Homework 9

Happy Halloween!

ECE 309 Fall 2019

Due: October 30, 2019

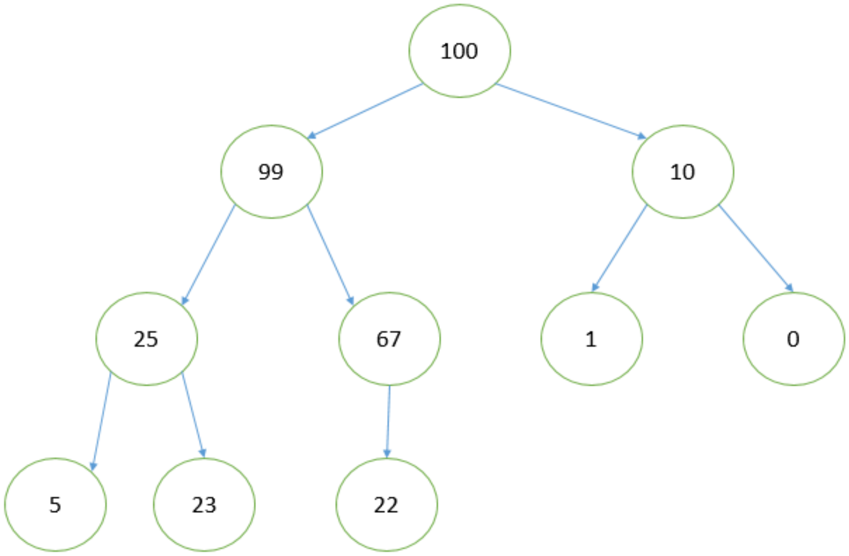
Upload an electronic copy of your answers to Moodle under HW9.

*This is a shared google document. This means (1) it may change to clarify content, and (2) other people can view your comments on this file. If you have questions, you are encouraged to comment directly on this document, but **do not add your answers here**. Make a copy into your private Google Drive and then edit the document.*

DO NOT ADD ANSWERS TO THE SHARED DOC! THAT'S CONSIDERED CHEATING!

1. Max-heap

a. [25 points] Draw a max-heap two ways, once as an array and once as a tree (nodes and edges), after all of the following insertions:
   ○ 5, 10, 99, 23, 22, 1, 0, 67, 25, 100



b. [15 points] Consider the following max-heap. What does it look like after removing the maximum value 3 times? Show the max-heap in array form after each removal.

100	75	25	66	22	19	11	50
75	66	25	50	22	19	11	
66	50	25	11	22	19		
50	22	25	11	19			

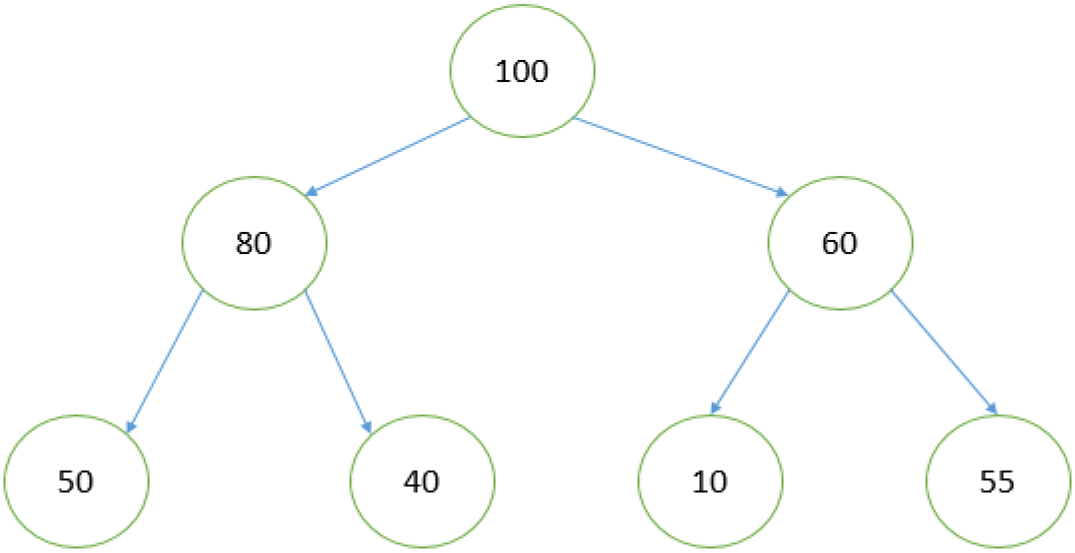
c. [15 points] The typical approach taken in the max-heap remove function moves the last element of the max-heap into the root and then percolates it down into position. What if we take a different approach that avoids moving the last element of the max-heap into the root. Instead, we move up the larger of the root's two children. Then, we repeat this on the node that was moved all the way down the tree stopping once we reach the last level of the tree. At the last level, there will be a leaf whose value was moved up. We move the last element of the max-heap into that vacated position to fill the hole.

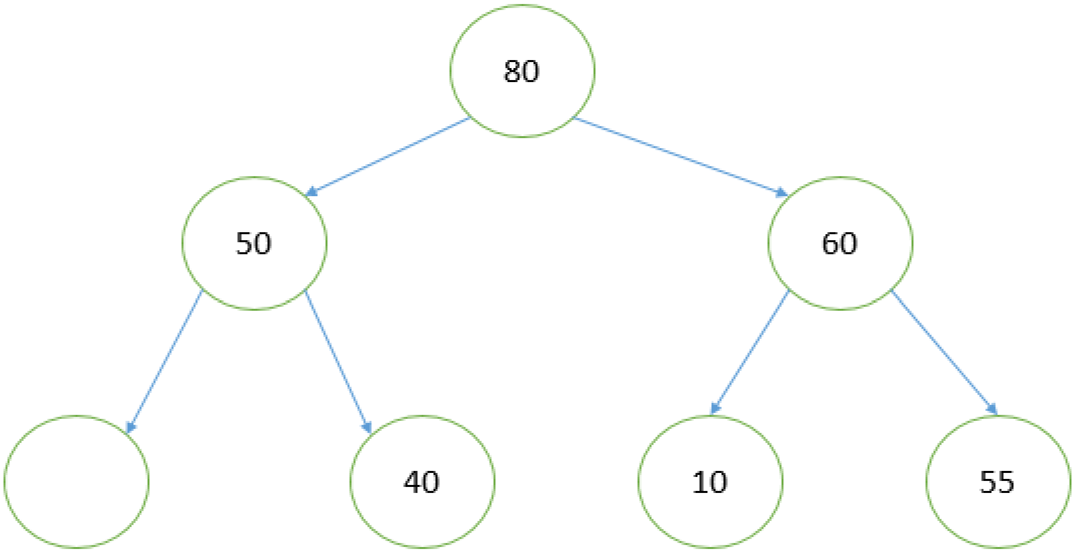
Here's the pseudocode:

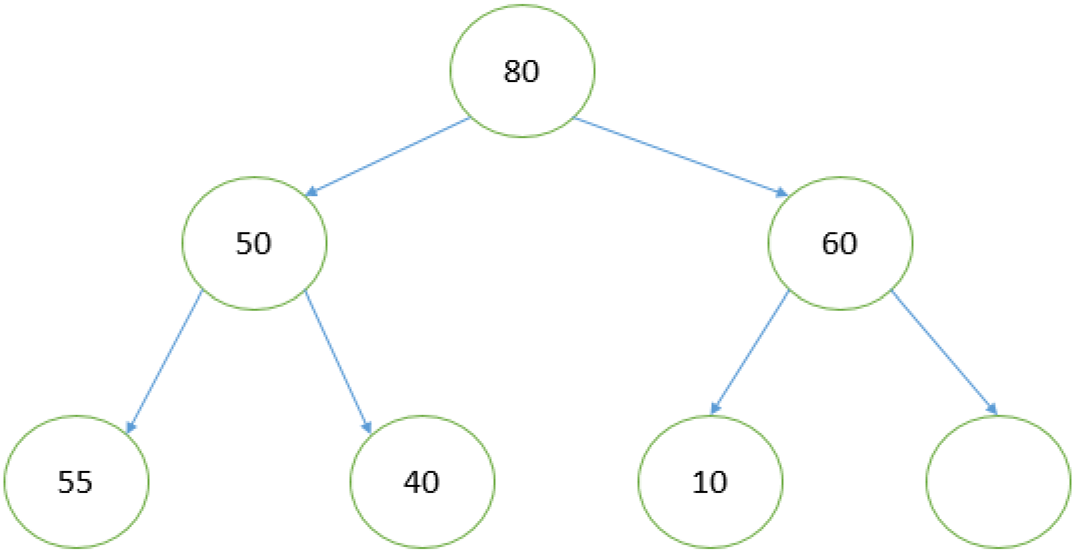
```
otherPercolateDown(index):
   while (index is not a leaf node):
      bigger = get index of child with larger value
      // if only one child, bigger is index for sole child
      heapArray[index] = heapArray[bigger]
      index = bigger
   // index is now a leaf node that's empty
   // move the last element into heapArray[index]
   heapArray[index] = heapArray[nextIndex--]
```

Does `otherPercolateDown` work correctly? If it does, is it faster than the typical `percolateDown` in terms of big-O? If it doesn't work, give an example max-heap where it fails to work properly.

It doesn't work correctly.


100
80
60
50
40
10
55


80
50
60
40
10
55



2. ZyLabs

(a) [25 points (as reported by ZyLab)] ZyLab 11.30. Validate that a string of parentheses and braces is properly balanced, in other words that every open parentheses/bracket has a matching closing one.

Here are some examples of balanced and unbalanced strings:

Balanced	Unbalanced
(5+3)+1	(
()()	)
({} ()) ()	({)}
{ { () } } []	(()))
[2 * (4+ {1-1})]	)(

Hint: to solve this problem you need to remember the reverse order of open parentheses/brackets/braces that you've already seen. Which data structure helps you remember things in reverse?

You should support parentheses (), braces {}, and brackets []. Other characters may be ignored. Strings may be of arbitrary length. You must implement the following function, read the comments below for additional specific requirements:

```
// Return true if str contains a balanced sring of (), {} and [].
// Otherwise, return false.
// Parameters:  str is the string to analyze.
//              error: if returning false, pass by reference the
//              position of the first unbalanced symbol. If the
//              string ends before seeing a matching
//              symbol, return the length of str.
bool isBalanced(std::string str, int &error)
{
   // IMPLEMENT
   return false;
}
```

(b) [20 points] Validate that BST's nodes are in the proper order by traversing the tree and validating the order among a node and its children. *You are required to implement this as single recursive function. Do not add other helper functions.* Use the following declarations:

```
class BTNode {
public:
  int data;
  BTNode *left;
  BTNode *right;
  BTNode(int d,BTNode *l=nullptr,BTNode *r=nullptr)
    :data(d),left(l),right(r)
  {}
};

bool isLegalBST(BTNode *subtree)
{
   // IMPLEMENT
   return false;
}
```

Consider a null tree as properly ordered, even though it contains no nodes.

- [5 points] Implementation must be a single recursive function. If it's not, you lose these points.

- [15 points] As reported in ZyLabs provided that you modify the template. Some test cases pass with the default template, but if you do not change the template code, you will earn 0 points.