CMSC132 Summer 2018 Midterm 2

Name (PRINT): _____

Instructions

- This exam is a closed-book and closed-notes exam.
- Total point value is 100 points.
- The exam is a 80 minutes exam.
- Please use a pencil to complete the exam.
- **WRITE NEATLY**. If we cannot understand your answer, we will not grade it (i.e., 0 credit).

1. T/F and Multiple Choice	/ 25
2. Short Answer	/ 45
3. Programming	/ 30
Total	/100

1. True / False and Multiple Choice

1)[1 pt] T / F Worst case insertion for Binary Tree is always better than worst case insertion for a sorted linked list

2)[1 pt] T / F It is possible to write every recursive function iteratively

3)[1 pt] T / F For a binary tree, A preorder traversal will always be different from a postorder traversal

4)[1 pt] T / F In an array-based implementation of a binary heap, where indices start at one, the parent of the k^{th} node is always at index k / 2.

5)[2 pts] The postOrder traversal of a binary search tree is {1, 12, 4, 22, 18, 16}. What is the new postOrder traversal after the insertion of 10 and 14?

a) 1, 12, 4, 22, 18, 16, 14, 10
b) 1, 10, 12, 14, 4, 22, 18, 16
c) 1, 10, 14, 12, 4, 22, 18, 16
d) 1, 12, 4, 10, 14, 22, 18, 16

6)[2 pts] Stack uses ____ paradigm, whereas the queue uses ____ paradigm.

a) FILO, LIFO
b) FIFO, LILO
c) LILO, FILO
d) LIFO, FIFO

7)[2 pts] The cost of inserting or removing an element to/from a binary search tree(BST) is O(N), where N is the total number of elements in the tree. The reason for that is:

a) BST keeps its entries sorted
b) BST is balanced.
c) BST is not always balanced.
d) BST is a version of Red-Black trees

8) [2 pts] Red & Black Tree guarantees searching in _____time.

a) O(log n)
b) O (n)
c) O (1)
d) O (n log n)

9) [2 pts] What is the output of **`fun(3)`**?

```
void fun(int n){
  if (n == 0){
      System.out.print("-");
  }else{
      System.out.print(n);
     fun(n - 1);
     System.out.print(n) ;
  }
}
```

a) 321-
b) 321-123
c) -123
d) Runtime Error

i.

10) [2 pts] What does the following function do for a given Linked List with first node as head?

```
int foo(Node head)
{
  if(head == null)   return 0;
  return 1 + foo(head.next);
}
```

a) Returns the sum of all nodes.
b) Adds 1 to the key of each node
c) Returns the length of the linked list
d) Adds 1 to the key in the head node

11) [2 pts] What is the worst-case time complexity for insert and remove operations in a Binary Heap?

a) O(n) for all
b) O(log n) for all
c) O(log n) insert, and O(n) for remove
d) O(log n) for remove, and O(n) for insert

12) [2 pts] What is the output of the following code fragment?

```
public static void main(String[] args) {
      Stack<Integer> stack = new Stack<Integer>();
      int n = 12;
      while (n > 0) {
            stack.push(n%2);
            n = n/2;
      }
      String result = "";
      while (!stack.isEmpty())
            result += stack.pop();
      System.out.println(result);
}
```

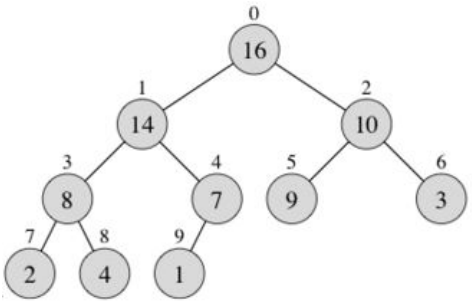
a) 1101
b) 0011
c) 0010
d) 1100

13)[3 pts] The following numbers are inserted into an empty binary search tree in the given order: 10, 2, 3, 5, 14, 1, 15, 12. What is the height of the binary search tree (the height is the maximum distance (number of edges) of a leaf node from the root, height of a tree with one node is 0.)?

a) 5
b) 3
c) 4
d) 2

14) [2 pts] In the max binary heap shown below, when **`remove()`** is called, which node (use the node key) will replace the deleted node before sink operation is called.

a) 14
b) 2
c) 1
d) 3



2. Short Answer

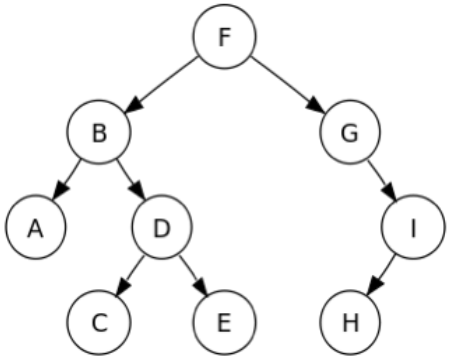
1) [3 pts] Computing a fibonacci sequence recursively can be very inefficient because we must calculate each term over again several times. For example, fib(10) would be called millions of times to compute fib(50). Explain how we can speed up this calculation without using iteration.

2) [3 pts] What are the **two** advantages of using a array over a singly-linked list?

3) [3 pts] Fill in the worst case time complexities of the insert operation for each of the following structures:

Structure	Binary Search Tree	Red-Black Tree	Binary Heap
Time Complexity	O( )	O( )	O( )

4) [6 pts] Write the preorder, inorder, and postorder traversal of the following binary tree:



preOrder	
inOrder	
postOrder	

5) [3 pts] What does this function do?

```
int foo(Node r) {
     if (r == null) return 0;
     if (r.left == null && r.right == null) return 0;
     return 1 + foo(r.left) + foo(r.right);
}
```

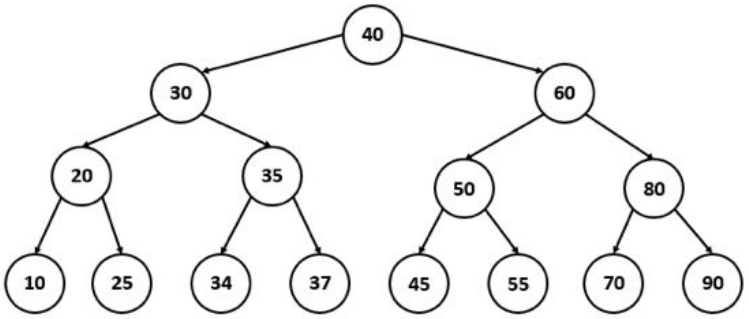
Answer:

6) [3 pts] Suppose that you do **binary search** for the key 82 in the following sorted array of size 15:

10 11 25 31 36 39 53 55 56 64 68 75 78 82 87.

Give the sequence of keys in the array that are compared with 82 Answer:

7) [3 pts] Draw the Binary Search Tree after you delete key 60.



8)[3 pts] Given the following contents of an array implementation of a stack, where 50 is at the top of the stack.

0	1	2	3	4	5
20	30	40	50		

Show the contents of the stack and the location of top after doing the following
stack.push(60);
stack.pop();
stack.push(70);
stack.push(80);
stack.pop();
stack.push(90)

0	1	2	3	4	5

9) [3 pts] Given the following contents of a circular array implementation of a queue

0	1	2	3	4	5
20	30	40			

first last

Show the contents of the queue and locations of **`first`** and **`last`** after doing the following:

Queue.dequeue();
Queue.enqueue(50);
Queue.dequeue();
Queue.enqueue(60);
Queue.enqueue(70);
Queue.dequeue();
Queue.enqueue(80);
Queue.enqueue(90);

0	1	2	3	4	5

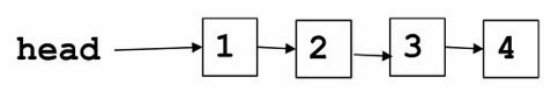
10) [6 pts] Construct a left-leaning red-black tree when the following elements are inserted in this order: 10, 15, 20, 18, 16. Show your steps for each number. Use dashed line for red edge.

11) [3 pts] If the given postOrder traversal for a **binary search tree** is {10,19,14,31,42,35,27}, construct the binary search tree.

12) [3 pts] Assume linked list

Function **foo** is defined as follows:

```
void foo(Node head){
  if(head == null) return;
  foo(head.next);
  System.out.print(head.data);
  System.out.print(head.data);
}
```



What is the output of **`foo(head)`**

13) [3 pts] Construct a max binary heap where the values are inserted in the following order: 10, 40, 60, 20, 50, 70, 30.
Show each step for partial credit (Hint: you need to “swim”)

3. Programming

[8 pts] A binary tree node is defined as

```
class Node{
      Integer key;
      Node left, right;
}
```

write the method "**`boolean sameShape(Node r1, Node r2)`**", which receives two trees and returns if the given two trees have the same shape. Same shape trees have same left, right nodes, but the keys may not be same. The following two trees have same shape.


```
boolean sameShape(Node r1, Node r2){




}
```

2) [8 pts]Write a method "`int countNodes(Node r, int min, int max)`" that takes a ***binary tree*** and a range and returns how many nodes fall within that range inclusive

```
int countNodes(Node r, int min, int max){

}
```

3) [6 pts] A singly-linked list node is defined as:

```
public class Node {
     int val;
     Node next;
     Node(int x) { val = x; }
}
```

Write a method "`Node removeNthNode(Node root, int n)`" that given a linked list, remove the *n*-th (starts at 0) node from the list and return its head.

```
Node removeNthNode(Node root, int n){

}
```

4) [8 pts] A binary search tree (BST) nis defined as

```
public class BST<Key extends Comparable<Key>, Value> {
      private Node root;
      private class Node {
      private Key key;
      private Value value;
      private Node left, right;
      }
}
```

Write a method "`Value search(Node h, Key key)`", which received a binary search tree and a key as arguments, and returns the value of the given key if the key exists. It returns null if the key does not exist. This method can be either recursive or iterative and must use binary search.

```
Value search(Node h, Key key){




}
```

END OF EXAM

Extra Questions

Suppose you had a grid of height *h* and width *w*. You are positioned at the bottom left box and you must get to the top right box and you can only move up or to the right. Write a recursive method to determine how many paths there are given any sized grid.
Examples: a 1x1 grid has 1 path, a 2x2 has 2 paths, a 3x3 has 5 paths, a 2x3 grid has 3 paths, etc

Solution:

```
public int size(int h, int w) {
	if (h < 0 || w < 0)
		return 0;   //out of bounds
	if (h == 0 && w == 0)
		return 0;   //base case
	return 1 + size(h-1, w) + size(h, w-1);
}
```

The following questions refer to the definition of a Node provided here:

```
class Node {
	int key;
	Node left, right;
	Node(int key) {
		this.key = key;
		this.left = null;
		this.right = null;
	}
}
```

Write a method "`int countNodes(Node r, int min, int max)`" that takes a ***binary tree*** and a range and returns how many nodes fall within that range inclusive

Solution:

```
public int countNodes(Node r, int min, int max) {
	if (r == null)
```

```
            return 0;
      int res = countNodes(r.right, min, max) + countNodes(r.left, min, max);
      if (r.key <= max && r.key >= min)
            res++;
      return res;
}
```

Write a method "`Node deleteAll(Node r, int threshold)`" that takes a ***binary search tree*** and a threshold and deletes all nodes with keys lower than the threshold.

Solution:

```
public Node deleteAll(Node r, int threshold) {
      if (r == null)
            return r;
      if (r.key >= threshold)
            r.left = deleteAll(r.left, threshold);
      else {
            r.right = deleteAll(r.right, threshold);
            r = r.right;
      }
      return r;
}
```

```
/**
 * Definition for singly-linked list.
 * public class Node {
 *     int val;
 *     Node next;
 *     Node(int x) { val = x; }
 * }
 */
```

Write a method "removeNthFromEnd(Node root, int n)" that given a linked list, remove the *n*-th node from the end of list and return its head.

Solution:

```
 public Node removeNthFromEnd(ListNode head, int n) {
        Node curr = head;
        Node peak = head;
        // if there is only one node in the Linked-List.
```

```
        ListNode prev = null;
        int i = 0;
```

```
 while (i < (n - 1)) {
            i++;
            peak = peak.next;
        }

        if (peak == null) {
            return head;
        }
        curr = head;
        peak = peak.next;
        i = 0;
        while (peak != null) {
            peak = peak.next;
            prev = curr;
            curr = curr.next;
        }

        if (curr == head) {
            return curr.next;
        } else {
            prev.next = curr.next;
            return head;
        }
    }
```

Suppose you have the below definition for a Node for a binary tree. Please write an insert method that will produce a left-complete tree.
Hint: Use a level order traversal to find the correct location to insert a Node. You may use a LinkedList or ArrayList for this purpose

```
public class BinaryTree<E> {
      Node root;
      private class Node {
            E data;
            Node left, right;
```

```
            Node (E data) {
                  this.data = data;
            }
      }

      public void insert(T data) {

      }

}
```

Solution

```
public void insert(E data) {
      if (root == null)
            root = new Node(data);
      else {
            LinkedList<Node> list = new LinkedList<Node>();
            list.add(root);
            insert(data, list);
      }
}

private void insert(T data, LinkedList<Node> list) {
      while (!list.isEmpty())
            Node n = list.remove(0);
            if (n == null) {
                  //this should never happen, if it does, we can't do anything
            }
            if (n.left == null) {
                  n.left = new Node(data);
                  return;
            } else if (n.right = new Node(data)) {
                  n.right = new Node(data);
                  return;
            } else {
                  list.add(n.left);
                  list.add(n.right);
            }
}
```

1) The doubly-linked list is advantageous over singly-linked list because accessing the end of the list with a tail reference takes O(n) time. T / F

2) A Red & Black Tree is a BST that sometimes has a node with two red edges connected to it. T / F

3) The order in which elements are inserted into a binary tree will affect the time it takes to retrieve them T / F

4) A height method that recursively compares the height of the left and right subtrees is an O(n) operation T / F

5) A balanced BST has a height of O(n), whereas an unbalanced BST has a height of O(nlogn) T / F

6) In an array-based implementation of a binary heap (starting from index 1), the left-child index of the k^{th} node is 2 * k and that of the right child is (2 * k) + 1 T / F

Which is something you would not use a stack for?

a) Reversing a word
b) Breadth-First Search
c) Depth-First Search
d) Backtracking (ie. edit->undo)

Which of the following computes the height of a ***complete*** binary tree where k is the number of nodes in the tree? (Assume all log values are truncated and a tree with one node has height 1)

a) $\log_2(k + 1)$
b) $\log_2(k) + 1$
c) $\log_2(k)$
d) Cannot determine

Which of the following traversals are sufficient enough by themselves to yield a unique binary search tree? (Circle all that apply)

a) inOrder
b) preOrder
c) postOrder
d) levelOrder

[2 pts] A circular linked list is one in which the last node is connected to the first node. If we set a node curr equal to first node at the start of traversal, the condition to check if we have reached the end of the list then becomes

a) `curr == first`
b) `curr.next ==null`
c) `curr.next.next == first`
d) `curr.next == first`

11) What is a true statement about recursion and iteration?

a) There are some problems which can only be done recursively.
b) Recursion uses more memory than iteration
c) There are some problems which can only be done iteratively.
d) Iteration uses more memory than recursion

13) What does the following method do?

```
void foo(Node r) {
```

```
        if (r == null) return;
        Stack<Node> stk = new Stack<Node>();
        stk.push(r);
        while (!stk.isEmpty()) {
              Node t = stk.pop();
              print(t.key);
              if (t.right != null) stk.push(t.right);
              if (t.left != null) stk.push(t.left);
        }
}
```

a) Print each key in a singly-linked list
b) Print each key with an InOrder traversal of a tree
c) Print each key with a PreOrder traversal of a tree
d) Print each key in a doubly-linked list

Construct a binary tree from the following traversals.
InOrder: F, J, G, E, K, A, B
PreOrder: G, J, F, A, E, K, B

What are the postOrder and levelOrder traversals of *your* binary tree above?

postOrder	
levelOrder	

Please explain why the below implementation of the height method will always be O(n) regardless of the shape of the tree.

```
public int height(Tree t) {
      if (t == null)
            return 0;
      else
            return 1 + Math.max(height(t.left), height(t.right));
}
```

For project 6, you were tasked with comparing the speed of your own binary search tree to that of the TreeMap from Java. Please explain why inserting elements in order for your binary search tree was significantly slower than insertion for the TreeMap.