static int CountComparisons;


    static int kthSmallest(int arr[], int l, int r, int k)
    {
        if (k > 0 && k <= r - l + 1)
        {
            int n = r - l + 1 ;

            int i;
```

```
        int []median = new int[(n + 4) / 5];
        for (i = 0; i < n/5; i++)
            median[i] = getMedian(arr,l + i * 5, 5);

        if (i*5 < n)
        {
            median[i] = getMedian(arr,l + i * 5, n % 5);
            i++;
        }

        int medOfMed = (i == 1)? median[i - 1]:
                                 kthSmallest(median, 0, i - 1, i / 2);

        int pos = partition(arr, l, r, medOfMed);

        if (pos-l == k - 1)
            return arr[pos];
        if (pos-l > k - 1)
            return kthSmallest(arr, l, pos - 1, k);

        return kthSmallest(arr, pos + 1, r, k - pos + l - 1);
    }
    return Integer.MAX_VALUE;
}

private static int getMedian(int[] arr, int l, int r) {
    Arrays.sort(arr,l,l+r);
    CountComparisons+=r;
    return arr[l+(r)/2];
}


static int partition(int arr[], int low, int high,int pivot)
{
    int i;
    for(i = low; i < high; i++)
        if (arr[i] == pivot)
        break;
    swap(arr, i, high);

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


private static void swap(int[] arr, int i, int j) {
    int temp=arr[i];
```

```
        arr[i]=arr[j];
        arr[j]=temp;
    }

}
```

Graph: