Istanbul Technical University
Faculty of Computer and Informatics
Computer Engineering Department

BLG 335E
The LaTeX
Report

Analysis of Algorithms I, Project 2

Leminur Çelik - 150190085

December 4th, 2022

Contents

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(lgn), 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(lgn).

Algorithm 1 Max heapify

```
function MAX-HEAPIFY(vector, i)
    left ← 2 * i
    right ← 2 * i + 1
    largest ← i
    if left ≤ size AND vector[left] > vector[i] then
        largest ← left
    end if
    if right ≤ size AND vector[right] > vector[largest] then
        largest ← right
    end if
    if largest ≠ 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(lgn). Calls are O(n). Therefore worst case running time is O(nlgn).

Algorithm 2 Build Max Heap

```
function BUILD-MAX-HEAP(vector)
    i ← size/2
    while i ≥0 do
        MAX-HEAPIFY(i)
        i ← 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(lgn) time. Since the function is inside the for loop, it will have O (nlogn) runtime. Therefore, the execution time for heap sort is O(nlgn).

Algorithm 3 Heap Sort

```
function HEAP-SORT(vector)
    BUILD-MAX-HEAP(vector)
    i ← (size − 1)
    while i > 0 do
        SWAP(vector[1],vector[i])
        size ← 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(lgn). The execution time of heap increase key function is O(lgn).

Algorithm 4 Heap Increase Key

```
function HEAP-INCREASE-KEY(vector, i)
    parent ← (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(lgn). Therefore, the execution time for heap insert is O(lgn).

Algorithm 5 Heap Insert

```
function HEAP-INSERT(vector, key)
    PUSH BACK(vector, key)
    size ← 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 ← 0
    while i < size do
        total ← total + heap[i]
        i ← i + 1
    end while
    mean ← 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 ← 0
    while i < size do
        total ← POW(heap[i] − mean, 2)
        i ← 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 ← 0 then
        median ← (heap[size/2] + heap[size/2 − 1])/2
    end if
    if size%2 ≠ 0 then
        median ← 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)(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 ← 0.25 × (size − 1)
    difference ← heap[CEIL(quartile)] − heap[FLOOR(quartile)]
    firstq ← (quartile − FLOOR(quartile)) × difference + heap[FLOOR(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)(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).

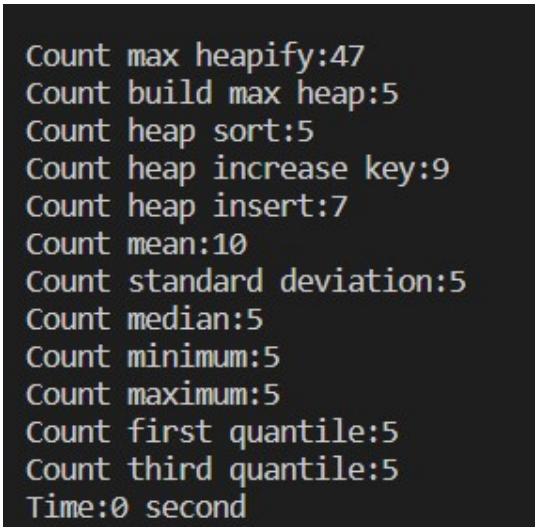
Algorithm 12 Third Quartile

```
function THIRD-QUARTILE(heap, size)
    quartile ← 0.75 × (size − 1)
    difference ← heap[CEIL(quartile)] − heap[FLOOR(quartile)]
    thirdq ← (quartile − FLOOR(quartile)) × difference + heap[FLOOR(quartile)]
    return thirdq
end function
```

3 Images

Screenshots of Count of Data Structures and Functions and Execution Time

Figure 1: Input1 File



3.1 Mean Function

Figure 2: Mean with 10 data

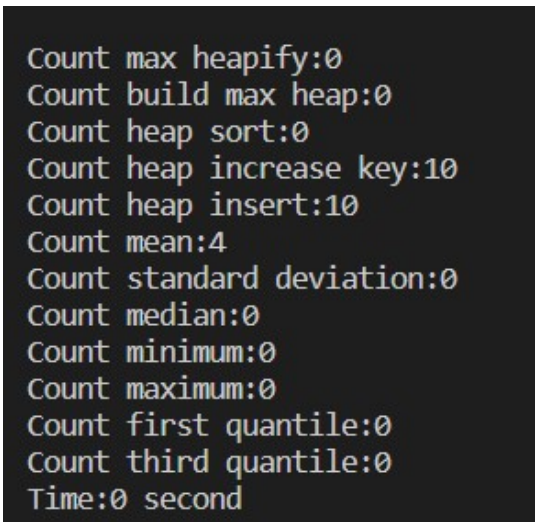


Figure 3: Mean with 100 data

```
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
```

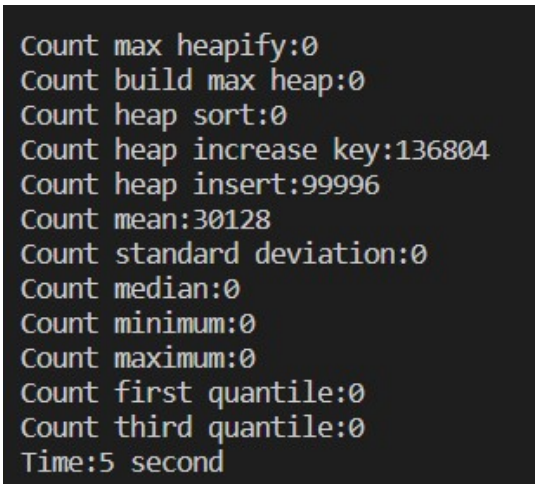
Figure 4: Mean with 1000 data

```
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



3.2 Standard Deviation Function

Figure 7: Standard Deviation with 10 data

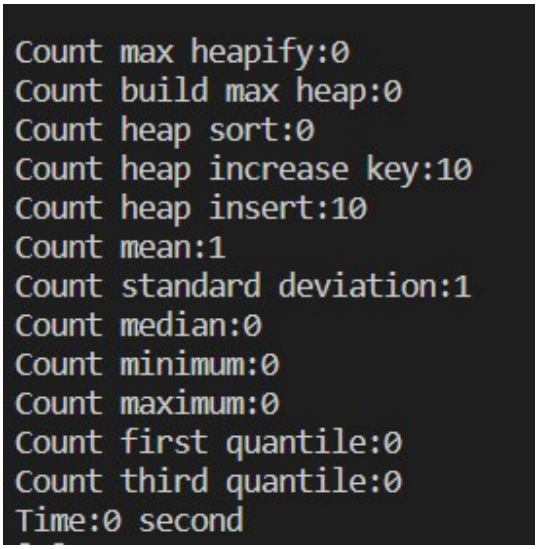


Figure 8: Standard Deviation with 100 data

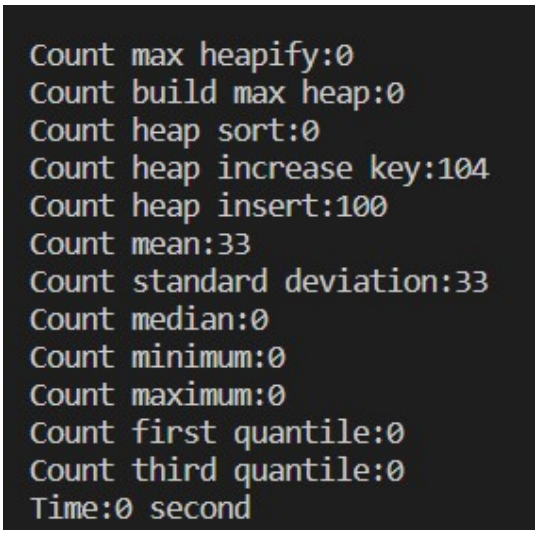


Figure 9: Standard Deviation with 1000 data

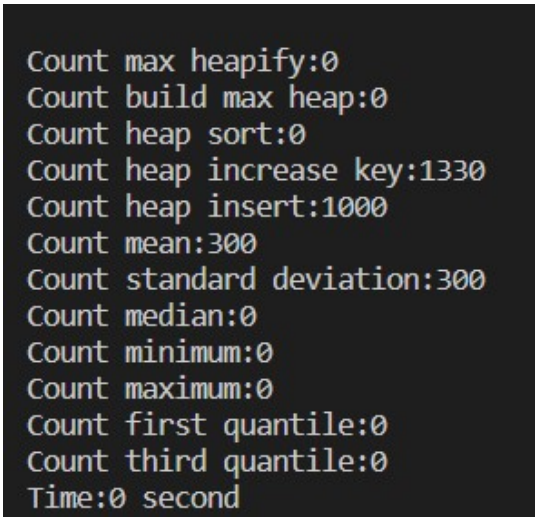


Figure 10: Standard Deviation with 10000 data

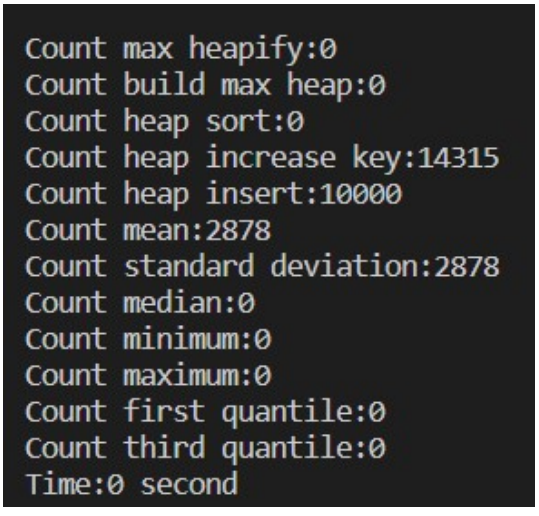
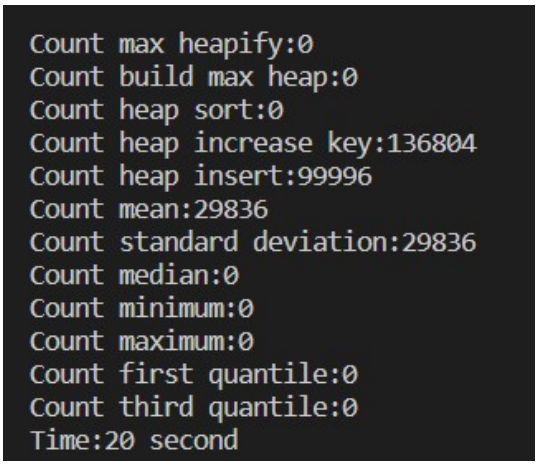


Figure 11: Standard Deviation with 100000 data



3.3 Median Function

Figure 12: Median with 10 data

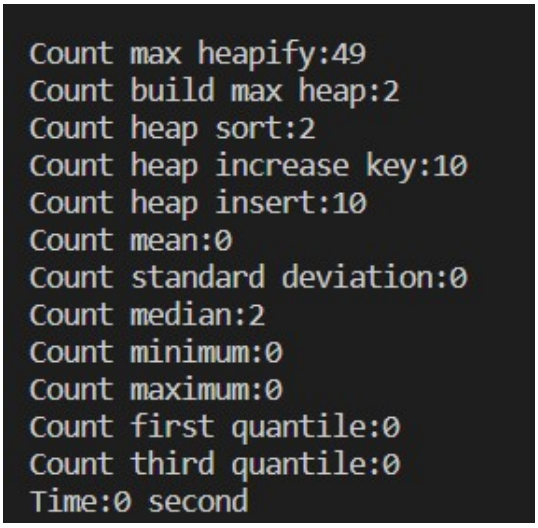


Figure 13: Median with 100 data

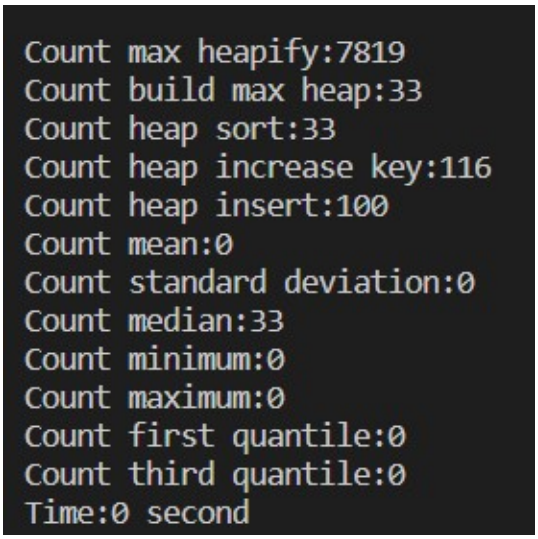


Figure 14: Median with 1000 data

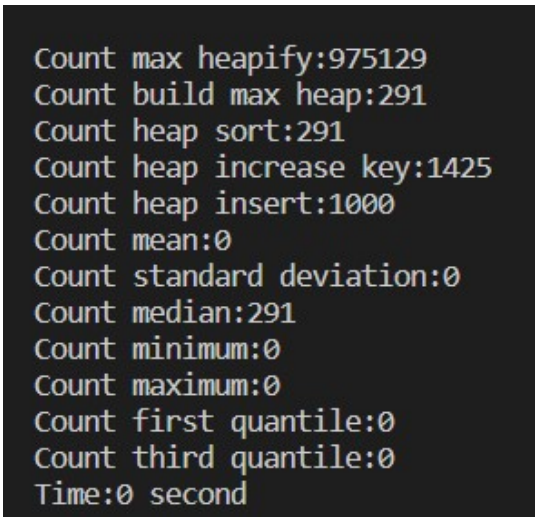


Figure 15: Median with 10000 data

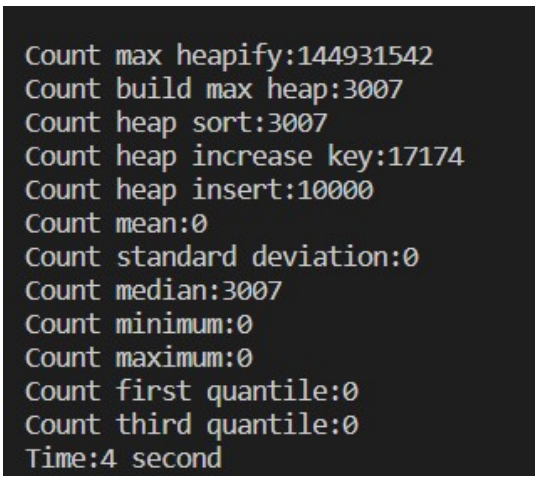
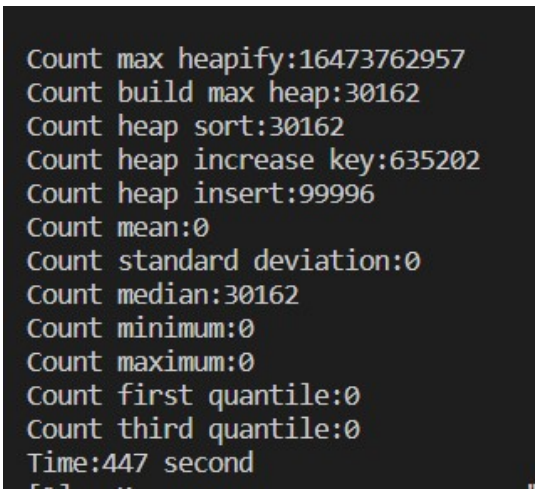


Figure 16: Median with 100000 data



3.4 Minimum Function

Figure 17: Minimum with 10 data

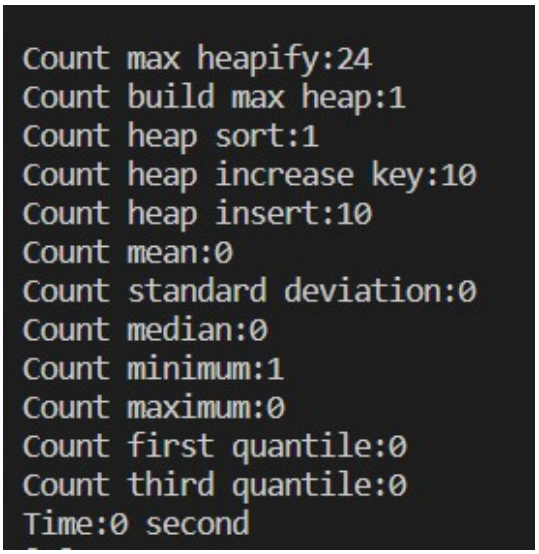


Figure 18: Minimum with 100 data

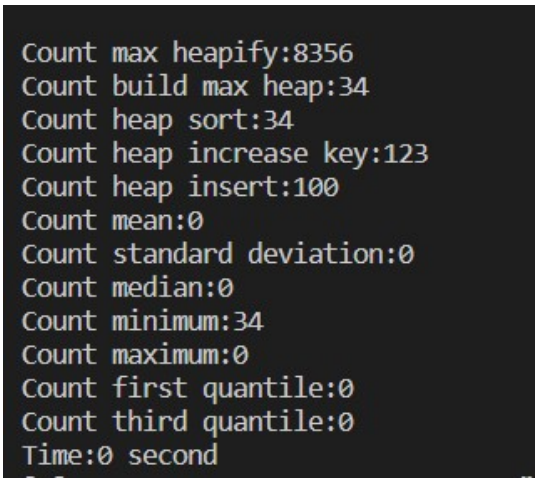


Figure 19: Minimum with 1000 data

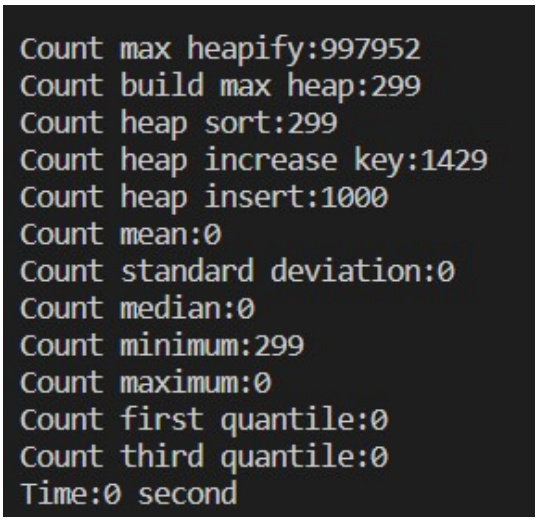


Figure 20: Minimum with 10000 data

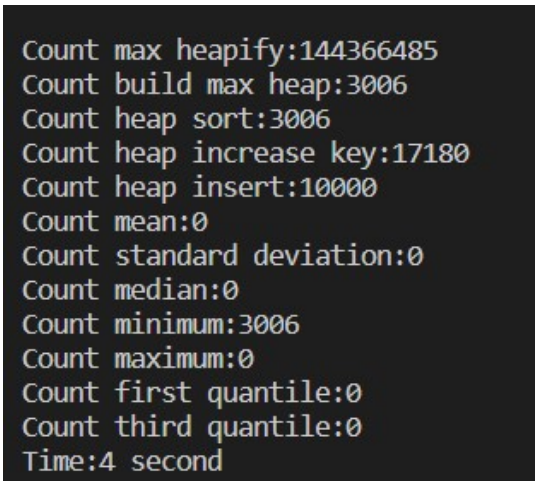
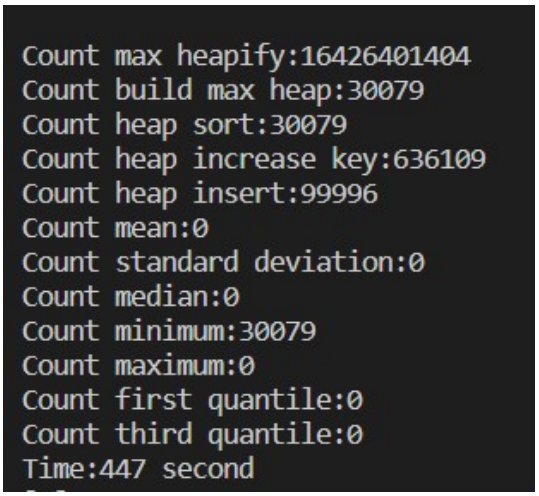


Figure 21: Minimum with 100000 data



3.5 Maximum Function

Figure 22: Maximum with 10 data

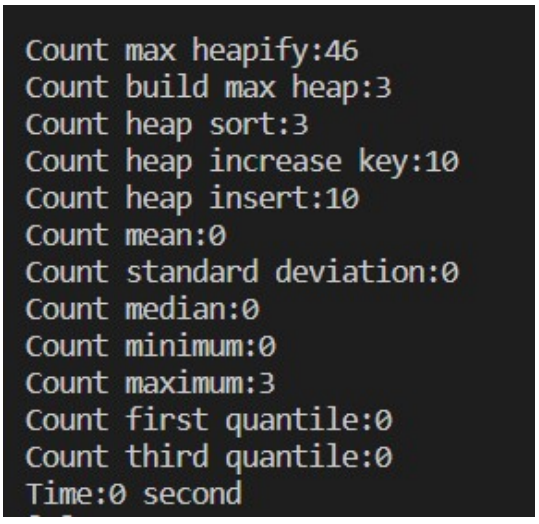


Figure 23: Maximum with 100 data

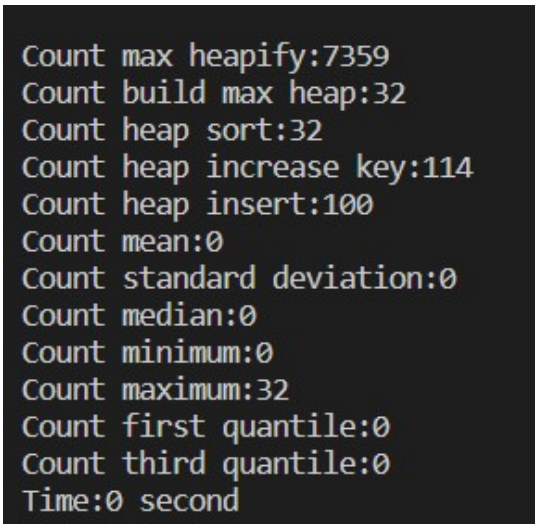


Figure 24: Maximum with 1000 data

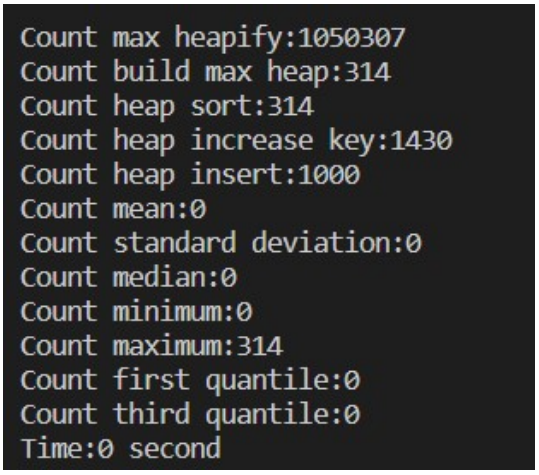


Figure 25: Maximum with 10000 data

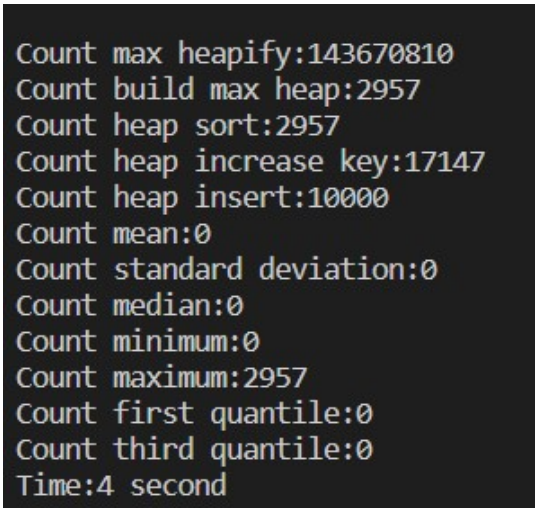
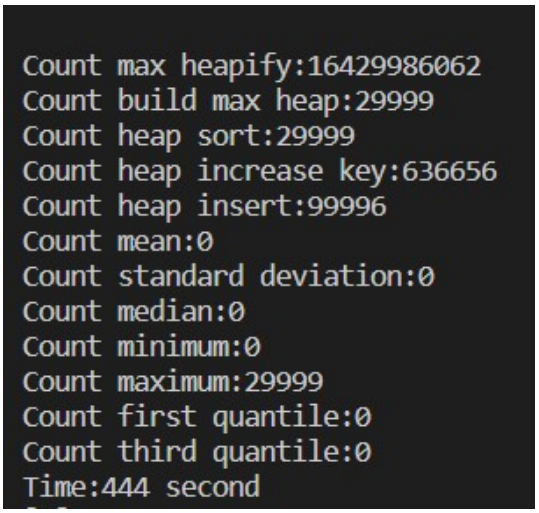


Figure 26: Maximum with 100000 data



3.6 First Quartile Function

Figure 27: First quartile with 10 data

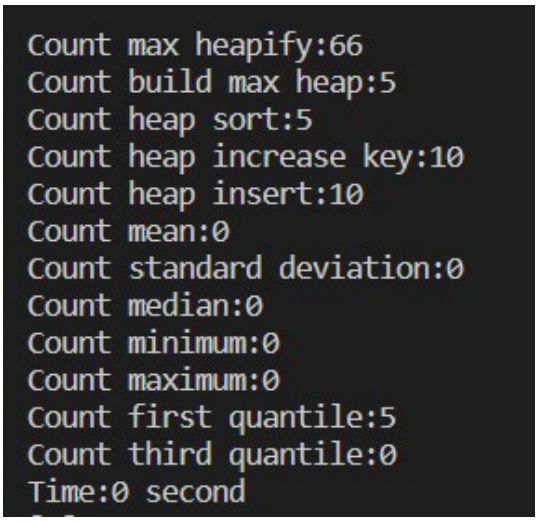


Figure 28: First quartile with 100 data

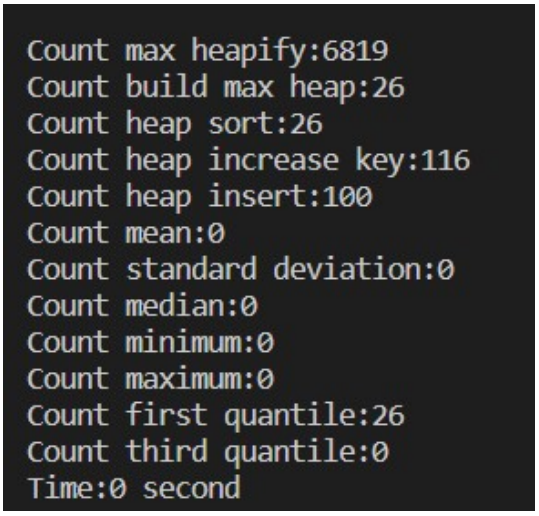


Figure 29: First quartile with 1000 data

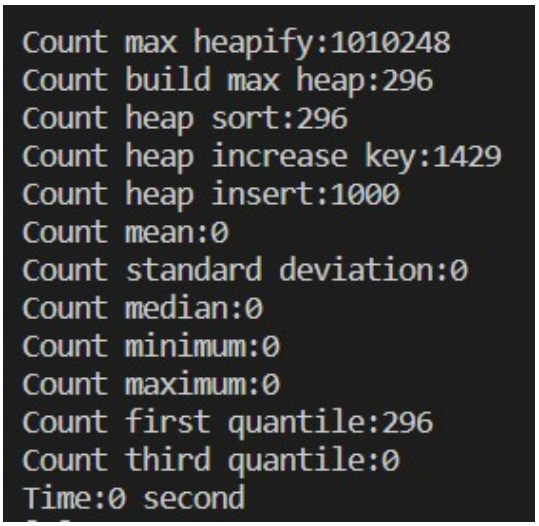


Figure 30: First quartile with 10000 data

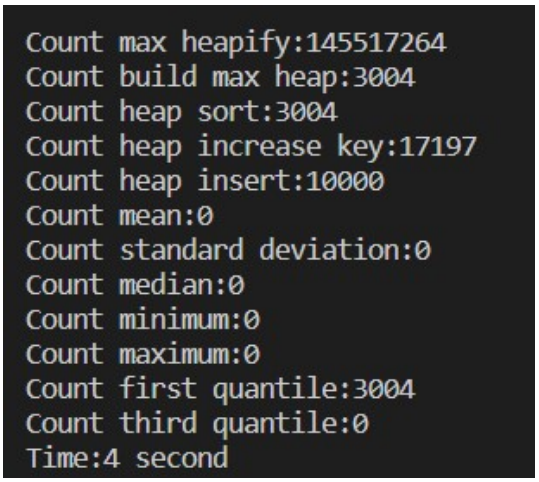
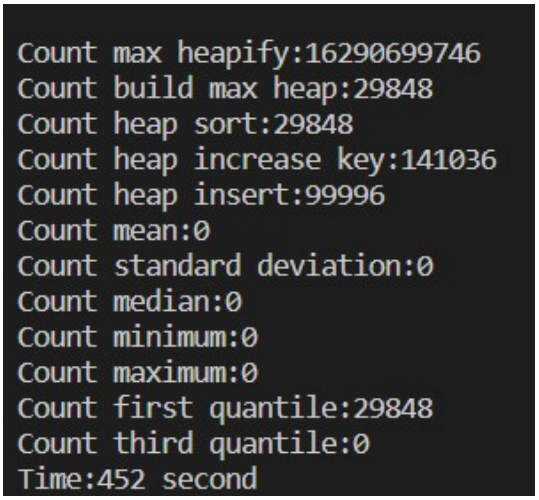


Figure 31: First quartile with 100000 data



3.7 Third Quartile Function

Figure 32: Third Quartile with 10 data

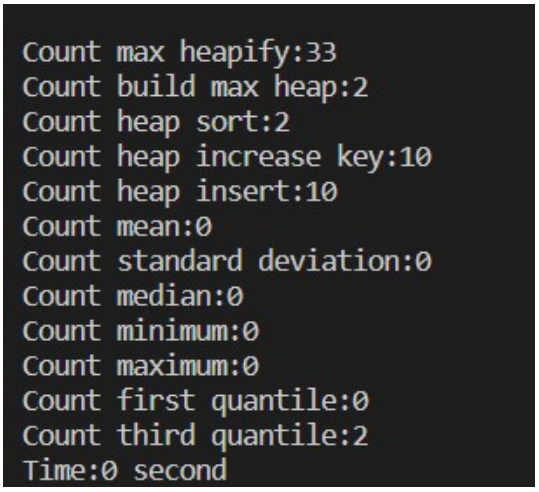


Figure 33: Third Quartile with 100 data

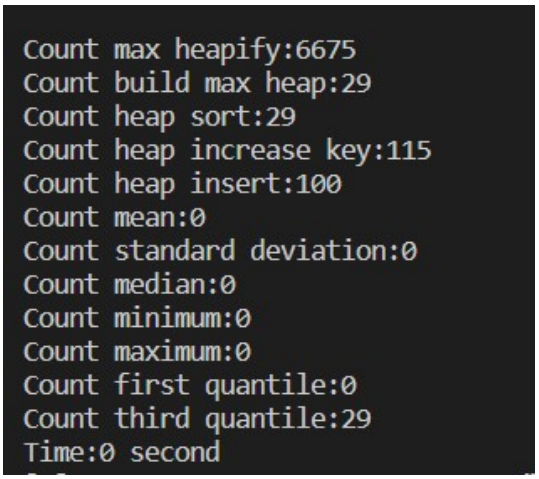


Figure 34: Third Quartile with 1000 data

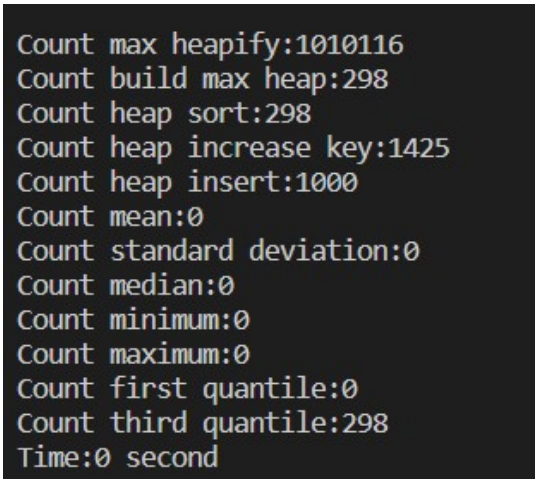


Figure 35: Third Quartile with 10000 data

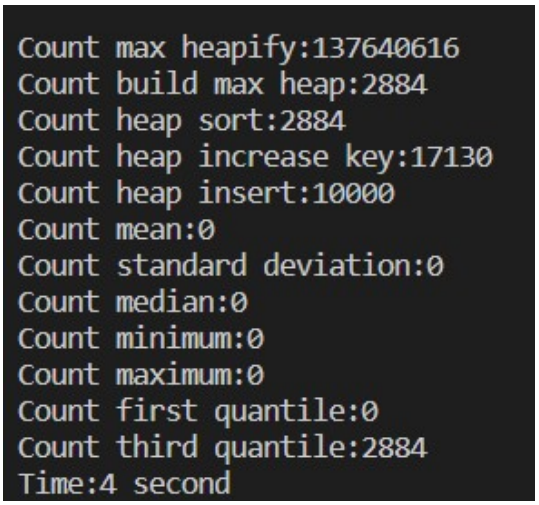
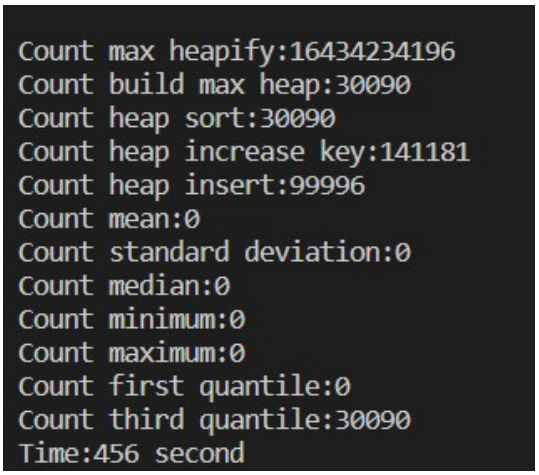


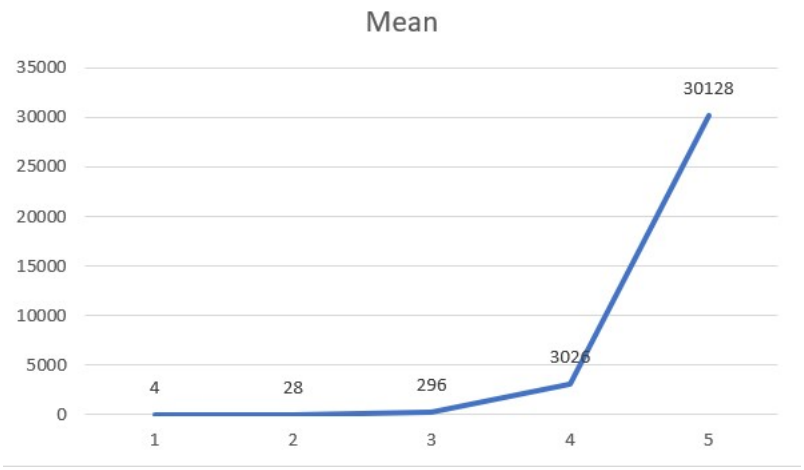
Figure 36: Third Quartile with 100000 data



4 Graphs

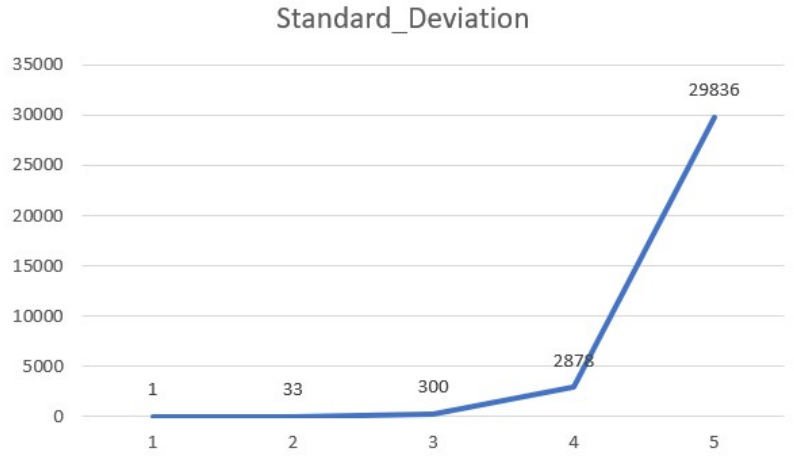
- Mean

Figure 37: Graph for Mean



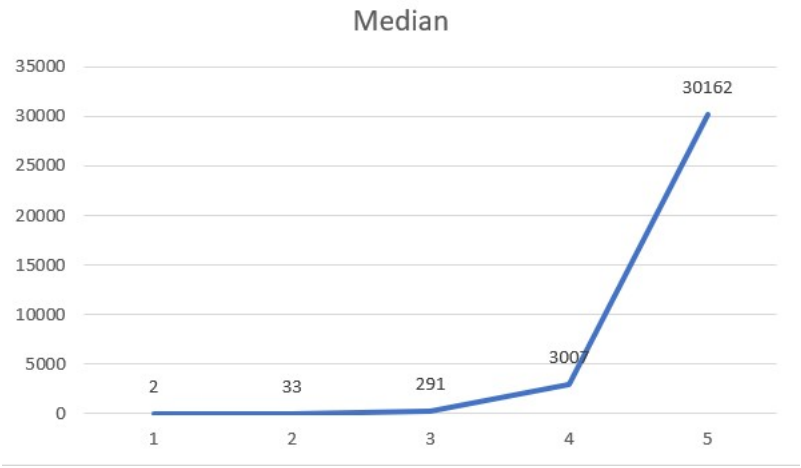
- Standard Deviation

Figure 38: Graph for Standard Deviation



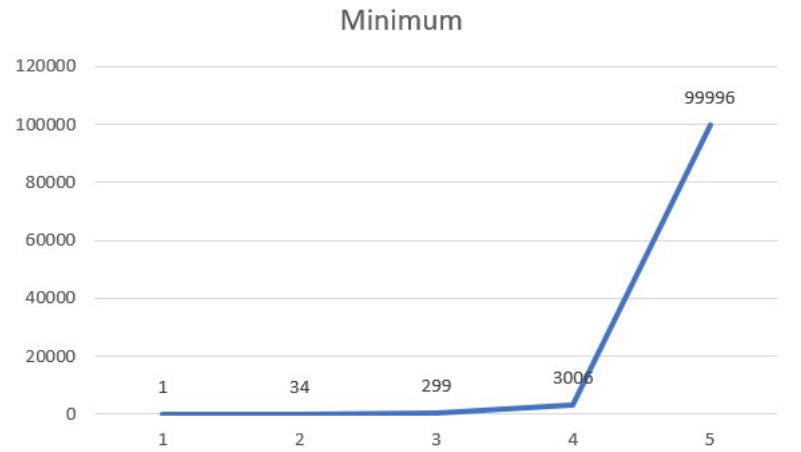
- Median

Figure 39: Graph for Median



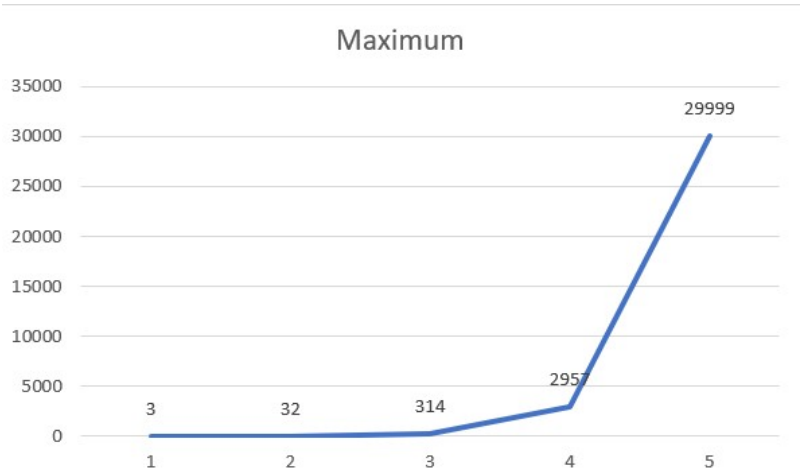
- Minimum

Figure 40: Graph for Minimum



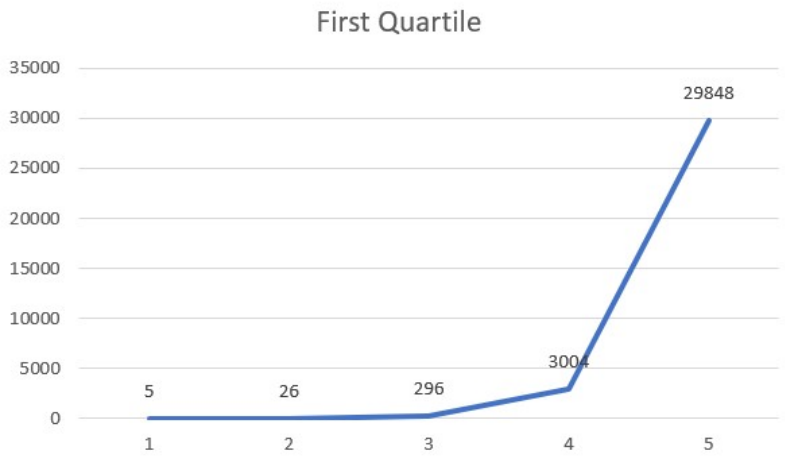
- Maximum

Figure 41: Graph for Maximum



- First Quartile

Figure 42: Graph for First Quartile



- Third Quartile

Figure 43: Graph for Third Quartile