

# CMPE 180A

## Data Structures and Algorithms in C++

Fall 2020

Instructor: Ron Mak

### Assignment #13

200 points

**Assigned:** Tuesday, November 17

**Due:** Tuesday, November 24 at 5:30 PM

**URL:** <http://codecheck.it/files/20042804593k6s1p5ai99unmsrad1kdmc96>

**Canvas:** Assignment 13: BST and AVL trees

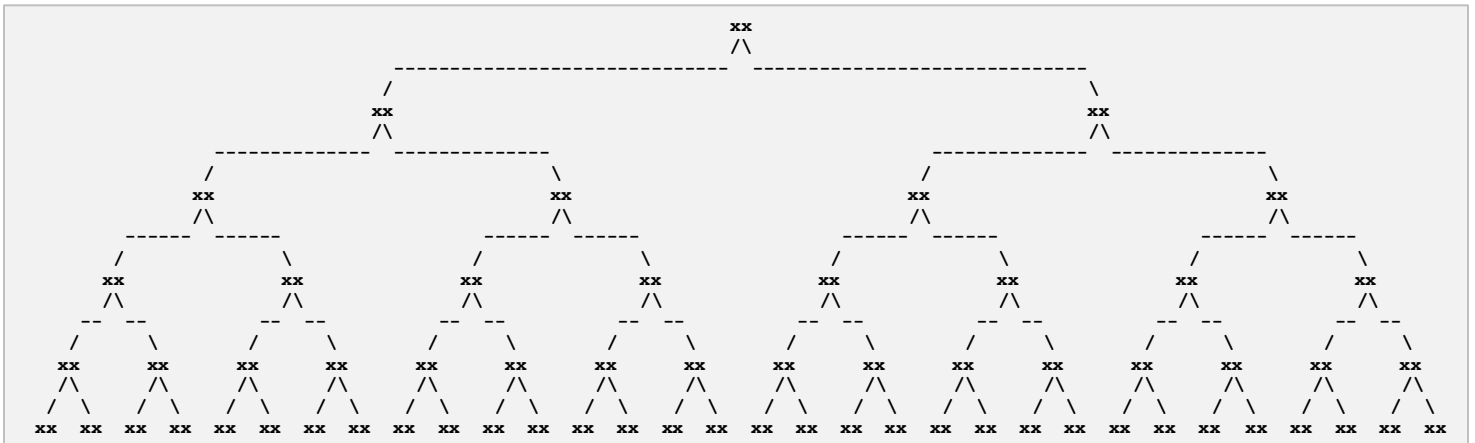
**Points:** 200

#### BST and AVL trees

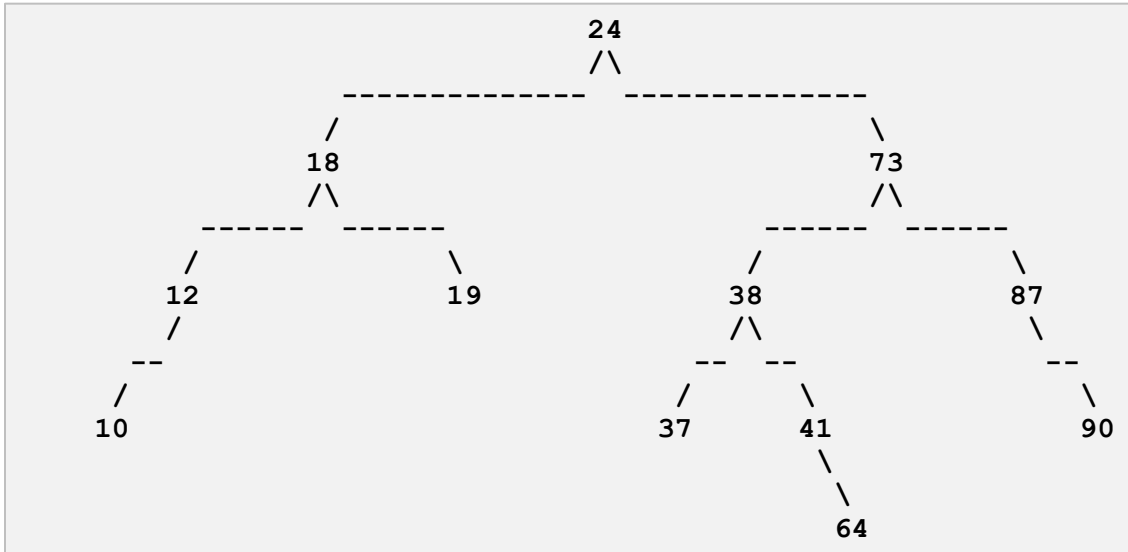
This assignment will give you practice with binary search trees (BST) and balanced Adelson-Velskii and Landis (AVL) trees.

#### A tree printer

You are provided a `TreePrinter` class with a `print()` function that will print any arbitrary binary tree. A template for how it prints a tree:



**TreePrinter** can print trees with height up to 5, i.e., up to 32 node values on the bottom row. An example of an actual printed tree:



### Part 1

The first part of the assignment makes sure that you can successfully insert nodes into and delete nodes from BST and AVL trees.

Insert node 62:

```
62
```

Insert node 71:

```
62
 \
  71
```

Insert node 29:

```
62
 /\
29 71
```

Insert node 88:

```
62
 /\
---  ---
/  \ /  \
29 71  \
      88
```

**First, create a BST node by node.** You will be provided the sequence of integer values to insert into the tree one at a time. Print the tree after each insertion. The tree will be unbalanced.

Then repeatedly delete the root of the tree. Print the tree after each deletion to verify that you did the deletion correctly. Stop when the tree becomes empty.

**Second, create an AVL tree, node by node,** by inserting the same sequence of integer values. You may need to rebalance the tree after an insertion. Print the tree after each insertion to verify that you are keeping it balanced.

Then, as you did with the BST, repeatedly delete the root of your AVL tree. You may need to rebalance the tree after a deletion. Print the tree after each deletion to verify that you are keeping it balanced.

Each time you do a rebalancing, print a message indicating which rotation operation and on which node. For example:

```
Insert node 10:
--- Single right rotation at node 21
```

A handy AVL tree balance checker:

```
int AvlTree::checkBalance(BinaryNode *ptr)
{
    if (ptr == nullptr) return -1;

    int leftHeight = checkBalance(ptr->left);
    int rightHeight = checkBalance(ptr->right);

    if ((abs(height(ptr->left) - height(ptr->right)) > 1)
        || (height(ptr->left) != leftHeight)
        || (height(ptr->right) != rightHeight))
    {
        return -2;        // unbalanced
    }

    return height(ptr); // balanced
}
```

### Expected output for Part 1

See <http://www.cs.sjsu.edu/~mak/CMPE180A/assignments/13/Part1-output.txt>

### Part 2

The second part of the assignment compares the performance of a BST vs. an AVL tree. Part 2 should be a separate program from Part 1. You will not be provided code for this part, but you should re-use code from Part 1 with any necessary modifications. Do this part outside of CodeCheck.

**First, generate  $n$  random integer values.**  $n$  is some large number, explained below. Insert the random integers one at a time into the BST and AVL trees. For each tree, collect the following statistics for all the insertion operations:

- **Probe counts.** A probe is whenever you visit a tree node via a pointer to the node and perform any one of these operations:
  - Check whether the node's left or right child link is null.
  - Use the left or right link to go to a child node.
  - Access the node's data value.If the pointer to the node is null, it is not a node visit and therefore not a probe.
- **Comparison counts.** A comparison is whenever you compare a node's data value to another value. A comparison is generally accompanied by a probe which accesses the node's data value.
- **Elapsed time** in milliseconds.

Do not print the tree after each insertion. Be sure to count probes and comparisons during AVL tree rotations.

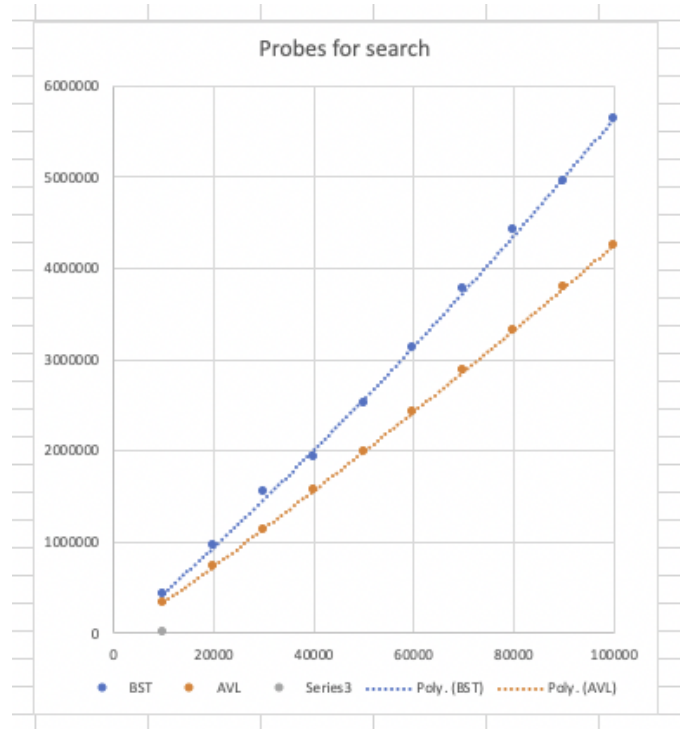
**Second, generate another  $n$  random integer values.** For each of the BST and AVL trees, count the total probes and comparisons and compute the total elapsed time to search for each value one at a time. It doesn't matter whether or not a search succeeds.

Choose values of  $n$  large enough to give you consistent timings that you can compare. Try values of  $n = 10,000$  to  $100,000$  in increments of  $10,000$ . Slow machines can use a different range of values for  $n$ , such as  $5,000$  to  $50,000$  in increments of  $5,000$ .

Print tables of these insertion and search statistics for the BST and AVL trees as comma-separated values (CSV). Use Excel or any other graphing tool to create the following graphs, each one containing two plots, one for BST and one for AVL:

- insertion probe counts
- search probe counts
- insertion compare counts
- search compare counts
- insertion elapsed time
- search elapsed time

**Generate Excel graphs.** First select the set of data sizes and the BST and AVL values (three columns) you want to graph, including column headers. Choose a scatter plot to insert. Then select one BST point on the graph, right-click to bring up the context menu, and add a polynomial trend line for the BST points. Add a trend line for the AVL points. Example graph:



### Example graphs for Part 2

Your graphs compare the performances of BST and AVL trees. See the examples in <http://www.cs.sjsu.edu/~mak/CMPE180A/assignments/13/Part2-graphs.pdf>. Your statistics may be different, but the trend lines should be similar, especially when showing whether BST or AVL performs better.

### Code

You can use any code from the lectures or from the textbook or from the Web. Be sure to give proper citations (names of books, URLs, etc.) if you use code that you didn't write yourself. Put the citations in your program comments. Do not copy from another student.

### What to submit

Submit into Canvas: Assignment #13:

- The signed zip file from CodeCheck for Part 1.
- A text copy of insertion and search statistics from Part 2.
- Copies (screen shots are OK) of the graphs from Part 2, or an Excel spreadsheet that contains the graphs.

## Rubrics

Criteria	Maximum points
<b>Part 1</b> <ul style="list-style-type: none"><li>• BST node insertions done correctly.</li><li>• BST node deletions done correctly.</li><li>• AVL tree rotations printed correctly.</li><li>• The AVL tree remains balanced after each node insertion.</li><li>• The AVL tree remains balanced after each node deletion.</li></ul>	<b>100</b> <ul style="list-style-type: none"><li>• 20</li><li>• 20</li><li>• 20</li><li>• 20</li><li>• 20</li></ul>
<b>Part 2</b> <ul style="list-style-type: none"><li>• BST insertion statistics</li><li>• AVL insertion statistics</li><li>• BST search statistics</li><li>• AVL search statistics</li><li>• Insertion probe counts graph</li><li>• Insertion compare counts graph</li><li>• Insertion elapsed time graph</li><li>• Search probe counts graph</li><li>• Search compare counts graph</li><li>• Search elapsed time graph</li></ul>	<b>100</b> <ul style="list-style-type: none"><li>• 10</li><li>• 10</li><li>• 10</li><li>• 10</li><li>• 10</li><li>• 10</li><li>• 10</li><li>• 10</li><li>• 10</li><li>• 10</li></ul>

## Academic integrity

You may study together and discuss the assignments, but what you turn in must be your individual work. Assignment submissions will be checked for plagiarism using Moss (<http://theory.stanford.edu/~aiken/moss/>). **Copying another student's program or sharing your program is a violation of academic integrity.** Moss is not fooled by renaming variables, reformatting source code, or re-ordering functions.

**Violators of academic integrity will suffer severe sanctions, including academic probation.** Students who are on academic probation are not eligible for work as instructional assistants in the university or for internships at local companies.