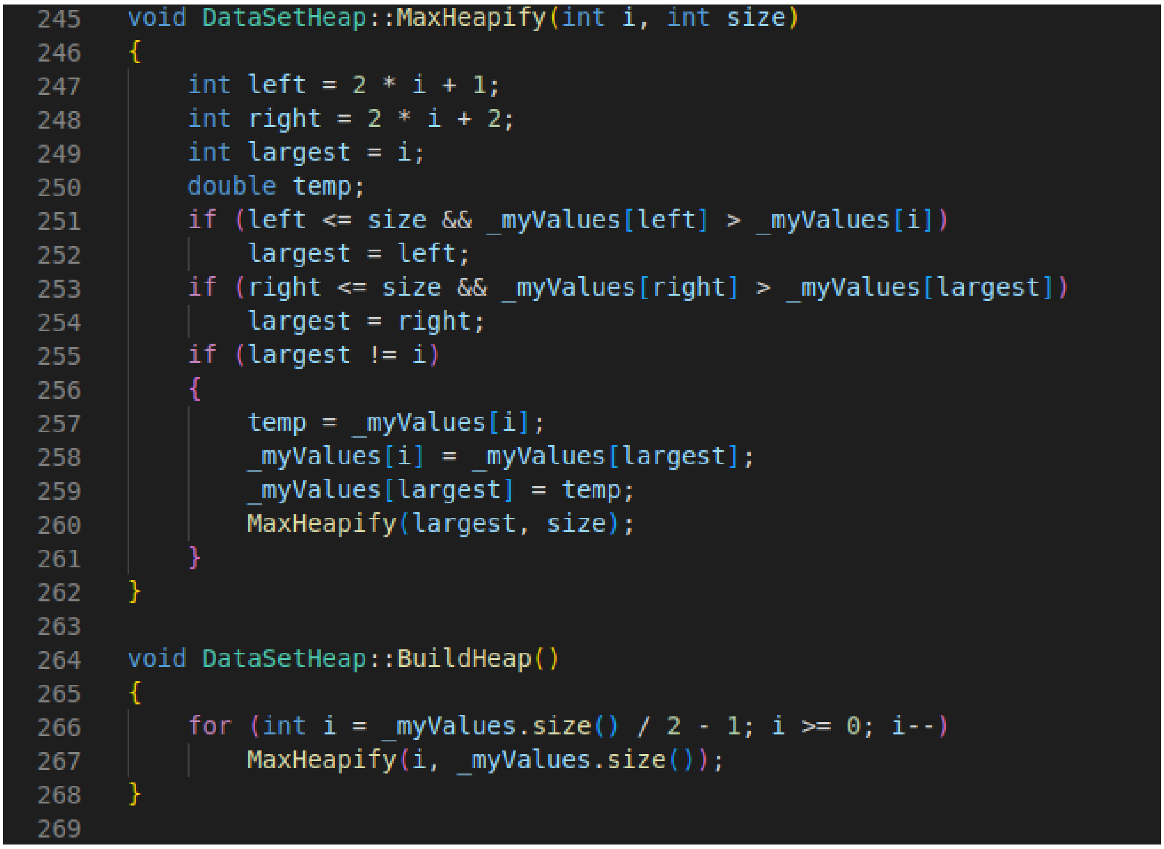
1 Introduction

This report is about Analysis of Algorithms I. Heap is used through homework and funtions are shown related to heap. Others functions are very clear in code.

2 My Codes

2.1 Heap



```
void DataSetHeap::MaxHeapify(int i, int size)
{
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    int largest = i;
    double temp;
    if (left <= size && _myValues[left] > _myValues[i])
        largest = left;
    if (right <= size && _myValues[right] > _myValues[largest])
        largest = right;
    if (largest != i)
    {
        temp = _myValues[i];
        _myValues[i] = _myValues[largest];
        _myValues[largest] = temp;
        MaxHeapify(largest, size);
    }
}

void DataSetHeap::BuildHeap()
{
    for (int i = _myValues.size() / 2 - 1; i >= 0; i--)
        MaxHeapify(i, _myValues.size());
}
```

Figure 1: Heapify

Max-heap data structure is used for this homework. It is a recursive function. Heap is builded just before ”print” command. Building heap is the trigger function to construct the heap. Complexity of this algorithm is O(nlogn). Psuedo code can be seen as following:

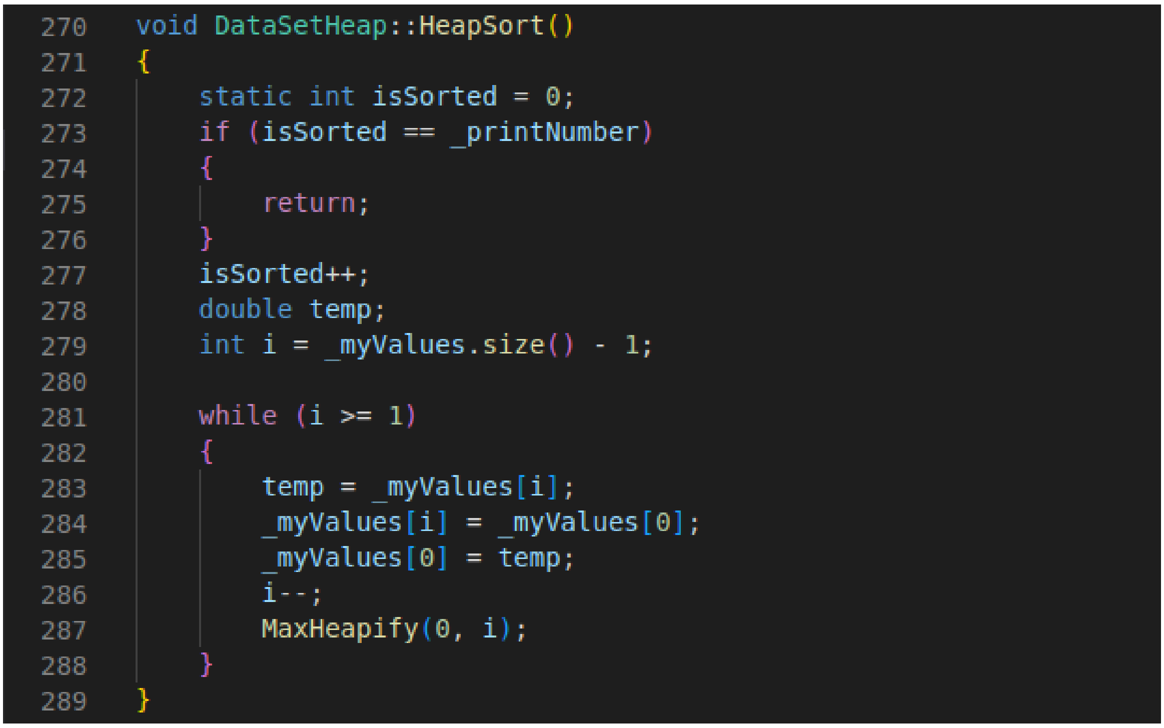
Algorithm 1 MAX-HEAPIFY

```
$l \leftarrow LEFT(i)$
$r \leftarrow RIGHT(i)$
$largest \leftarrow i$
**if** $l \leq heapsize[Vect]$ and $Vect[l] \geq A[i]$ **then**
    $largest \leftarrow l$
**end if**
**if** $r \leq heapsize[Vect]$ and $Vect[r] \geq Vect[largest]$ **then**
    $largest \leftarrow r$
**end if**
**if** $largest \neq i$ **then**
    swap Vect[i] and Vect[largest]
    MAX-HEAPIFY($largest$ , $Vect$)
**end if**
```

Algorithm 2 BUILD HEAP

```
$i \leftarrow heapsize[Vect]/2 - 1$
**for** $i$ >0 **do**
    $i \leftarrow i - 1$
    MAX-HEAPIFY($largest$, $Vect$)
**end for**
```

2.2 Sorting



```cpp
void DataSetHeap::HeapSort()
{
    static int isSorted = 0;
    if (isSorted == _printNumber)
    {
        return;
    }
    isSorted++;
    double temp;
    int i = _myValues.size() - 1;

    while (i >= 1)
    {
        temp = _myValues[i];
        _myValues[i] = _myValues[0];
        _myValues[0] = temp;
        i--;
        MaxHeapify(0, i);
    }
}
```

Figure 2: Sorting

For sorting purposes, heap sort is used. First part of the code is for checking whether the heap is sorted or not. Complexity is O(nlogn) Sorting algoritm is called in these three case, during execution of code:

- First Quantile
- Median
- First Quantile

Rest of the algorithm can be seen as psuedo code:

Algorithm 3 HEAPSORT

```
for $i \leftarrow heapsize[Vect]$ downto 1 do
    swap Vect[0] and Vect[i]
    $heapsize[Vect] \leftarrow heapsize[Vect] - 1$
    MAX-HEAPIFY(0 , $Vect$)
end for
```

2.3 Running Times

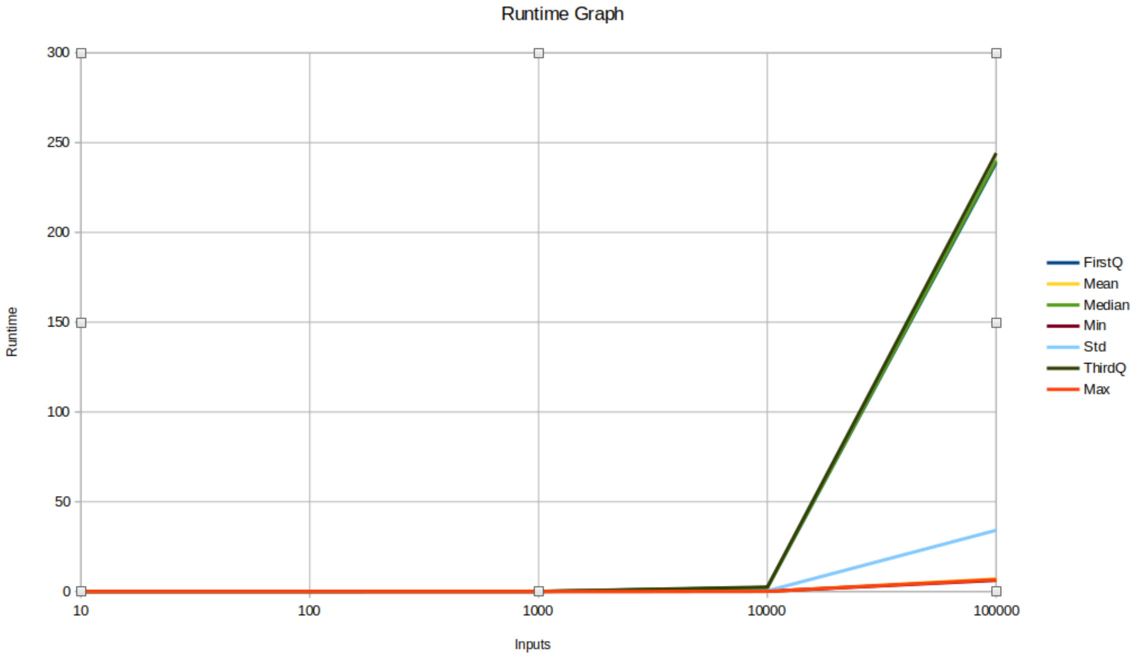
This table shows the average runtimes for any cases. Each case tested 10 times at least.

Table 1: Table of Running Times

input 1.txt	0.001 seconds
input firstq10.txt	0.001 seconds
input firstq100.txt	0.002 seconds
input firstq1000.txt	0.022 seconds
input firstq10000.txt	2.1 seconds
input firstq100000.txt	3m58.879 seconds
input max10.txt	0.001 seconds
input max100.txt	0.004 seconds
input max1000.txt	0.005 seconds
input max10000.txt	0.09 seconds
input max100000.txt	6.653 seconds
input mean10.txt	0.001 seconds
input mean100.txt	0.001 seconds
input mean1000.txt	0.02 seconds
input mean10000.txt	0.085 seconds
input mean100000.txt	6.985 seconds
input median10.txt	0.001 seconds
input median100.txt	0.002 mseconds
input median1000.txt	0.024 seconds
input median10000.txt	2.166 seconds
input median100000.txt	3m59.985 seconds
input min10.txt	0.001 seconds
input min100.txt	0.003 seconds
input min1000.txt	0.005 seconds
input min10000.txt	0.1 seconds
input min100000.txt	6.235 seconds
input std10.txt	0.002 seconds
input std100.txt	0.004 seconds
input std1000.txt	0.012 seconds
input std10000.txt	0.354 seconds
input std100000.txt	34.109 seconds
input thirdq10.txt	0.002 seconds
input thirdq100.txt	0.004 seconds
input thirdq1000.txt	0.029 seconds
input thirdq10000.txt	2.457 seconds
input thirdq100000.txt	4m04.164 seconds

2.4 Graphs

Figure 3: Graph


Runtime Graph
Runtime
Inputs
FirstQ
Mean
Median
Min
Std
ThirdQ
Max