**CptS 223 – Advanced Data Structures in C++**
**Fall 2020**

# *Take-Home Exam 1*

Friday, October 16, 2020

**Your Name: _____Koji Natsuhara_____**

**ID Number: ___11666900_____**

**TA's Name: _____Nathan Wisner_____**

READ THE FOLLOWING INSTRUCTIONS:

This exam is take-home. You must work **individually** on this exam. You may use your book, notes, and online resources if necessary. Please either handwrite or type your answers into this document. Late exam solutions will **not** be accepted! **Please show all work!!!**

You must submit the exam through Git by Monday, October 19, midnight. On your local file system, and inside of your git repo for the class, create a new branch called Exam 1. Available to the branch create a .pdf called Exam1.pdf. Add, commit, and push Exam1.pdf using the technique discussed in class. Be sure to answer each question precisely. Do not provide superfluous details in your answers. *NOTE: you do not need to comment your code solutions.*

| Part | Points Possible | Points Earned |
|---|---|---|
| I | 30 | |
| II | 30 | |
| III | 40 | |
| BONUS | 5 | |
| **Total** | **100** | |

Your Name: _____ TA's Name: _____

***Part I: Conceptual Questions (30 pts) – Short answer, Fill-in-the-blank, and Multiple-choice***

1. **(3 pts)** In your own words, describe the purpose behind performing algorithm analysis.

The purpose behind performing algorithm analysis is to identify the space and time complexity of your algorithm. Using this form of analysis, you can estimate the rate at which the execution time grows relative to the size of the parameter you input. By doing this, you can effectively decide whether your algorithm is efficient or not.

2. **(3 pts)** What is the *Rule of the Big Five* in C++? Explain.

The Rule of the Big Five in C++ is that if you include any of these following functions, then you must include all of them:

Copy Constructor
Copy Assignment Operator
Move Constructor
Move Assignment Operator
Destructor

3. **(3 pts)** What are *Makefiles* and why do we use them for programs in this class? Explain.

Makefiles are used for programs to help compile many .cpp files in one line through the terminal. Because a program may require many .cpp files, it will become very tedious and time consuming to type out all the .cpp files to run the program, Makefiles simplify this process.

4. **(4 pts)** What do the following Linux commands do?

   a. **(1 pt)** man
   Displays the user manual for a command.

   b. **(1 pt)** mv
   Move files from one directory to another, or a directory from one place to another.

   c. **(1 pt)** mkdir
   Makes a new directory in a filesystem

   d. **(1 pt)** rm
   Removes a file or directory in the filesystem

Your Name: _____ TA's Name: _____

5. **(3 pts)** What is the general principle behind *software testing*? Explain.

The general principle behind software testing is to perform test cases for a function to make sure it works for the most extreme cases. Software testing is a form of debugging to check the actual result vs. the expected result. For example, if you were to software test an insert function for an BST, then you would you need to perform a test case if the BST was empty and if the BST has a node already inserted in.

6. **(3 pts)** Given the following function, what is the worst-case Big-O time complexity?

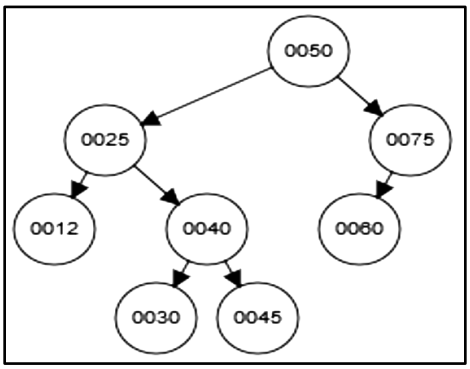
_____O( n )_____

```
// Prints all subarrays in arr[0..n-1]
void subArray(int arr[], int n)
{
    // Pick starting point
    for (int i=0; i <n; i++)
    {
        // Pick ending point
        for (int j=i; j<n; j++)
        {
            // Print subarray between current starting
            // and ending points
            for (int k=i; k<=j; k++)
            {
                cout << arr[k] << " ";
            }

            cout << endl;
        }
    }
```

Your Name: _____ TA's Name: _____

7. **(5 pts)** Fill-in-the-blank - Using the following BST, answer the questions below:




a. **(1 pt)** How many *children* does the node containing the number 0040 have? ___2_____

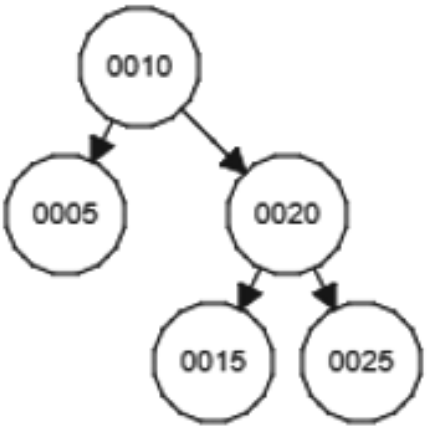
b. **(1 pt)** What is the *height* of the tree? ___3_____

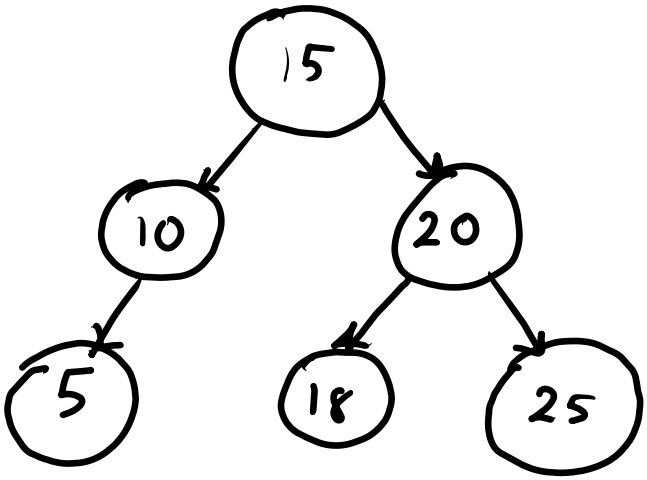
c. **(1 pt)** What is the *depth* of the node with value 0040? ___2_____

d. **(1 pt)** At what *level* is the root node? ___0_____

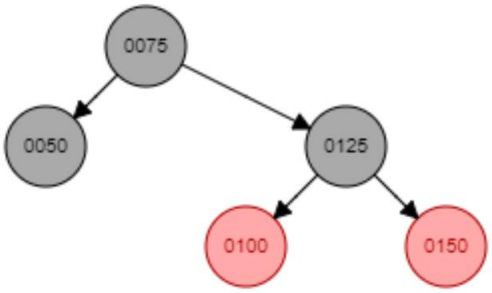
e. **(1 pt)** What is the *height* of the node with value 0040? ___2_____

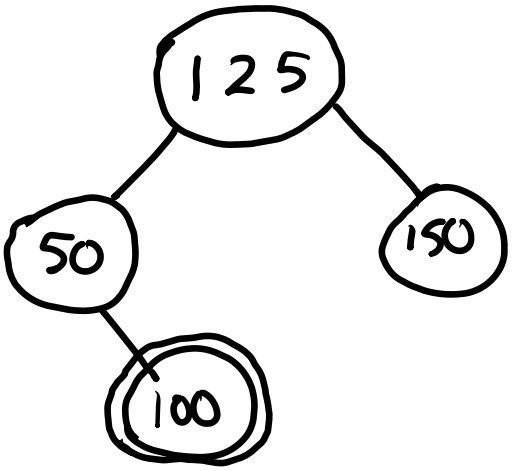
8. **(3 pts)** Insert the value 0018 into the following *AVL* tree; draw the resulting tree:



9.
**(3 pts)** Delete the value 0075 from the following *red-black* tree; draw the resulting tree:

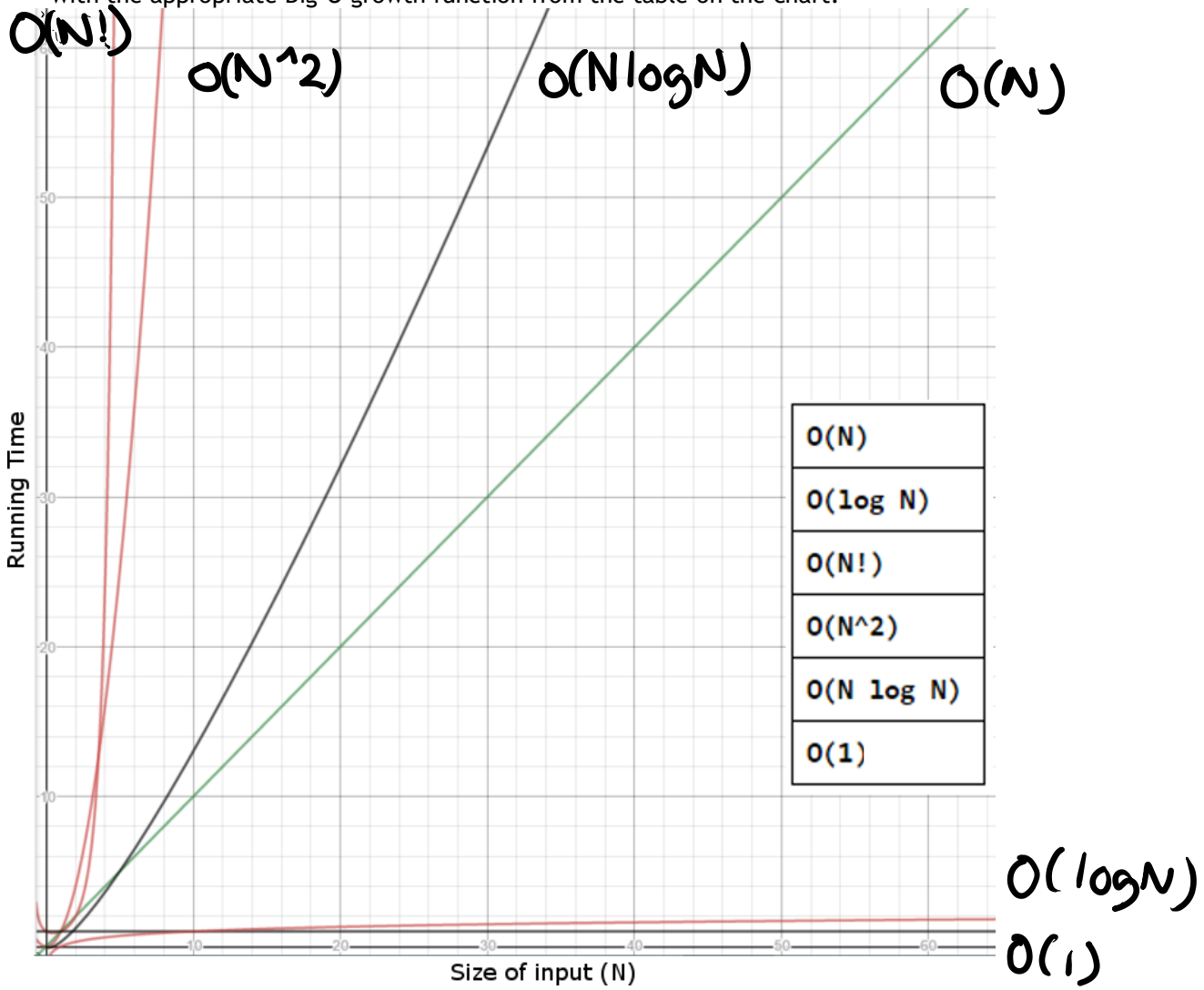


Your Name: _____ TA's Name: _____

***Part II: Algorithm Analysis and Big-O Notation (30 pts)***

10. **(4 pts)** The graph below shows different function growth complexities (Big-O orders). Label each line with the appropriate Big-O growth function from the table on the chart.



Your Name: _____ TA's Name: _____

11. **(6 pts)** Given the following function answer the questions below.

```
void someFunction( vector<int> & list ) {
   int clean_pass = false;
    for( int i = 1; !clean_pass && i < list.size(); i++ ) {
      clean_pass = true;
      for(int j = 0; j < (list.size() - i); j++)
            if( list[j] > list[j+1] ) {
                int temp = list[j];
                list[j] = list[j+1];
                list[j+1] = temp;
                clean_pass = false;
        }
    }
```

a. **(2 pts)** What does the function do?
This function takes in a vector<int> as a parameter and then sorts the vector from smallest to largest.

b. **(2 pts)** What is the worst-case Big-O *time* complexity for algorithm?
The worst-case Big-O time complexity for this algorithm is O(n^2).

c. **(2 pts)** What is the Big-O *space* complexity for algorithm?
The Big-O space complexity for this algorithm is O(n).

**11. (4 pts)** Given a *binary* tree, NOT *search* tree, what is the worst-case Big-O *time* complexity for finding the *max* value in the tree? List any assumptions you make.

The Big-O time complexity of finding the max value in a binary tree, NOT search tree would be O(n).

**12. (4 pts)** Given a red-black tree, what is the worst-case Big-O *time* complexity for finding the *min* value in the tree? List any assumptions you make

The worst-case Big-O time complexity for finding the min value in a red-black tree is O(logn).

14. **(4 pts)** Given an *AVL* tree, what is the worst-case Big-O *time* complexity for *inserting* an item in to the tree? List any assumptions you make.

The worst-case Big-O time complexity for inserting an item into an AVL tree would be O(logn).

15. **(4 pts)** Given a *BST*, not necessarily an AVL tree, what is the worst-case Big-O *time* complexity for determining the *height* of a node in the tree? List any assumptions you make.

Assuming that the BST is not self-balancing and that the height of the node isn't recorded within the node itself, the worst-case Big-O time complexity for determining the height of a node in the tree would be O(n).

16. **(4 pts)** We need to *eliminate* all *duplicates* from a string. What is the worst-case Big-O *time* complexity of your algorithm? List any assumptions you make. Also, justify your answer with a concise explanation.

Given a string to eliminate all duplicate characters within the string would be O(n^2). This is because this function would require a loop to pass through all the characters within the string to tally the number of each character within the string into an array. Afterwards, another loop is required to eliminate all the duplicates from the string based on the tally array from the previous loop. Because you are required to loop the entire string along with checking the array to identify if that character is a duplicate, the worst-case Big-O time complexity is O(n^2) where n is the length of the string.

***Part III: C++ Programming Questions (40 pts)***

17. **(15 pts)** You are given the following class definition for a BSTNode:

```
class BSTNode {
      private:
            int data, height;
            BSTNode * left, * right;
      public:
            BSTNode() : left(nullptr), right(nullptr) { }
};
```

Write a function `isBalancedBST()` that accepts a pointer to a `BSTNode` and determines if a BST is balanced. **After you write your function, state its *time* complexity using Big-O notation.** Make sure to include a function header and to determine an appropriate return type for your function. You may assume appropriate getters and setters are defined for the BSTNode class. List any other assumptions that you make.

This is assuming that all the nodes inserted into the BST have their correct height.

```
bool isBalancedBST(BSTNode *& t)
{
        if(t == nullptr)
        {
                return true; //The tree is balanced
        }
        If(t->left->height - t->right->height > 1 || t->right->height - t->left->height > 1)
        {
                return false; //The tree is not balanced
        }
        else
        {
                return true; //The tree is balanced
        }
}
```

The time complexity in Big-O notation for this function is O(1).

Your Name: _____ TA's Name: _____

18. **(25 pts) Parts a. and b. are related.**

a. **(10 pts)** You are given the following functions for an *AVL* tree, where `pTree` points to an `AVLNode` where an *imbalance* is found, and `heightDiff()` returns the difference between the heights of an `AVLNode`'s left and right children:

```
void AVLTree::leftRotate(AVLNode *& pTree);
void AVLTree::rightRotate(AVLNode *& pTree);
void AVLTree::leftRightRotate(AVLNode *& pTree);
void AVLTree::rightLeftRotate(AVLNode *& pTree);
int AVLNode::heightDiff();
```

First, write the function definition for the `rightRotate()` function given above. You are given the following definition for the `AVLNode` class:

```
class AVLNode {
private:
      int data, height;
      AVLNode * left, * right;
public:
      AVLNode() : left(nullptr), right(nullptr) { }
};
```

You may assume appropriate getters and setters are defined. List any other assumptions that you make.

```
rightRotate(AVLNode *& pTree)
{
      AVLNode *pTree2 = pTree->right;
      pTree->right = pTree2->left;
      pTree2->left = pTree;
      pTree->height = std::max(height(pTree2->left), height(pTree2->right)) + 1;
      pTree2->height = std::max(height(pTree2->right), pTree->height) + 1;
      pTree = pTree2;
}
```

b. **(15 pts)** Next, write a recursive function to insert a node into an AVL tree. Your insert function MUST ensure the tree is balanced after performing the insertion. You may assume the data values inserted into each tree are unique. Also, please use any functions described in part a. List any other assumptions that you make.Use the following function header:

This is assuming that a copy constructor and move constructor has been implemented within the AVLNode class.

Copy Constructor:

```
AVLNode(const int data,  AVLNode *left, AVLNode *right)
{
        This->data = data;
        This->height = 0;
        This->left = left;
        This->right = right;
}
```

Move Constructor:

```
AVLNode(const int data,  AVLNode *left, AVLNode *right)
{
        This->data = std::move(data);
        This->height = 0;
        This->left = left;
        This->right = right;
}
```

Your Name: _____ TA's Name: _____

`void AVLTree::insert(AVLNode *& pTree, int newData)`

```
void AVLTree::insert(AVLNode *& pTree, int newData)
{
	If(pTree == nullptr)
	{
		pTree = new AVLNode {newData, nullptr, nullptr};
	}
	Else if(newData <= pTree->data)
	{
		insert(pTree->left, newData);
	}
	Else if(newData > pTree->data)
	{
		insert(pTree->right, newData);
	}
	If(pTree->heightDiff() > 1)
	{
		If(pTree->left->left->height >= pTree->left->right)
		{
			leftRotate(pTree);
		}
		Else
		{
			rightleftRotate(pTree);
		}
	}
	Else
	{
		If(pTree->right->height - pTree->left->height > 1)
		{
			If(pTree->right->right->height >= pTree->right->left->height)
			{
				rotateRight(pTree);
			}
			Else
			{
				leftRightRotate(pTree);
			}
		}
	}
	pTree->height = std::max(pTree->left->height, pTree->right->height) + 1;
}
```

Your Name: _____ TA's Name: _____

**BONUS (5 pts):** Discuss tradeoffs between using the C++ STL *list* and *vector*.

Vectors in C++ are good for holding and element accessing. Vectors are considered synchronized objects and are efficient in random access. However, vectors are not as efficient in inserting or deleting data in the middle. Lists are more efficient than vectors in inserting and deleting data, but are not synchronized objects and cannot access elements randomly without iterating through the previous elements first.