CSE 302 Homework Assignment 3

Due: Mar 30th, 2025, 11:59 PM

Instructions

Answer all questions carefully and completely. Ensure that your code for programming tasks compiles and functions as expected. **All submissions are to be made via Gradescope. The code part should be submitted through a separate Gradescope session.** Refer to the course syllabus and the Blackboard Assignments page for code submission details and help. **Also see the last page for submission guidelines.**

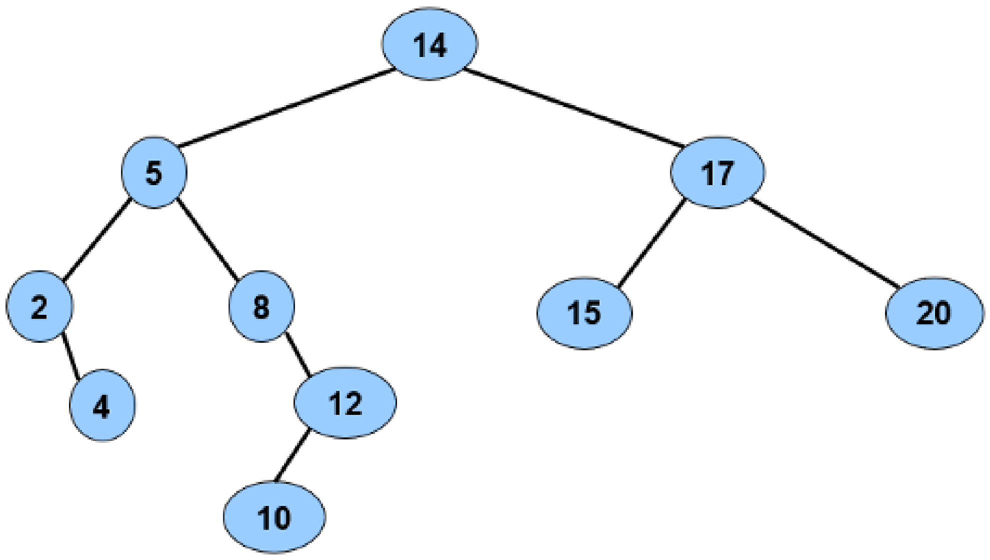
Problems

Q1 (20 pts) Answer the following questions about recursion in 2-6 sentences each.

(1) What are **base cases** and **general cases**? How are conditional statements relevant to them when we implement recursion?

(2) What is **multiple recursion**? Why is it usually impractical for real-world usage? What alternative solutions can be used to avoid the disadvantages of recursion?

Q2 (30 pts) Given the following binary search tree named *myBST*, answer the questions. (Assume it is the non-self-balancing implementation in our lecture.)



(1) Which nodes are leaf nodes? Explain in a sentence or two.

(2) What are the lists of node values when *myBST* is traversed using (i) **in-order traversal** and (ii) **post-order traversal**?

(3) Draw the new *myBST* after calling myBST.DeleteItem(14), and explain why it looks like this in a few sentences.

Programming

Q3 (50 pts) This problem requires you to add **two** new functions to the binary search tree (BST) implementation.

Instructions:

- **You must use the BST implementation (BST.cpp and BST.h) bundled with this homework as the base. Do not use a different version from an external source!**

Part I

Function Requirements: You need to add a new member function, `int BST::GetHeight()`, which returns the height of the BST. The height is defined as the maximum number of edges required to travel from the root node to any of its descendants. Specially, a single-node tree has a height of zero and an empty tree has a height of -1.

Example: For the *myBST* in Q2, calling myBST.GetHeight will return 4. It is the edges between 14 and 10, which are the ones between 14 & 5, 5 & 8, 8 & 12, and 12 & 10. Among all the descendants, you need to travel the most edges from root node 14 to node 10, which is also the deepest in the same way.

Part II

Function Requirements: You need to add another new member function, `void BST::SplitBalance()`, which will rebalance the BST using the **split-balancing** algorithm covered in Session 24.

Recommended Approach:

1. Store the BST elements in sorted order using a traversal queue and an array.
2. Rebuild the tree from scratch using the sorted array and the split-balancing algorithm.

IMPORTANT: You should assume 10,000 elements will be inserted into the BST. This means you can use a fixed-length array. Otherwise, if you use a dynamic array, make sure it works as expected for 10,000 elements. You should not assume the elements' values or initial arrangments in the BST.

Additional Notes and Reminders:

- Match your function names, return types, and parameter types exactly as specified.
- **Always initialize all variables explicitly.** For example, if you want to set `myInt` to 0, do

  ```
  int myInt = 0;
  ```

  instead of just:

  ```
  int myInt;
  ```

Submission

Submit your solutions via Gradescope. Be sure to include:

- Follow this template.
- All written responses in a single document (.txt, .doc, or .pdf, preferably PDF) named "LN_FN_3" where LN is your last name, and FN is your first name.
- The code for Q3 should be submitted as two files, BST.cpp and BST.h. You will be allowed up to 10 submissions. Again, make sure that the code can compile successfully using the latest stable distribution of GCC and GNU Make.