CSCI-1200 Data Structures - Spring 2024
Test 3 – Thursday, April 4th 6pm-7:50pm

- This exam has 4 problems worth a total of 100 points.
- This packet contains 13 pages numbered 1-13. Please count the pages of your exam and raise your hand if you are missing a page.
- The packet contains 1 blank page, which is page 14 (not numbered). If you use the blank page to solve a problem, make a note in the original box and clearly label which problem you are solving on the blank page.
- **DO NOT REMOVE THE STAPLE OR SEPARATE THE PAGES OF YOUR EXAM. DOING SO WILL RESULT IN A -10 POINT PENALTY!**
- You may have pencils, eraser, pen, tissues, water, and your RPI ID card on your desk. Place everything else on the floor under your chair. Electronic equipment, including computers, cell phones, calculators, music players, smart watches, cameras, etc. is not permitted and must be turned "off" (not just vibrate).
- Raise your hand if you need to ask a proctor something that is NOT related to one of the questions on the test.
- Please state clearly any assumptions that you made in interpreting a question. Unless otherwise stated you may use any technique that we have discussed in lecture, lab, or on the homework.
- Please write neatly. If we can't read your solution, we can't give you full credit for your work.
- You do not need to write `#include` statements for STL libraries. Writing `std::` is optional. The keyword `auto` is not allowed.

Problem 1: Sets and Maps. [/14]

Part 1 Determine the output of this program. [/4]

```
#include <iostream>
#include <vector>
#include <set>

// std::set::insert() returns a pair, with its member pair::first set to an iterator
    pointing to either the newly inserted element or to the equivalent element already
    in the set. The pair::second element in the pair is set to true if a new element was
    inserted or false if an equivalent element already existed.
std::vector<int> myFunc(std::vector<int>& nums) {
    std::vector<int> ans;
    std::pair<std::set<int>::iterator, bool> tmp;
    std::set<int> set1;
    int size = nums.size();
    for(int i = 0; i < size; ++i) {
        tmp = set1.insert(nums[i]);
        if(tmp.second == false) {
            ans.push_back(*(tmp.first));
        }
    }
    return ans;
}

int main() {
    std::vector<int> nums = {4, 3, 2, 7, 8, 2, 3, 1};
    std::vector<int> result = myFunc(nums);

    // print the results
    std::cout << "result is: ";
    for (int num : result) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

This program will print one message to STDOUT, please complete this message:

```
        result is: ____________________
```

Solution:

```
    // this function finds all the duplicates in the vector.
    result is: 2 3
```

Part 2 Determine the output of this program. [/6]

```
#include <iostream>
#include <vector>
#include <map>

std::vector<int> myFunc(std::vector<int>& nums) {
    std::vector<int> ans;
    std::map<int,int> counters;
        for(int i=0;i<nums.size();i++){
            counters[nums[i]]++;
            if(counters[nums[i]]>1){
                ans.push_back(nums[i]);
```

```
            }
        }

        return ans;
}

int main() {
    std::vector<int> nums = {4, 3, 2, 7, 8, 2, 3, 1, 6, 10, 5, 5, 9};
    std::vector<int> result = myFunc(nums);

    // print the results
    std::cout << "result is: ";
    for (int num : result) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

This program will print one message to STDOUT, please complete this message:

```
        result is: ____________________
```

Solution:

```
    // this function finds all the duplicates in the vector.
    result is: 2 3 5
```

Part 3 Determine the output of this program. [/4]

```
#include <iostream>
#include <map>
#include <string>

char myFunc(const std::string& str) {
    std::map<char, int> frequencyMap;
    for (int i = 0; i < str.length(); ++i) {
        char ch = str[i];
        frequencyMap[ch]++;
    }

    char result = '\0';
    int maxFrequency = 0;
    for (std::map<char, int>::iterator it = frequencyMap.begin(); it !=
        frequencyMap.end(); ++it) {
        if (it->second > maxFrequency) {
            maxFrequency = it->second;
            result = it->first;
        }
    }
    return result;
}

int main() {
    std::string inputString = "Kevin Durant";
    char result = myFunc(inputString);
    std::cout << "result is: " << result << std::endl;
    return 0;
}
```

This program will print one message to STDOUT, please complete this message:

```
        result is: ____________________
```

Solution:

```
    // this function finds and prints the character from the input string that has the highest frequency.
    result is: n
```

Problem 2: Fill in the blanks [/20]

Part 1 The following function finds the height of a complete binary tree. Definition of a complete binary tree: every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. Definition of tree height: the height of the tree is defined as the length of the longest path from the root node to a leaf node. The height of an empty tree (a tree with no nodes) is 0. The height of a tree with only one node (the root node) is 1.

As of now, one line of this function is incomplete. Please complete this one single line. Do not write more than one line of code. **Do not call any external functions, such as std::max().** [/8]

```
// definition for a binary tree node.
class TreeNode {
public:
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

int height(TreeNode *root){
    if(root==NULL){
        return 0;
    }


    ------------------------------------------------------------------------------
}
```

Solution:

```
/* a complete binary tree's height is its left child's height plus 1. */
int height(TreeNode *root){
    if(root==NULL) return 0;

    return 1+height(root->left);
}
```

Part 2 Given a vector of integers arr, the following function returns true if the number of occurrences of each value in the vector is unique or false otherwise. As of now, two lines of this function are incomplete. Please complete these two lines. Do not write more than one line of code in each blank. [/12]

```
bool uniqueOccurrences(vector<int>& arr) {
        // map elements in vector arr to its number of occurrence
        std::map<int, int> map1;
        int size = arr.size();
        for(int i=0;i<size;i++){
            map1[arr[i]]++;
        }

        std::set<int> set1;
        std:map<int, int>::iterator itr = map1.begin();
```

```
        while(_____________________________________________){


            if(_____________________________________________){

                return false;
            }
            set1.insert(itr->second);
            itr++;
        }
        return true;
}
```

Example 1:

Input: arr = [1,2,2,1,1,3]
Output: true
Explanation: The value 1 has 3 occurrences, 2 has 2 and 3 has 1. No two values have the same number of occurrences.

Example 2:

Input: arr = [1,2]
Output: false
Explanation: The value 1 has 1 occurrence, 2 has 1 occurrence. These two values have the same number of occurrences.

Example 3:

Input: arr = [-3,0,1,-3,1,1,1,-3,10,0]
Output: true
Explanation: The value 1 has 4 occurrences, -3 has 3, 0 has 2, and 10 has 1. No two values have the same number of occurrences.

Solution:

```
bool uniqueOccurrences(vector<int>& arr) {
        // map elements in vector arr to its number of occurrence
        std::map<int, int> map1;
        int size = arr.size();
        // create the map
        for(int i=0;i<size;i++){
            map1[arr[i]]++;
        }

        std::set<int> set1;
        std:map<int, int>::iterator itr = map1.begin();

        while(itr!=map1.end()){

            if( set1.find(itr->second)!=set1.end() ){
                return false;
            }
            set1.insert(itr->second);
            itr++;
        }
        return true;
}
```

Problem 3: Big O Notation. [/20]

What's the runtime complexity of the following functions, in terms of n.

Function 1: Given the root of a binary tree, this function traverses the binary tree. Assume the binary tree has n nodes.

```
// definition for a binary tree node
class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

void function(TreeNode* root) {
    if(root == nullptr) return;
    std::cout << root->val << std::endl;
    function(root->left);
    function(root->right);
}
```

Function 2: This function searchs for a specific value in a balanced binary search tree. Assume the binary tree has n nodes, and it is an exactly balanced binary search tree.

```
// definition for a binary tree node
class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

bool search(TreeNode* root, int target) {
    if (root == nullptr) {
        return false;
    }
    if (root->val == target) {
        return true;
    }
    if (target < root->val) {
        return search(root->left, target);
    } else {
        return search(root->right, target);
    }
}
```

Function 3: This function searches all keys in a map. Assume the map has n elements.

```
void searchAllKeys(const std::map<int, int>& m) {
    for (std::map<int, int>::const_iterator it = m.begin(); it != m.end(); ++it) {
        int keyToFind = it->first;
        if (m.find(keyToFind) != m.end()) {
            std::cout << "Key " << keyToFind << " found!" << std::endl;
        } else {
            std::cout << "Key " << keyToFind << " not found!" << std::endl;
        }
    }
}
```

Function 4: This function prints all elements in a set. Assume the set has n elements.

```
void visitSet(const std::set<int>& s) {
```

```
    for (std::set<int>::const_iterator it = s.begin(); it != s.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;
}
```

Function 5: This function visits a board, whose width and height are both n. Assume this function will be called by another function like this: visitBoard(board, 0, 0);

```
void visitBoard(const std::vector<std::vector<char>>& board, int row, int col) {
    int n = board.size(); // Assuming square board

    // Base case: If we reach the end of the board
    if (row == n) {
        return;
    }

    // Process current cell
    char cell = board[row][col];
    std::cout << cell << " ";

    // Move to the next cell in the same row
    if (col + 1 < n) {
        visitBoard(board, row, col + 1);
    }
    // Move to the first cell of the next row
    else {
        std::cout << std::endl; // Move to next row
        visitBoard(board, row + 1, 0);
    }
}
```

// **write your answers here:**

Function 1: _____

Function 2: _____

Function 3: _____

Function 4: _____

Function 5: _____

Solution:

```
Function 1: O(n)
Function 2: O(log n)
Function 3: O(n(log n))
Function 4: O(n)
Function 5: O(n^2)
```

Problem 4: Trees. [/46]

Part 1 Determine the output of this program. [/5]

```
#include <iostream>

// definition for a binary tree node
class TreeNode {
```

```
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

void dfs(TreeNode* root, int max, int& count) {
    if (root == nullptr)
        return;

    if (root->val >= max) {
        max = root->val;
        count++;
    }

    dfs(root->left, max, count);
    dfs(root->right, max, count);
}

int myFunc(TreeNode* root) {
    if (root == nullptr)
        return 0;

    int max = root->val;
    int count = 0;
    dfs(root, max, count);
    return count;
}

int main() {
    // constructing a binary tree for testing
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(1);
    root->right = new TreeNode(4);
    root->left->right = new TreeNode(3);
    root->right->left = new TreeNode(1);
    root->right->right = new TreeNode(5);

    std::cout << "result is: " << myFunc(root) << std::endl;

    return 0;
}
```

This program will print one message to STDOUT, please complete this message:

```
        result is: ____________________
```

Solution:

// this program finds the number of good nodes in the binary tree. Given a binary tree root, a node X in the tree is named good if in the path from root to X there are no nodes with a value greater than the value of X.
result is: 4

Part 2 Determine the output of this program. [/5]

```
#include <iostream>
#include <algorithm> // for std::max
```

```
// definition for a binary tree node.
class TreeNode {
public:
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

int myFunc(TreeNode* root) {
    if(root == NULL){
        return 0;
    }
    // the function std::max(a,b) returns the largest of a and b. if both are equivalent, a is returned.
    return (1 + std::max(myFunc(root->left), myFunc(root->right)));
}

// utility function to create a new TreeNode
TreeNode* newNode(int data) {
    TreeNode* node = new TreeNode(data);
    return node;
}

int main() {
    TreeNode* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->left->left->left = newNode(8);
    root->left->left->right = newNode(9);
    root->left->right->left = newNode(10);
    root->left->right->right = newNode(11);
    root->right->left->left = newNode(12);
    root->right->left->right = newNode(13);
    root->right->right->left = newNode(14);
    root->right->right->right = newNode(15);
    root->left->left->left->left = newNode(16);
    root->left->left->left->right = newNode(17);
    root->left->left->right->left = newNode(18);
    root->left->left->right->right = newNode(19);
    root->left->right->left->left = newNode(20);

    std::cout << "result is: " << myFunc(root) << std::endl;

    return 0;
}
```

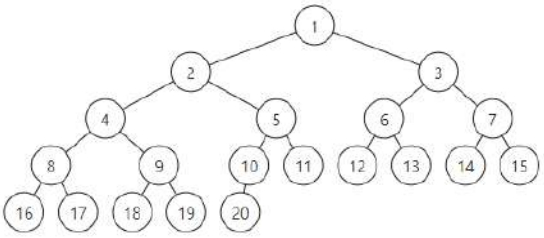
This program will print one message to STDOUT, please complete this message:

```
result is: ____________________
```

Solution:

// this program prints the maximum depth of a binary tree. A binary tree's maximum depth is the number of nodes

along the longest path from the root node down to the farthest leaf node.
result is: 5



Part 3 Determine the output of this program. [/10]

```
#include <iostream>
#include <vector>
#include <queue>

// definition for a binary tree node.
class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

std::vector<std::vector<int>> myFunc(TreeNode* root) {
    std::vector<std::vector<int>> ans;
    if (root == nullptr) return ans;

    std::queue<TreeNode*> q;
    q.push(root);

    bool flag = true;
    while (!q.empty()) {
        int sz = q.size();
        std::vector<int> level(sz);

        for (int i = 0; i < sz; ++i) {
            TreeNode* temp = q.front();
            q.pop();

            int index = (flag) ? i : (sz - 1 - i);
            level[index] = temp->val;

            if (temp->left){
                q.push(temp->left);
            }
            if (temp->right){
                q.push(temp->right);
            }
        }

        flag = !flag;
        ans.push_back(level);
    }

    return ans;
```

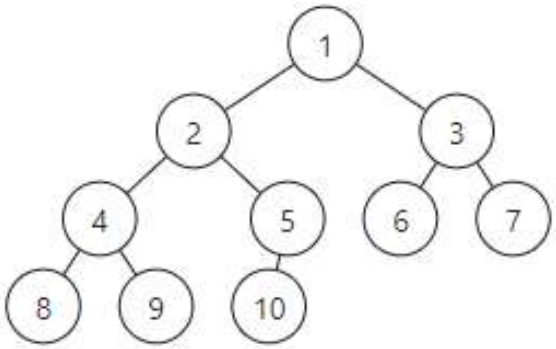
```
}

// utility function to create a new TreeNode
TreeNode* newNode(int data) {
    TreeNode* node = new TreeNode(data);
    return node;
}

int main() {
    // create a binary tree
    TreeNode* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->left->left->left = newNode(8);
    root->left->left->right = newNode(9);
    root->left->right->left = newNode(10);

    std::vector<std::vector<int>> result = myFunc(root);

    // display the result
    std::cout << "result is: " ;
    for (int i=0; i<result.size(); ++i) {
        for (int j=0; j<(result[i]).size(); ++j) {
            std::cout << result[i][j] << " ";
        }
    }
    std::cout << std::endl;

    return 0;
}
```

This program will print one message to STDOUT, please complete this message:

```
result is: ____________________________________________________________________________
```

Solution:

// this program prints the zigzag level order traversal of a binary tree.
result is: 1 3 2 4 5 6 7 10 9 8



Part 4 Determine the output of this program. [/5]

```
#include <iostream>

// definition for a binary tree node.
class TreeNode {
public:
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

int myFunc(TreeNode* root, int low, int high) {
    if (root == nullptr) {
        return 0;
    }

    // If the current node's value is less than low,
    // the entire left subtree can be ignored.
    if (root->val < low) {
        return myFunc(root->right, low, high);
    }
    // If the current node's value is greater than high,
    // the entire right subtree can be ignored.
    else if (root->val > high) {
        return myFunc(root->left, low, high);
    }
    // If the current node's value is within the range [low, high],
    // include the current node's value in the sum and recursively
    // search in both left and right subtrees.
    else {
        return root->val + myFunc(root->left, low, high) + myFunc(root->right, low, high);
    }
}

int main() {
    // create a binary search tree
    TreeNode* root = new TreeNode(10);
    root->left = new TreeNode(5);
    root->right = new TreeNode(15);
    root->left->left = new TreeNode(3);
    root->left->right = new TreeNode(7);
    root->right->right = new TreeNode(18);

    int low = 7;
    int high = 15;
    int result = myFunc(root, low, high);

    // print the result
    std::cout << "result is: " << result << std::endl;

    return 0;
}
```

This program will print one message to STDOUT, please complete this message:

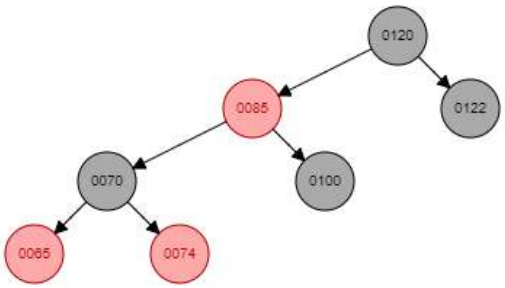
```
result is: ____________________
```

Solution:

// Given the root node of a binary search tree and two integers low and high, the above function returns the sum of values of all nodes with a value in the inclusive range [low, high].
result is 32. (7+10+15=32)

Part 4 Draw a red black tree for the following number sequence: 100, 120, 122, 85, 70, 74, 65. [/6]

Solution:



Part 5 Write a function called deleteTree(), which takes the root node pointer of a tree as its sole argument. This function deletes every node of the tree. [/15]

Your function will be tested using the following program. Your function must make sure this program does not have memory leaks. Keep in mind that this is NOT a binary tree.

```
#include <iostream>
#include <vector>

// definition for the tree node
class TreeNode {
public:
    int val;
    std::vector<TreeNode*> children;
    TreeNode(int x) : val(x) {}
};

// we will insert your function here, right before the definition of the main function.

int main() {
    // Example tree:
    //        1
    //      / | \
    //     2  3  4
    //    / \    /
    //   5   6  7
    TreeNode* root = new TreeNode(1);
    root->children.push_back(new TreeNode(2));
    root->children.push_back(new TreeNode(3));
    root->children.push_back(new TreeNode(4));
    root->children[0]->children.push_back(new TreeNode(5));
    root->children[0]->children.push_back(new TreeNode(6));
    root->children[2]->children.push_back(new TreeNode(7));

    // delete all nodes from the tree
    deleteTree(root);

    return 0;
}
```

Solution:

```
void deleteTree(TreeNode* root) {
    if (root == nullptr) {
        return;
    }

    // delete each child recursively
    for (TreeNode* child : root->children) {
        deleteTree(child);
    }

    // delete the current node
    delete root;
}
```