CMPT225, Spring 2021

Midterm Exam - Sample Solutions

Name_____

SFU ID: |__|__|__|__|__|__|__|__|__|__|

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. You should write your solutions directly in this word file, and submit it to Coursys. Submitting a pdf is also ok.
2. No late submissions, no exceptions
3. Write your name and SFU ID on the top of this page.
4. This is an open book exam.
   You may use textbooks, calculators, wiki, stack overflow, geeksforgeeks, etc.
   If you do, specify the references in your solutions.
5. Discussions with other students are not allowed.
   Posting questions online asking for solutions is not allowed.
6. The exam consists of four (4) problems. Each problem is worth 25 points.
7. Write your answers in the provided space.
8. Explain all your answers.
   Really, explain all your answers.

Good luck!

Problem 1 [25 points]

A. (3 points each) For each sentence decide whether it is True or False. **Write a brief explanation**.

1) Every recursive method can be written without making recursive calls.

ANSWER: True. we can simulate recursion by implementing the execution stack ourselves

2) Is the following definition legal of foo()? i.e., does it compile?

```
public int foo() { return null; }
```

ANSWER: False. int is a primitive type, and null refer only to objects not to primitive types

3) Is the following definition legal of foo()? i.e., does it compile?

```
public Integer foo() { return null; }
```

ANSWER: True. Integer is a class, and the return value is a null object.

4) Suppose we have two functions

```
public void foo(String s) { ...}
public void bar(String s) { ...}
```

Suppose the running time of foo $O(n^2)$ and the running time of bar is $O(n^3)$, where n is the length of the string. Is it true that foo() is faster than bar() on the input ”0101”.

ANSWER: False. Big-O notation refers only to running time for large n’s, but for an input of length 4 foo() could be slower than bar()

B. (6 points) Consider the following function. Use big-O notation to express the running time of the following function on an array of length n with start=0 and end=n-1. **Explain your answer.**

```
public int f1(int array[], int start, int end) {
        if (start == end) {
                array[start]++;
        }
        else {
                int quarter = (end - start)/4;
                f1(array, start, end - 2*quarter);
                f1(array, start + 2*quarter, end);
                f1(array, start + quarter, end - quarter);
        }
}
```

ANSWER: on arrays of length n we can write $T(n) = 3T(n/2) + O(1)$.
By Master Theorem we get that $T(n) = O(n^{\log_2(3)}) = O(n^{1.585})$

C. (7 points) Write a non-recursive method equivalent to f3(int n) below whose running time is $O(n^2)$. **Explain your answer, and explain the running time.**

```
public static int f2(int n) {
        if (n <= 2)
                return n;
        int sum = 0;
        for (int i = 1; i < n; i++)
                sum += f2(i) + f2(i-1);

        return sum;
}
```

ANSWER:

```
public static int f2(int n) {
        if (n<=2)
                return n;
        int f[] = new int[n+1];
        f[0] = 0; f[1] = 1; f[2] = 2;
        for (int j = 3; j < n+1; j++) {
                int sum = 0;
                for (int i = 1; i < n; i++)
                        sum += f[i] + f[i-1];
                f[j] = sum;
        }
        return f[n];
}
```

Problem 2 [25 points]

In this problem use the following definition of Binary Tree. You may assume the classes have the standard getters/setters.

```
public class BTNode<T> {
    private T data;
    private BTNode<T> leftChild;
    private BTNode<T> rightChild;
    private BTNode<T> parent;

}

public class BinaryTree<T> {
    private BTNode<T> root;

}
```

A. (6 points) Write a method for the class Binary Tree that returns the number of leaves in the tree. The running time should be O(size of the tree).
Explain your algorithm and running time.

```
public int numberOfLeaves() {

}
```

ANSWER: we use a helper method:

```
private int numberOfLeaves(BTNode<T> node) {
        if (node==null)
                return 0;
        return numberOfLeaves(node.getLeftChild()) +
                        numberOfLeaves(node.getRightChild()) + 1;
}
public int numberOfLeaves() {
        return numberOfLeaves(root);
}
```

The running time is clearly O(size of tree) since we touch every vertex exactly once

B. (7 points) Write a method that returns the preorder of the tree in an ArrayList. The running time should be O(size of the tree). **Explain your algorithm and running time**.

```
public ArrayList<T> preOrderArrayList() {
}
```

ANSWER: again, we use a helper method:

```
private ArrayList<T> preOrderArrayList(BTNode<T> node) {
        if (node==null)
                return new ArrayList<T>();
        else {
                ArrayList<T> ret = new ArrayList<T>();
                ret.add(node.getData());
                ret.addAll(preOrderArrayList(node.getLeftChild()));
                ret.addAll(preOrderArrayList(node.getRightChild()));
                return ret;
        }
}

public ArrayList<T> preOrderArrayList() {
        return preOrderArrayList(this.root);
}
```

The running time is clearly O(size of tree) since we touch every vertex exactly once

C. (12 points) Write a method for the class Binary Tree that gets another tree and checks if the trees have the same preOrder traversal. The running time should be O(n) in the worst case, where n is the size of the smaller tree. For example, if one tree has n vertices, and the other has n^2 vertices, the running time should be O(n). **Explain your algorithm and running time**.

```
public boolean samePreOrder(BinaryTree<T> tree) {


}
```

ANSWER: The idea is to run the iterative algorithm for preorder on both trees simultaneously. If at some point we get different values, return false. Otherwise, we return true.

```
public boolean samePreOrder(BinaryTree<T> tree) {
        if (tree == null)
                return false;

        BTNode<T> cur1, cur2;
        Stack<BTNode<T>> stack1 = new Stack<BTNode<T>>(); // used for this
        Stack<BTNode<T>> stack2 = new Stack<BTNode<T>>(); // used for other tree

        stack1.push(this.root);
        stack2.push(tree.root);
        while (!stack1.isEmpty() && !stack2.isEmpty()) {
                cur1 = stack1.pop();
                cur2 = stack2.pop();
                if (!cur1.getData().equals(cur2.getData()))
                        return false;

                if (cur1.getRightChild() != null)
                        stack1.push(cur1.getRightChild());
                if (cur1.getLeftChild() != null)
                        stack1.push(cur1.getLeftChild());

                if (cur2.getRightChild() != null)
                        stack1.push(cur2.getRightChild());
                if (cur2.getLeftChild() != null)
                        stack1.push(cur2.getLeftChild());
        }

        return  (stack1.isEmpty() && stack2.isEmpty())
}
```

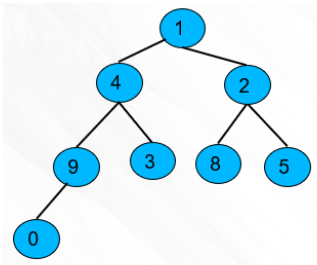
Problem 3 [25 points]

A. (7 points) Let A = [1, 4, 2, 9, 3, 8, 5, 0]. Apply the build-minHeap algorithm on A using the linear time algorithm we saw in class, and draw the tree representation of the heap. **Draw the intermediate steps.**

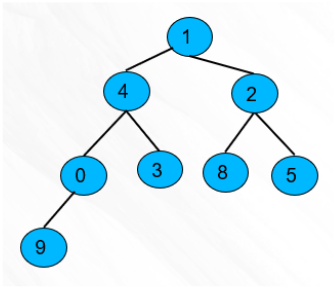
Recall the buildHeap algorithm:

```
Treat the array as a complete binary tree
For each vertex v starting from the bottom
    Apply heapify(v)
```

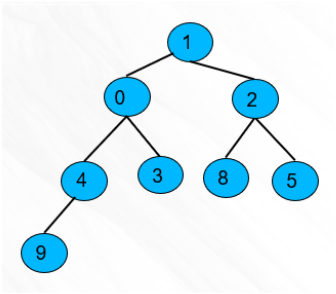
ANSWER:



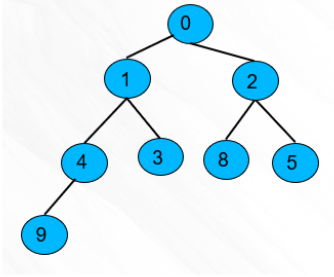
heapify(9) ==>



heapify(4) ==>

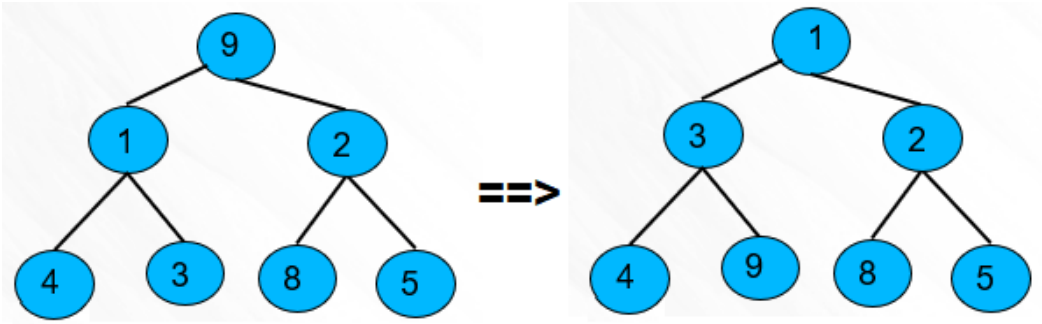


heapify(1) ==>



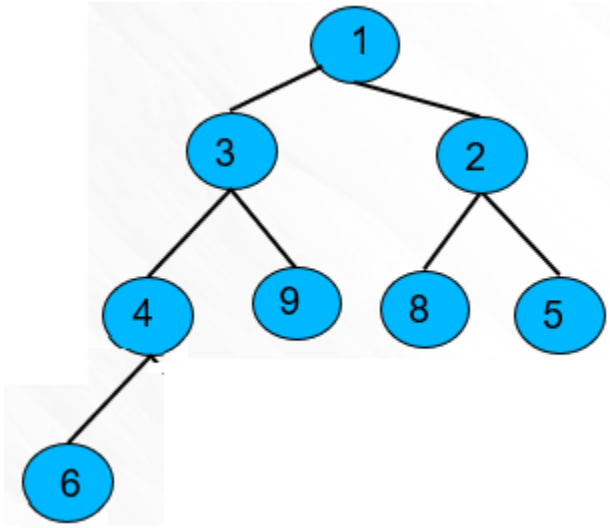
B. (4 points) Apply removeMin() on the heap obtained in part A, and draw the resulting heap.

ANSWER: 0 is removed from the tree, and will be returned. The we move 9 to the root and push it down to its position



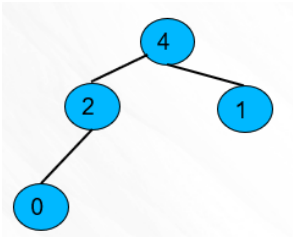
C. (4 points) Add 6 to the heap obtained in part B, and draw the resulting heap.

ANSWER: we first add 6 to as the left child of 4. 6 is greater than 4 and so we are done.



D. (4 points) If in the buildHeap() algorithm we iterate from top down, and heapify (push down) each node, will we get a heap in the end? If yes, prove it. If no provide a counter-example

ANSWER: No, consider the array [4,0,1]. This corresponds to the initial tree.



It is clear that using this algorithm 0 will not become the root.

E. (6 points) Write an algorithm that gets a Binary Search Tree of ints and creates a minHeap with the same values in O(n) time, where n is the size of the tree. The heap should be represented as an array.

ANSWER: simply create the inorder traversal of the BST.

Problem 4 [25 points]

A. (8 points) Write an algorithm that gets a root of a Binary Search Tree of ints and a number k, and returns a List with all numbers in the tree with value at most k. Write the algorithm as efficiently as possible. Ideally the running time should be O(length of the output + height of tree)
Explain your answer and explain the running time.

```
public ArrayList<Integer> valuesAtMostK(BTNode<Integer> bstRoot, int k) {
```

ANSWER: idea we create a list from the left subtree of the root, and then we go to the right subtree only if root <= k.

```
public ArrayList<Integer> valuesAtMostK(BTNode<Integer> bstRoot, int k) {
        if (bstRoot == null)
                return new ArrayList<Integer>();

        // find the values in the left cubtree
        ArrayList<Integer> ret =  valuesAtMostK(bstRoot.getLeftChild(), k);

        // search for values in the right subtree only if root is <= k
        if (bstRoot.getData() <= k) {
                ret.add(bstRoot.getData());
                ret.addAll(valuesAtMostK(bstRoot.getRightChild(), k));
        }
        return ret;
}
```

For the running time note that the function first goes to the minimal element in the BST, which takes O(height) time, but then it will only visit the nodes that are at most k. Therefore, the running time is O(height + output)

```
}
```

B. (8 points) Write an algorithm that gets two Stacks and checks if they are equal. In the end of the algorithm the stacks should be in their original state.

```
public <E> boolean equalStacks(Stack<E> s1, Stack<E> s2) {
```

ANSWER: we remove items from both stack simultaneously, check that they are equal.

```
public <E> boolean equalStacks(Stack<E> s1, Stack<E> s2) {
        if (s1 == null && s2 == null)
                return true;
        if (s1 == null && s2 != null)
                return false;
        if (s1 != null && s2 == null)
                return false;

        boolean areEqual = true;
        Stack<E> tmp1 = new Stack<E>();
        Stack<E> tmp2 = new Stack<E>();
        E cur1, cur2;
        while (!s1.isEmpty() && !s2.isEmpty() && areEqual) {
                cur1 = s1.pop();
                cur2 = s2.pop();
                if (!cur1.equals(cur2))
                        areEqual = false;
                tmp1.push(cur1);
                tmp2.push(cur2);
                // even if cur1 and cur2 are equal they might be not identical
                // hence we store both copies
        }

        if (!s1.isEmpty() || !s2.isEmpty())
                areEqual = false;

        // return back from tmps to stacks
        while (!tmp1.isEmpty())
                s1.push(tmp1.pop());

        while (!tmp2.isEmpty())
                s2.push(tmp2.pop());

        return areEqual;
}
```

C. (3 points each)

- Convert the following expression from infix to prefix: ((7 - 1) + (8 * (6 / 2)))

ANSWER: + - 7 1 * 8 / 6 2

- Convert the following expression from prefix to infix: / + 4 - 10 2 + 3 0

ANSWER: ((4+ (10-2)) / (3+0))

- Convert the following expression from prefix to postfix: / 14 - - 10 2 / 3 3

ANSWER: first convert to infix: (14 / ((10 - 2) - (3/3)))
Then to postfix: 14 10 2 - 3 3 / - /