CMPSCI 187 (Spring 2018) Lab 11: Heap

The goal of this lab is to get familiar with the Heap data structure. To work on the assignment:

- Go to ***File -> Make a Copy*** to make an editable copy of this Google Doc for your account
- Follow the instructions to complete the assignment
- When you are done, go to ***File -> Download As -> PDF Document***
- Log in to Gradescope and submit your PDF

The lab is due by 5:00 pm today. Please make sure that you submit to Lab 10, not Project!

Section 1: Fill in the Blanks [3 x 2 points = 6 points]

To begin, briefly review the Heap lecture slides, particularly what's the definition of a heap (a max-heap by default), what are its properties, and what's bubbleUp and bubbleDown.

Generally, we bubble up or down the node whenever the max-heap rule is violated. We bubble up or down a node not only when we add or remove the node in the heap, but also when we modify the value of a certain node in the heap. This is because when a node value is changed in the heap, parent values might no longer be bigger than children values. When this rule is violated, we need to bubble up or down to restore it to a valid heap. Now answer the following two questions:

1. **In a max-heap, if a node's value is increased, we need to apply bubble up** (bubble up or down?).
If a node's value is decreased, we need to apply bubble down (bubble up or down?).

2. Given the following valid heap stored in an array:

9	**5**	**8**	**4**	**3**	**7**	**6**	**2**	**1**

we will modify some nodes' values in the following two questions. **You need to choose to apply either bubble up or down to restore it to a valid heap**, and write down the resulting array accordingly.

1. If the node with value 5 (the second element in the array) is changed to value 0, apply bubble up or down to restore it to a valid heap, and write down the result.

9	4	8	2	3	7	6	0	1

2. Starting from the original heap again (NOT the result of part a): if the node with value 1 (the last element) is changed to 8, restore it to a valid heap and write down the result.

9	8	8	5	3	7	6	2	4

Section 2. MaxHeap (14 points)

Given the following Heap data structure, complete the three methods below. **Starter code is provided** to you so that you can compile and verify your code. Instead of using Eclipse, you can copy and paste the starter code to any **online Java compiler** and test your code that way.

```
class MaxHeap<T extends Comparable<T>> {
      private T[] heap = null;  // array storing heap elements
      private int size = 0;     // number of elements
```

A. Write a method called `set`, which sets a new value to the element at a given index, and then call either bubbleUp or down to restore the heap to be valid. [4 points]

```
public void set(int index, T value) {
    if(index >= size) return;
    T prev = heap[index];
    heap[index] = value;
    if (value.compareTo(prev) > 0)
        bubbleUp(index);
    if (value.compareTo(prev) < 0)
        bubbleDown(index);
}
```

B. Write a recursive version of the `bubbleUp` method. The parameter is the index of an element that needs to be bubbled up. You are allowed to create a private helper method if you decide that the public `bubbleUp` method does not contain enough parameters for recursion. Do NOT use any loop (`for, while` etc.) otherwise you will get 0 point. [5 points]

```
public void bubbleUp(int index) {
    int parent = (index - 1) / 2;
    if (index > 0 && heap[index].compareTo(heap[parent]) > 0) {
        T temp = heap[index];
        heap[index] = heap[parent];
        heap[parent] = temp;
        bubbleUp(parent);
    }
}
```

C. Write a recursive version of the `bubbleDown` method. The parameter is the index of an element that needs to be bubbled down. You are allowed to create a private helper method if you decide that the public `bubbleDown` method does not contain enough parameters for recursion. Do NOT use any loop (`for, while` etc.) otherwise you will get 0 point. [7 points]

```
public void bubbleDown(int index) {
    int next = greaterChildOf(index);
    if (next != -1 && heap[index].compareTo(heap[next]) < 0) {
        T temp = heap[index];
        heap[index] = heap[next];
        heap[next] = temp;
        bubbleDown(next);
    }
}

private int greaterChildOf(int index) {
    int left = 2 * index + 1;
    int right = 2 * index + 2;
    if (left >= size) {
        if (right >= size)
              return -1;
        else
              return right;
    } else if (right >= size) {
        return left;
    } else {
        return heap[left].compareTo(heap[right]) > 0 ? left : right;
    }
}
```