

Istanbul Technical University  
Faculty of Computer and Informatics  
Computer Engineering Department

BLG 335E  
The L<sup>A</sup>T<sub>E</sub>X  
Report

**Analysis of Algorithms I, Project 2**  
Leminur Çelik - 150190085

December 4<sup>th</sup>, 2022

# Contents

<b>1</b>	<b>Heap Functions</b>	<b>1</b>
1.1	Max Heapify . . . . .	1
1.2	Build Max Heap . . . . .	1
1.3	Heap Sort . . . . .	2
1.4	Heap Increase Key . . . . .	2
1.5	Heap Insert . . . . .	3
<b>2</b>	<b>Estimate Statistics</b>	<b>3</b>
2.1	Mean . . . . .	3
2.2	Standard Deviation . . . . .	4
2.3	Median . . . . .	4
2.4	Minimum . . . . .	4
2.5	Maximum . . . . .	5
2.6	First Quartile . . . . .	5
2.7	Third Quartile . . . . .	5
<b>3</b>	<b>Images</b>	<b>6</b>
3.1	Mean Function . . . . .	6
3.2	Standard Deviation Function . . . . .	8
3.3	Median Function . . . . .	10
3.4	Minimum Function . . . . .	12
3.5	Maximum Function . . . . .	13
3.6	First Quartile Function . . . . .	15
3.7	Third Quartile Function . . . . .	17
<b>4</b>	<b>Graphs</b>	<b>19</b>

# 1 Heap Functions

I used heap data structure in my implementation. I created a max-heap to store elements given in the input file. Max-heap property is that each parent node is largest than its children. The largest element is stored in the root. The height of a heap is equal to height of its root. Its height is  $O(\lg n)$ , because the heap consists of  $n$  elements and has the property of a complete binary tree. I wrote 5 classes as shown below:

- Max-Heapify
- Build Max Heap
- Heap Sort
- Heap Increase Key
- Heap Insert

## 1.1 Max Heapify

Since it does not contain loops and does not depend on the number of steps, the runtime is a constant time. Execution time is  $O(1)$ . However, it creates a recursive call because it calls itself at the end of the function. For a full binary tree, half of the tree is included. usually  $2/3$  of it is included. At worst, the last line is half filled. Therefore  $T(n) = T(2n/3) + O(1)$ . This is equal to  $O(\lg n)$ .

---

**Algorithm 1** Max heapify

---

```
function MAX-HEAPIFY(vector, i)
    left  $\leftarrow 2 * i$ 
    right  $\leftarrow 2 * i + 1$ 
    largest  $\leftarrow i$ 
    if left  $\leq$  size AND vector[left] > vector[i] then
        largest  $\leftarrow$  left
    end if
    if right  $\leq$  size AND vector[right] > vector[largest] then
        largest  $\leftarrow$  right
    end if
    if largest  $\neq i$  then
        SWAP(vector[i], vector[largest])
        MAX-HEAPIFY(vector, largest)
    end if
end function
```

---

## 1.2 Build Max Heap

Each call to max-heapify function costs  $O(\lg n)$ . Calls are  $O(n)$ . Therefore worst case running time is  $O(n \lg n)$ .

---

**Algorithm 2** Build Max Heap

---

```
function BUILD-MAX-HEAP(vector)
     $i \leftarrow size/2$ 
    while  $i \geq 0$  do
        MAX-HEAPIFY( $i$ )
         $i \leftarrow i - 1$ 
    end while
end function
```

---

### 1.3 Heap Sort

First build a max heap. Then, remove the largest element from the heap and move it to the first position in the sorted array. The element placed at the top of the heap shift to the right position. Build max heap function runs  $O(n)$  time. Assignments take constant time. Max heapify function runs  $O(\lg n)$  time. Since the function is inside the for loop, it will have  $O(n \lg n)$  runtime. Therefore, the execution time for heap sort is  $O(n \lg n)$ .

---

**Algorithm 3** Heap Sort

---

```
function HEAP-SORT(vector)
    BUILD-MAX-HEAP(vector)
     $i \leftarrow (size - 1)$ 
    while  $i > 0$  do
        SWAP( $vector[1], vector[i]$ )
         $size \leftarrow size - 1$ 
        MAX-HEAPIFY( $vector, 1$ )
    end while
end function
```

---

### 1.4 Heap Increase Key

Loop executes the depth of the tree. The depth of a binary tree is  $O(\lg n)$ . The execution time of heap increase key function is  $O(\lg n)$ .

---

**Algorithm 4** Heap Increase Key

---

```
function HEAP-INCREASE-KEY(vector,  $i$ )
     $parent \leftarrow (i - 1)/2$ 
    if  $vector[parent] > 0$  then
        if  $vector[i] > vector[parent]$  then
            SWAP( $vector[i], vector[parent]$ )
            HEAP-INCREASE-KEY( $parent$ )
        end if
    end if
end function
```

---

## 1.5 Heap Insert

When a new element arrives, it is added to the heap in the insert function. There is no loops. Assignments get constant time. At the end of the function heap increase key function is called. Running time is  $O(\lg n)$ . Therefore, the execution time for heap insert is  $O(\lg n)$ .

---

**Algorithm 5** Heap Insert

---

```
function HEAP-INSERT(vector, key)
    PUSH BACK(vector, key)
     $size \leftarrow size + 1$ 
    HEAP-INCREASE-KEY(heap-size)
end function
```

---

## 2 Estimate Statistics

There are 7 statistics functions:

- Mean
- Standard Deviation
- Median
- Minimum
- Maximum
- First Quartile
- Third Quartile

### 2.1 Mean

The mean is calculated by dividing the sum of all values by the number of values. While loop takes  $n$  steps. Therefore, the execution time for mean function is  $O(n)$ .

---

**Algorithm 6** Mean

---

```
function MEAN(heap, size)
     $total \leftarrow 0$ 
    while  $i < size$  do
         $total \leftarrow total + heap[i]$ 
         $i \leftarrow i + 1$ 
    end while
     $mean \leftarrow total / size$ 
    return mean
end function
```

---

## 2.2 Standard Deviation

The standard deviation is calculated by dividing the sum of the squares of the difference of all values from the arithmetic mean by the number of values minus one. In the function, mean function is called. Mean function executes  $O(n)$  times. While loop takes  $n$  steps. Therefore, the execution time for standard deviation is  $O(n)$ .

---

**Algorithm 7** Standard Deviation

---

```
function STANDARD-DEVIATION(heap, size)
    MEAN(heap, size)
    total  $\leftarrow$  0
    while i < size do
        total  $\leftarrow$  POW(heap[i] - mean, 2)
        i  $\leftarrow$  i + 1
    end while
    return SQRT(total / (size - 1))
end function
```

---

## 2.3 Median

The median is equal to the middle number if the number of values is odd, and if it is even, it is obtained by dividing the sum of the two middle numbers by two. The execution time of median function is  $O(1)$ .

---

**Algorithm 8** Median

---

```
function MEDIAN(heap, size)
    if size%2  $\leftarrow$  0 then
        median  $\leftarrow$  (heap[size/2] + heap[size/2 - 1])/2
    end if
    if size%2  $\neq$  0 then
        median  $\leftarrow$  heap[((size + 1)/2) - 1]
    end if
    return median
end function
```

---

## 2.4 Minimum

When heap is sorted, the array is sorted from smallest to largest. Therefore, the value in the smallest index equals the minimum value. The execution time of minimum function is  $O(1)$ .

---

**Algorithm 9** Minimum

---

```
function MINIMUM(heap)
    return heap[0]
end function
```

---

## 2.5 Maximum

When heap is sorted, the array is sorted from smallest to largest. Therefore, the value in the last index equals the maximum value. The execution time of maximum function is  $O(1)$ .

---

**Algorithm 10** Maximum

---

```
function MAXIMUM(heap, size)  
    return heap[size - 1]  
end function
```

---

## 2.6 First Quartile

When heap is sorted, the array is sorted from smallest to largest. Locate the position of the value in the distribution. First Quartile is calculated by formula  $(1/4)(\text{size}-1)$ . Then, find difference between two values less than quartile value and bigger than quartile value. And then, get the decimal part of the quartile and multiply with difference. Finally, add the result to the less than quartile value. Assignments take constant time. The execution time of first quartile function is  $O(1)$ .

---

**Algorithm 11** First Quartile

---

```
function FIRST-QUARTILE(heap, size)  
    quartile  $\leftarrow 0.25 \times (\text{size} - 1)$   
    difference  $\leftarrow \text{heap}[\text{CEIL}(\text{quartile})] - \text{heap}[\text{FLOOR}(\text{quartile})]$   
    firstq  $\leftarrow (\text{quartile} - \text{FLOOR}(\text{quartile})) \times \text{difference} + \text{heap}[\text{FLOOR}(\text{quartile})]$   
    return firstq  
end function
```

---

## 2.7 Third Quartile

When heap is sorted, the array is sorted from smallest to largest. Locate the position of the value in the distribution. Third Quartile is calculated by formula  $(3/4)(\text{size}-1)$ . Then, find difference between two values less than quartile value and bigger than quartile value. And then, get the decimal part of the quartile and multiply with difference. Finally, add the result to the less than quartile value. Assignments take constant time. The execution time of third quartile function is  $O(1)$ .

---

**Algorithm 12** Third Quartile

---

```
function THIRD-QUARTILE(heap, size)  
    quartile  $\leftarrow 0.75 \times (\text{size} - 1)$   
    difference  $\leftarrow \text{heap}[\text{CEIL}(\text{quartile})] - \text{heap}[\text{FLOOR}(\text{quartile})]$   
    thirdq  $\leftarrow (\text{quartile} - \text{FLOOR}(\text{quartile})) \times \text{difference} + \text{heap}[\text{FLOOR}(\text{quartile})]$   
    return thirdq  
end function
```

---

### 3 Images

#### Screenshots of Count of Data Structures and Functions and Execution Time

Figure 1: Input1 File

```
Count max heapify:47
Count build max heap:5
Count heap sort:5
Count heap increase key:9
Count heap insert:7
Count mean:10
Count standard deviation:5
Count median:5
Count minimum:5
Count maximum:5
Count first quantile:5
Count third quantile:5
Time:0 second
```

#### 3.1 Mean Function

Figure 2: Mean with 10 data

```
Count max heapify:0
Count build max heap:0
Count heap sort:0
Count heap increase key:10
Count heap insert:10
Count mean:4
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```



**Figure 3:** Mean with 100 data

```
Count max heapify:0
Count build max heap:0
Count heap sort:0
Count heap increase key:104
Count heap insert:100
Count mean:28
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 4:** Mean with 1000 data

```
Count max heapify:0
Count build max heap:0
Count heap sort:0
Count heap increase key:1330
Count heap insert:1000
Count mean:296
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 5:** Mean with 10000 data

```
Count max heapify:0
Count build max heap:0
Count heap sort:0
Count heap increase key:14315
Count heap insert:10000
Count mean:3026
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 6:** Mean with 100000 data

```
Count max heapify:0
Count build max heap:0
Count heap sort:0
Count heap increase key:136804
Count heap insert:99996
Count mean:30128
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:5 second
```

### 3.2 Standard Deviation Function

**Figure 7:** Standard Deviation with 10 data

```
Count max heapify:0
Count build max heap:0
Count heap sort:0
Count heap increase key:10
Count heap insert:10
Count mean:1
Count standard deviation:1
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 8:** Standard Deviation with 100 data

```
Count max heapify:0
Count build max heap:0
Count heap sort:0
Count heap increase key:104
Count heap insert:100
Count mean:33
Count standard deviation:33
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 9:** Standard Deviation with 1000 data

```
Count max heapify:0
Count build max heap:0
Count heap sort:0
Count heap increase key:1330
Count heap insert:1000
Count mean:300
Count standard deviation:300
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 10:** Standard Deviation with 10000 data

```
Count max heapify:0
Count build max heap:0
Count heap sort:0
Count heap increase key:14315
Count heap insert:10000
Count mean:2878
Count standard deviation:2878
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 11:** Standard Deviation with 100000 data

```
Count max heapify:0
Count build max heap:0
Count heap sort:0
Count heap increase key:136804
Count heap insert:99996
Count mean:29836
Count standard deviation:29836
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:20 second
```

### 3.3 Median Function

**Figure 12:** Median with 10 data

```
Count max heapify:49
Count build max heap:2
Count heap sort:2
Count heap increase key:10
Count heap insert:10
Count mean:0
Count standard deviation:0
Count median:2
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 13:** Median with 100 data

```
Count max heapify:7819
Count build max heap:33
Count heap sort:33
Count heap increase key:116
Count heap insert:100
Count mean:0
Count standard deviation:0
Count median:33
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```



**Figure 14:** Median with 1000 data

```
Count max heapify:975129
Count build max heap:291
Count heap sort:291
Count heap increase key:1425
Count heap insert:1000
Count mean:0
Count standard deviation:0
Count median:291
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 15:** Median with 10000 data

```
Count max heapify:144931542
Count build max heap:3007
Count heap sort:3007
Count heap increase key:17174
Count heap insert:10000
Count mean:0
Count standard deviation:0
Count median:3007
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:4 second
```

**Figure 16:** Median with 100000 data

```
Count max heapify:16473762957
Count build max heap:30162
Count heap sort:30162
Count heap increase key:635202
Count heap insert:99996
Count mean:0
Count standard deviation:0
Count median:30162
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:447 second
```

### 3.4 Minimum Function

Figure 17: Minimum with 10 data

```
Count max heapify:24
Count build max heap:1
Count heap sort:1
Count heap increase key:10
Count heap insert:10
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:1
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```

Figure 18: Minimum with 100 data

```
Count max heapify:8356
Count build max heap:34
Count heap sort:34
Count heap increase key:123
Count heap insert:100
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:34
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```

Figure 19: Minimum with 1000 data

```
Count max heapify:997952
Count build max heap:299
Count heap sort:299
Count heap increase key:1429
Count heap insert:1000
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:299
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 20:** Minimum with 10000 data

```
Count max heapify:144366485
Count build max heap:3006
Count heap sort:3006
Count heap increase key:17180
Count heap insert:10000
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:3006
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:4 second
```

**Figure 21:** Minimum with 100000 data

```
Count max heapify:16426401404
Count build max heap:30079
Count heap sort:30079
Count heap increase key:636109
Count heap insert:99996
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:30079
Count maximum:0
Count first quantile:0
Count third quantile:0
Time:447 second
```

### 3.5 Maximum Function

**Figure 22:** Maximum with 10 data

```
Count max heapify:46
Count build max heap:3
Count heap sort:3
Count heap increase key:10
Count heap insert:10
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:3
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 23:** Maximum with 100 data

```
Count max heapify:7359
Count build max heap:32
Count heap sort:32
Count heap increase key:114
Count heap insert:100
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:32
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 24:** Maximum with 1000 data

```
Count max heapify:1050307
Count build max heap:314
Count heap sort:314
Count heap increase key:1430
Count heap insert:1000
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:314
Count first quantile:0
Count third quantile:0
Time:0 second
```

**Figure 25:** Maximum with 10000 data

```
Count max heapify:143670810
Count build max heap:2957
Count heap sort:2957
Count heap increase key:17147
Count heap insert:10000
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:2957
Count first quantile:0
Count third quantile:0
Time:4 second
```



**Figure 26:** Maximum with 100000 data

```
Count max heapify:16429986062
Count build max heap:29999
Count heap sort:29999
Count heap increase key:636656
Count heap insert:99996
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:29999
Count first quantile:0
Count third quantile:0
Time:444 second
```

### 3.6 First Quartile Function

**Figure 27:** First quartile with 10 data

```
Count max heapify:66
Count build max heap:5
Count heap sort:5
Count heap increase key:10
Count heap insert:10
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:5
Count third quantile:0
Time:0 second
```

**Figure 28:** First quartile with 100 data

```
Count max heapify:6819
Count build max heap:26
Count heap sort:26
Count heap increase key:116
Count heap insert:100
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:26
Count third quantile:0
Time:0 second
```

**Figure 29:** First quartile with 1000 data

```
Count max heapify:1010248
Count build max heap:296
Count heap sort:296
Count heap increase key:1429
Count heap insert:1000
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quartile:296
Count third quartile:0
Time:0 second
```

**Figure 30:** First quartile with 10000 data

```
Count max heapify:145517264
Count build max heap:3004
Count heap sort:3004
Count heap increase key:17197
Count heap insert:10000
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quartile:3004
Count third quartile:0
Time:4 second
```

**Figure 31:** First quartile with 100000 data

```
Count max heapify:16290699746
Count build max heap:29848
Count heap sort:29848
Count heap increase key:141036
Count heap insert:99996
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quartile:29848
Count third quartile:0
Time:452 second
```

### 3.7 Third Quartile Function

Figure 32: Third Quartile with 10 data

```
Count max heapify:33
Count build max heap:2
Count heap sort:2
Count heap increase key:10
Count heap insert:10
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:2
Time:0 second
```

Figure 33: Third Quartile with 100 data

```
Count max heapify:6675
Count build max heap:29
Count heap sort:29
Count heap increase key:115
Count heap insert:100
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:29
Time:0 second
```

Figure 34: Third Quartile with 1000 data

```
Count max heapify:1010116
Count build max heap:298
Count heap sort:298
Count heap increase key:1425
Count heap insert:1000
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:298
Time:0 second
```

**Figure 35:** Third Quartile with 10000 data

```
Count max heapify:137640616
Count build max heap:2884
Count heap sort:2884
Count heap increase key:17130
Count heap insert:10000
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:2884
Time:4 second
```

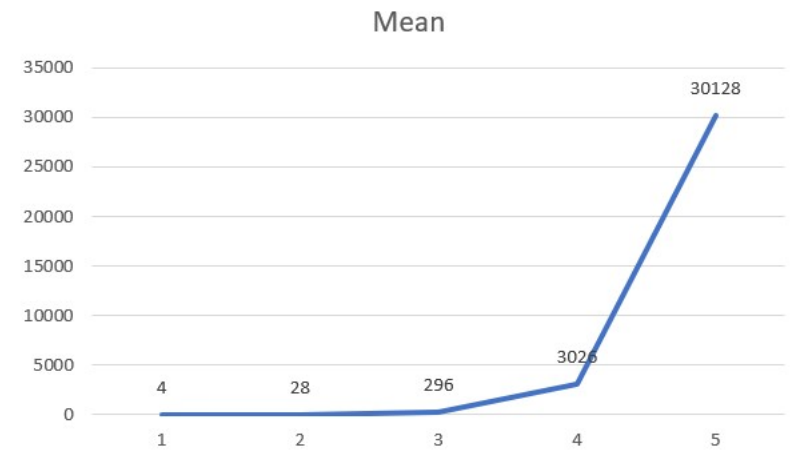
**Figure 36:** Third Quartile with 100000 data

```
Count max heapify:16434234196
Count build max heap:30090
Count heap sort:30090
Count heap increase key:141181
Count heap insert:99996
Count mean:0
Count standard deviation:0
Count median:0
Count minimum:0
Count maximum:0
Count first quantile:0
Count third quantile:30090
Time:456 second
```

## 4 Graphs

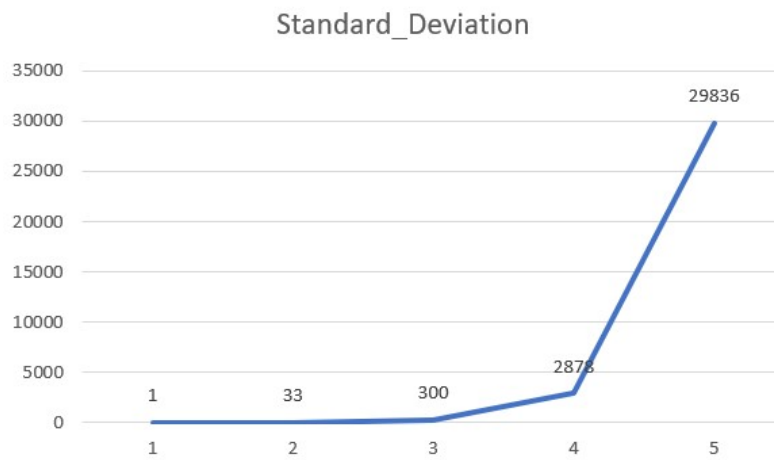
- Mean

**Figure 37:** Graph for Mean



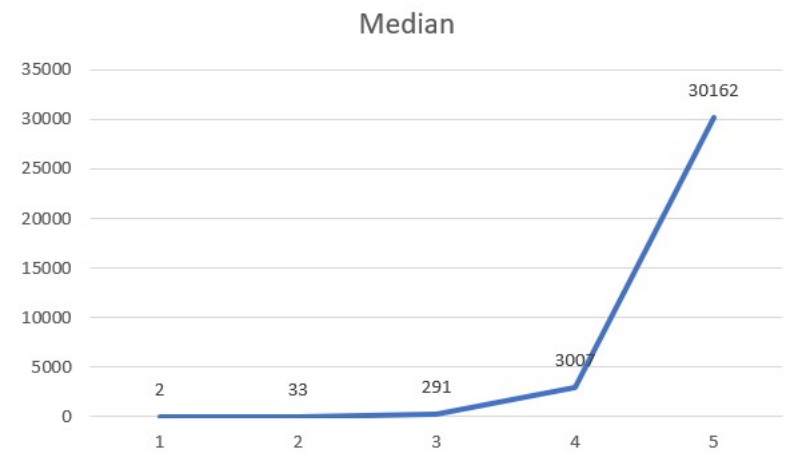
- Standard Deviation

**Figure 38:** Graph for Standard Deviation



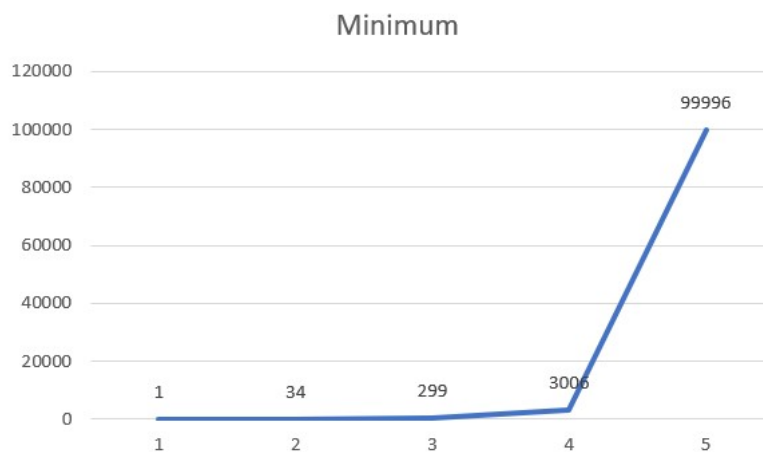
- Median

**Figure 39:** Graph for Median



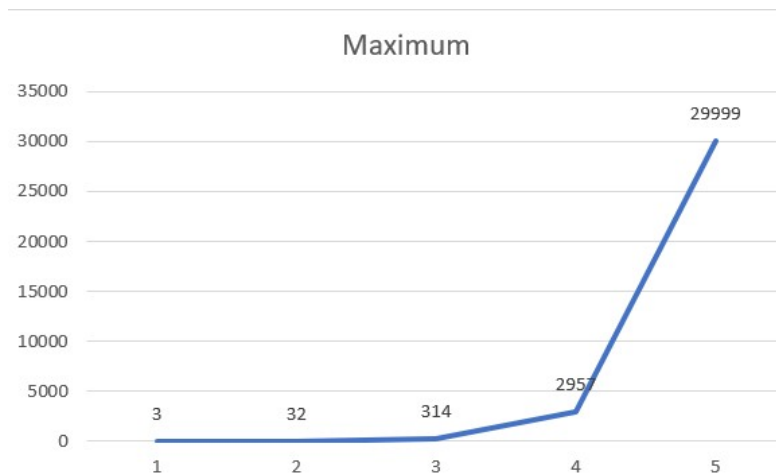
- Minimum

**Figure 40:** Graph for Minimum



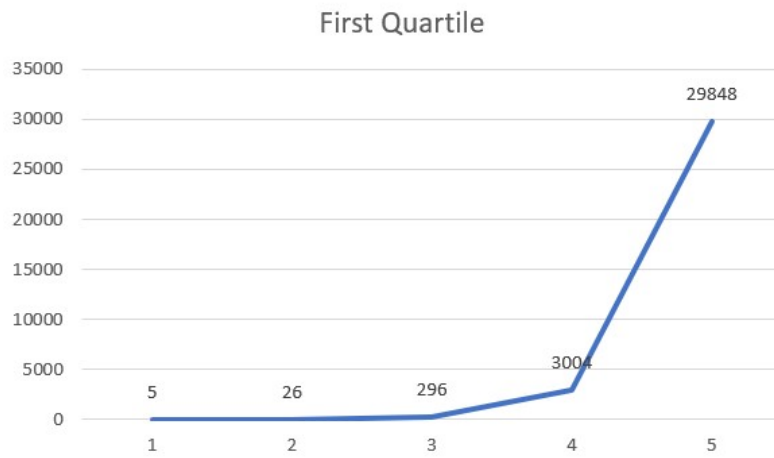
- Maximum

**Figure 41:** Graph for Maximum



- First Quartile

**Figure 42:** Graph for First Quartile



- Third Quartile

**Figure 43:** Graph for Third Quartile

