CMPSCI 187 Final Spring 2018

Name:_____ Student ID:_____

Section 1. Short Answer (4 x 10 = 40 points)

1. What is the worst case complexity of this code in terms of **N**?

```
public static int f(int N) {
        if (N == 0) { return 1; }
        else { return f(N-1) + f(N-1); }
}
```

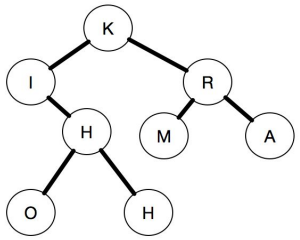
O ( )

2. In a **max-heap** of size N with no duplicates, what range of array indexes could contain the **minimum** element?

_____ <= i <= (N -1)

3. The largest element in a BST is always a leaf. (**True or False**)

4. What is the **post-order** traversal of the following binary tree?



5. Starting with an array of size n, of numbers in some random order, if you do the following, you end up with a proper max-heap.(**TRUE/FALSE**).

for (int i=0; i < n; i++) { bubbleDown(i); }

6. Suppose that we have an undirected graph with N vertices, such that the graph is **connected**. What is the smallest number of edges the graph could have?

7. If you consider all comparison-based sorting algorithms, what is the best worst-case complexity in terms of an input size of N? (Read that carefully!)

O( )

8. The following hashtable has a **capacity of 10**, and __ marks empty slots. The index of each slot is shown below for convenience. Using linear probing (probe length = 1), a new key **26** will be inserted into which slot? **[60 __ __ 53 23 __ 36 46 27 __]**

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

9. What's the output of the code below?

```
int[] a = { 5, 3, 1, 9, 6, 2, 7 };
int pivotValue = 4;
int storeIndex = 0, t = 0;
for(int i=0; i<a.length; i++) {
  if(a[i] < pivotValue) {
    t = a[i];
    a[i] = a[storeIndex];
    a[storeIndex] = t;
    storeIndex ++;
  }
}
System.out.print(a[2]);
```


10. If there are N items, the bubble sort makes this many comparisons:

O( )

Section 2. Heap (20 points)

Fixing a buggy heap. Let's say you are given an array representation of a heap. But as it turns out, there are some nodes in the heap that may not actually be in the right place, specifically they might be smaller than one of their children. Write a method to fix the buggy heap.

- **Do NOT create new methods.**
- **Do NOT create any new data structures such as arrays.**
- **Do NOT add instance/class variables**

Assume you have access to implementations of bubbleUp(i) and bubbleDown(i) and swap(a,b) which can swap two elements of the heap[]. There are nElems in the heap. Try to be as efficient and elegant as possible (fewer comparisons and lines of code is better).

```
int heap[];
int nElems;

public void fixBuggy(){




}
```

Section 3. Linked Lists and BSTs (20 points)

Construct a **sorted** linked list (smallest to largest) from a BST using recursion.

- **Do NOT create new methods.**
- **Do NOT create any new data structures such as arrays.**
- **Do NOT add instance/class variables**
- **Do NOT use any other LinkedList methods.**
- **Do NOT assume the BST is non-empty**
- **Your solution must be O(n)**

```
class LLNode<T> {
  public T data;
  public LLNode<T> next;
  public LLNode(T data) { this.data = data;}
}
class BSTNode<T> {
  public T data;
  public BSTNode<T> left, right;
  public BSTNode(T data) { this.data = data;}
}

public class LinkedList<T>{
  LLNode<T> head, tail;
  /* DO NOT ADD ANY INSTANCE VARIABLES, You can make local variables. */

  private void constructSortedList(BSTNode<T> node){









  }

  public LinkedList(BSTNode<T> root){
    constructSortedList(root);
  }
}
```

Section 4. Search (20 points)

Print out the data every node in a graph in the order of its minimum distance from the starting point. There are no duplicates in this graph. Print out the start node first.

- **Do NOT create new methods.**
- **Do NOT add instance/class variables**
- **Do NOT use any other Graph methods.**
- **Do NOT use recursion**
- **You may use one other data structure, such as a Queue<T> or a Stack<T>**
- **Use System.out.println(vertex) to print out the data of each node.**

```
public class Graph<T> {
 boolean isEmpty();
 boolean isFull();

 void clearVisits();
 void visitVertex(T vertex);
 getNextUnvisitedNeighbor(T vertex); // will return null if all neighbors have been
visited
 .
 .
 .
}
```

HINT: Stop and think about what kind of search would work best here.

```
public void printNodes (T start) {
  clearVisits();









}
```