CS590 homework 3 – Max, and Min Heaps

The due date for this assignment is Friday, Oct 19th, at 11.59pm. This assignment is worth 10% of your final grade.

Any sign of collaboration will result in a 0 and being reported to the Graduate Academic Integrity Board. Late submission policy described in the syllabus will be applied.

Questions (100 points)

1. You are given the design of two classes for the max, and min heap structures in the files *minHeap.h*, and *maxHeap.h*. Provide the implementation of these two classes in files *minHeap.cpp*, and *maxHeap.cpp* in order to satisfy all the required functionalities defined in the header files for each one of the functions. Note, that you cannot use any additional headers than the ones already included in the code provided.

2. Assume that you are given a stream of random numbers that you have to store. You are asked to find, and maintain the median value of these numbers, as new values are generated. The solution to this problem can be achieved with using two heaps, a min heap, and a max heap. The implementation of the heaps from problem 1 will be used. The outline of the code you will have to write for this, is provided in the *medianHeaps.h* file. Provide the implementation of this header file in the file *medianHeaps.cpp*. A test case has been provided in the *main.cpp* file, but feel free to write additional test cases to make sure your code works. (Additional test cases are not requested, and will not be graded). Note, that you cannot use any additional headers than the ones already included in the code provided.

Remarks:

- You are not allowed to use code from online resources. Your submission will be tested against that, and will receive a 0, and a report to the Graduate Academic Integrity Board if it is detected.
- No additional libraries are allowed to be used, besides the ones already included in the source code shared with you.
- Makefiles are provided for both problems to build the code in LinuxLab.
- The programming, and testing will take some time. Start early.
- Feel free to use the provided source code for your implementation. You have to document your code.

Stevens Institute of Technology

Department of Computer Science

CS590: ALGORITHMS

Homework Assignment 3

Submitted by:
Hadia Hameed
CWID: 10440803

Yijia Tan
CWID: 10427079

Submitted to:
Prof. Iraklis Tsekourakis

December 6, 2018

1 Problem I

You are given the design of two classes for the max, and min heap structures in the files minHeap.h and maxHeap.h. Provide the implementation of these two classes in files minHeap.cpp and maxHeap.cpp. In order to satisfy all the required functionalities defined in the header files for each one of the functions.

1.1 Constructor:

```
maxHeap::maxHeap(int cap){
     heapArray = new int [cap];
     heapSize = 0;
     capacity = cap;
}
```

1.2 insertKey(int k):

Algorithm 1 Insert a new key in the heap.

```
1: procedure INSERTKEY(INT K)
2:     if Heap size is smaller than the capacity of the array. then
3:         Create a new node i at the bottom of the heap and set its value to k
4:         while Node i is not the root and key at node i is less than key at parent of node i do
5:             swap node i and parent of node i
6:             i = parent of node i
7:         Increment the heap size by 1
8:     else
9:         Throw an error that the heap is full
```

```
void maxHeap::insertKey(int k){
if(heapSize < capacity){
     *(heapArray + heapSize) = k;
     int i = heapSize;
     while(i!=0 && (k > *(heapArray + parent(i)))){
     int temp = *(heapArray + i);
     *(heapArray + i) = *(heapArray + parent(i));
     *(heapArray + parent(i)) = temp;
     i = parent(i);
     }
     heapSize = heapSize + 1;
}
else{
     cout<<"Heap is full."<<endl;
}
}
```

1.2.1 Creating an Empty Heap:

```
int main()
{
    maxHeap h(4);
    h.insertKey(3);
    h.insertKey(4);
    h.insertKey(2);
    h.insertKey(16);
    h.displayHeap();

    return 0;
}
```

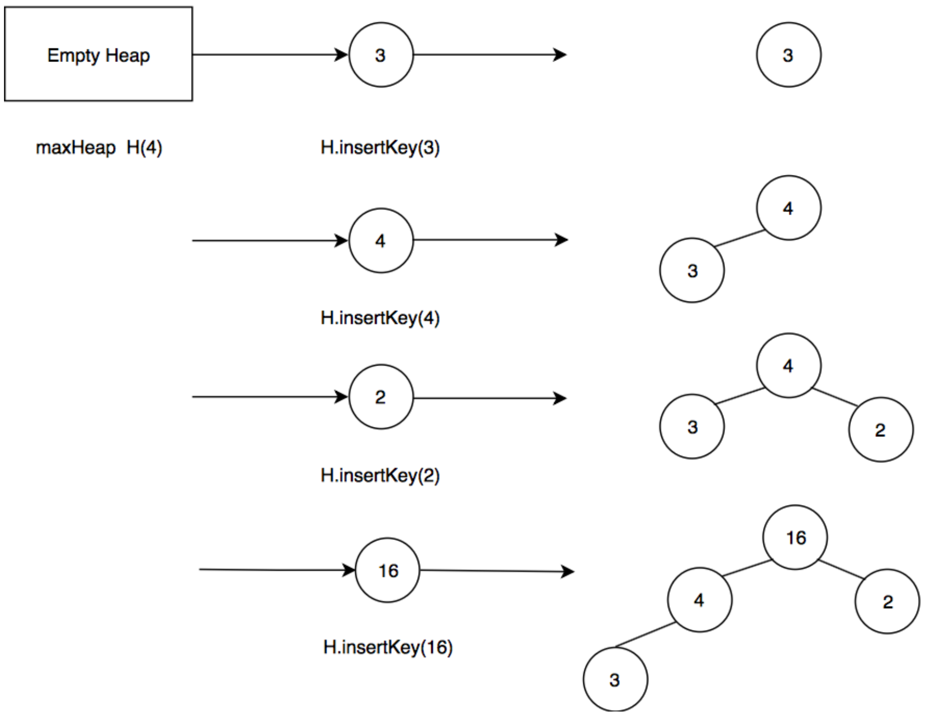

Figure 1: Initializing a heap

1.3 maxHeapify

Algorithm 2 Maintain the maxheap property.

```
1: procedure maxHeapify(int i)
2:     l = Left child of node i
3:     r = Right child of node i
4:     if l is less than heap size and key at node l is greater than key at node i then
5:         largest = l
6:     else
7:         largest = i
8:     if r is less than heap size and key at node r is greater than key at node i then
9:         largest = r
10:    if largest is not node i then
11:        swap the node i with node largest
12:        maxHeapify(largest)
```

```
void maxHeap::maxHeapify(int i){
     if (i >= heapSize){
       cout<<"Index i exceeds heap size."<<endl;
     }
     else {
       int* A = heapArray;
       int l = left(i);
       int r = right(i);
       int largest = 0;
       if(l < heapSize && *(A+l) > *(A+i))
          largest = l;
       else
          largest = i;
       if (r < heapSize && *(A+r)>*(A+largest))
          largest = r;
       if (largest != i ){
          int temp = *(A+largest);
          *(A+largest) = *(A+i);
           *(A+i) = temp;
           maxHeapify(largest);
       }
    }
}
```

1.4 deleteKey(int i):

Algorithm 3 Delete a key at index i in the heap.

```
1: procedure DELETEKEY(INT I)
2:     Check if the node i exists in the heap
3:     Copy value of i in variable index
4:     while Node i is not the last node do
5:         replace value in node i with value in node i+1
6:         i = i + 1
7:     maxHeapify(index)
8:     Decrement heap size by 1
```

```
void maxHeap::deleteKey(int i) {
    if (i >= heapSize)
        cout<<"Index exceeds heap size."
    else {
        int n = heapSize;
        int index = i;
        while(i<n-1){
          *(heapArray+i) = *(heapArray+i+1);
           i = i + 1;
     }
    maxHeapify(index);
    heapSize = heapSize - 1;
    }
}
```

1.5 extractMax()

Algorithm 4 Extract the maximum value from the heap.

```
1: procedure DELETEKEY(INT I)
2:     Check to see if the heap is not empty
3:     max = root of the heap
4:     newHeap = Copy the entire heap excluding the root in another array
                    and place the last element at the first position
5:     Replace the old heap with newHeap
6:     Decrement heap size by 1
7:     return max
```

```
int maxHeap::extractMax(){
    if(heapSize==0){
       cout<<"Heap underflow"<<endl;
        return 0;
    }
    else{
       int n = heapSize - 1;
       int max = *(heapArray);
       int * newData = new int[n];
       for ( int i = 0; i < n; i ++){
         if (i == 0)
           newData[i] = heapArray[n];
         else
           newData[i] = heapArray[i];
       }
       delete [] heapArray;
       heapArray = newData;
       maxHeapify(0);
       heapSize = heapSize - 1;
       return max;
    }
}
```

1.6 increaseKey(int i, int new_val)

Algorithm 5 Increasing value at node i to a higher value k.

```
1: procedure INCREASEKEY(INT I, INT NEW_VAL)
2:     Check to see if the heap is not empty
3:     Check to see if the new value is greater than the current value at node $i$
4:     Replace the current value at node $i$ with new value
5:     while Node $i$ is not the root and parent of $i$ is less than $i$ do
6:         swap node $i$ and its parent
```

```
void maxHeap::increaseKey(int i, int new_val){
   if(heapSize == 0){
     cout<<"Heap underflow"<<endl;
     return;
   }
   if (i >= heapSize){
     cout<<"index i is out of bounds"<<endl;
     return;
   }
   if(new_val < *(heapArray + i))
     cout<<"New key is smaller than the current key."<<endl;
   else{
     *(heapArray + i) = new_val;
     while(i > 0 && *(heapArray + parent(i)) < *(heapArray + i)){
       int temp = *(heapArray+i);
       *(heapArray+i) = *(heapArray+parent(i));
       *(heapArray + parent(i)) = temp;
       i = parent(i);
     }
   }
}
```

Note: *The functions for minHeap will be similar except the greater than signs will be reversed and the root would have the minimum value.*

2 Problem II

Assume that you are given a stream of random numbers that you have to store. You are asked to find, and maintain the median value of these numbers, as new values are generated. The solution to this problem can be achieved with using two heaps, a min heap, and a max heap. Provide the implementation of medianHeaps.h in the file medianHeaps.cpp.

```
int medianHeaps::getMedian()
{
    int medianMinHeap =0;
    int medianMaxHeap =1;
    int median;
    if (medianHeapSize%2 == 0){
        int half = medianHeapSize/2;
        for(int i=0;i<half;i++){
            medianMinHeap = minH->extractMin();
            medianMaxHeap = maxH->extractMax();
        }
        median = (medianMinHeap+medianMaxHeap)/2;
    }
    else{
        while(medianMinHeap!=medianMaxHeap){
        medianMinHeap = minH->extractMin();
        medianMaxHeap = maxH->extractMax();
    }
        median = medianMinHeap;
    }
    return median;
}
```

Algorithm 6 Finding median using min and max heaps.

```
procedure MEDIANHEAPS::GETMEDIAN()
    Add the random numbers in minHeap and maxHeap
    medianMinHeap = 0
    medianMaxHeap = 1
    if medianHeapSize is even then
        half = medianHeapSize/2
        i = 0
        while i < half do
            medianMinHeap = minHeap.extractMin()
            medianMaxHeap = maxHeap.extractMax()
            $i = i + 1$
        median = (medianMinHeap+medianMaxHeap)/2
    else
        while medianMaxHeap is not equal to medianMinHeap do
            medianMaxHeap = maxHeap.extractMax()
            medianMinHeap = maxHeap.extractMin()
        median = medianMinHeap
    return median
```