COMP 210 – Data Structures and Analysis (Sec 1&2)

Assignment #4 – Binary Search Trees

Issue Date: October 12th, 2023

Due Date: October 23rd, 2023, 11:55PM

Marks: 5

In this Assignment, you will be required to write Java methods to fulfill several tasks related to Binary Search Trees. Before continuing, make sure you're caught up on the lectures on the topic. You will use what we have learned about Binary Search Trees in this assignment, and how to put them into practice using Java.

Don't stress if something doesn't work — come to the office hours and/or post on Piazza! If you feel entirely stuck and unable to resolve your issue, email Qiwei Zhao: qiwei@cs.unc.edu and the LAs team will work together to help you!

Part A. Complete the Prerequisites

Complete the following requirements as done in previous assignments.

1. Prepare the Java Develop Environment

This step should already have been completed in the previous assignments. You should have already installed the required software such as Java, and IntelliJ.

2. Set up your java project environment.

To set up the starter code for Assignment #4, follow these steps:

1. Open IntelliJ IDEA.
2. Navigate to your previously created **COMP210** Project. Within this project, right-click on the src directory and choose "New Package". Name this new package "**assn04**".
3. Locate the assn04_java.zip file within your homework package and unzip it. This will reveal the starter code Java files.
4. Copy all four Java files from the unzipped folder and paste them into the "assn04" package you created in the COMP210 Project.
5. Ensure that the JDK is set up as it was in previous assignments.

With these steps completed, you are all set to proceed with this assignment. For the subsequent portions of the exercise, you have the freedom to design your program as you see fit, provided it adheres to the criteria detailed in part B. This means your program should process inputs as described in part B and generate outputs as required. Feel free to incorporate any helper methods or classes you deem beneficial. There are likely multiple ways to successfully complete this assignment.

Part B. Write a program to process the following linked list problems.

1. Overview of the starter code and list of tasks to be performed.

In this assignment your task is to complete a series of methods on binary search trees. You are being provided with the following 4 java class files:

• Main.java	This is the main user program. You can test your code using this.
• BST.java	This is the interface of for the BST classes, and specified the methods that you need to implement.
• EmptyBST.java	This is an implementation for an *empty BST*.
• NonEmptyBST.java	This is where you need to implement your solutions.

Following are the details of your tasks to be performed along with some suggestions that you may find useful:

1, Locating Methods for Editing:

- In **NonEmptyBST.java**, methods that require your attention are marked with a "TODO" comment.
- Edit only the methods located at the top of the file. Methods already implemented and situated at the bottom of the file should remain unchanged.

2, Package Contents Review:

- While other Java files are present in the package, you are advised against modifying them.
- The package includes two implementations of a Binary Search Tree (BST). Familiarize yourself with the contents, as understanding them is important for completing the assignment.
- Reviewing associated lectures and previously covered code will improve your understanding and aid in task completion.

3, Testing Your Code:

Main.java offers an example of how to test your solution. Unlike in Assignment 3, we haven't provided a **toString** method. We believe using the debug function is an effective approach to inspect the BST structure for this assignment. Nevertheless, if you feel a string representation would be beneficial, consider crafting a helper **toString** method to display the BST and relevant data. Tailor your approach based on your needs.

This assignment is more complex than previous assignments. Allocate ample time, and ensure that you start **early**. While working on various methods, recursion might be a preferable approach. However, bear in mind:

- The left and right subtrees of a node are, in themselves, binary search trees.
- If e.g. a **left** subtree isn't empty, all nodes within it have key values **less** than the root node's key value.

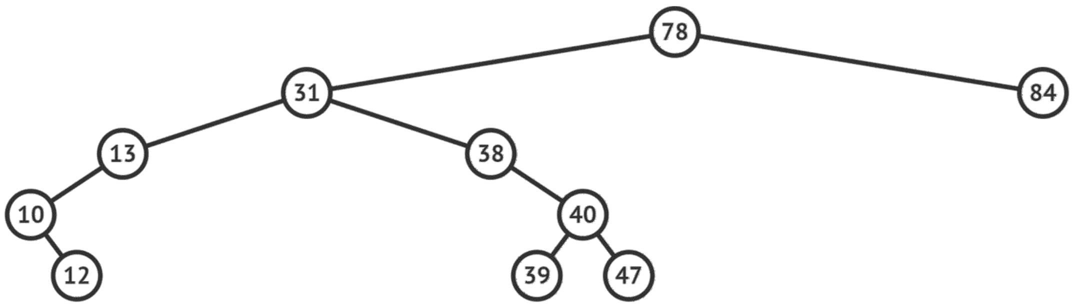
- When dealing with subtrees, numerous boundary conditions may arise. It's beneficial to enumerate all potential scenarios to prevent bugs.

Good luck, and plan your time wisely to ensure a thorough understanding and completion of the assignment on time.

Task 1: Insert

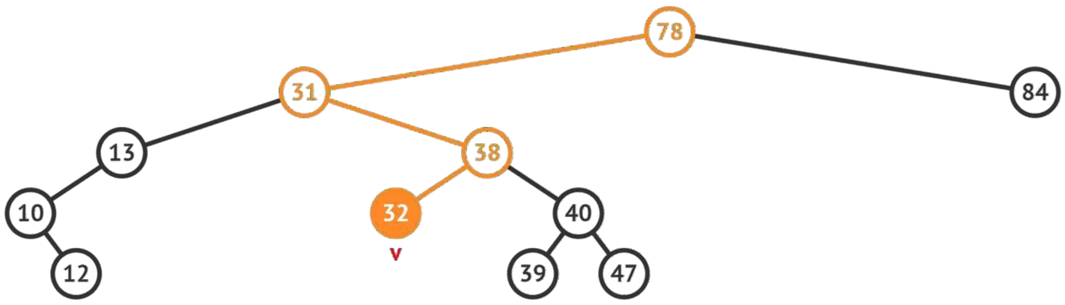
In this task, you must complete the **BST<T> insert (T element)** method, which should insert an element into the tree in the appropriate position and return the mutated tree after insertion or a new tree with the inserted element. (In case of equality, you can insert in the right subtree).

Example:



Insert (32)

Return:

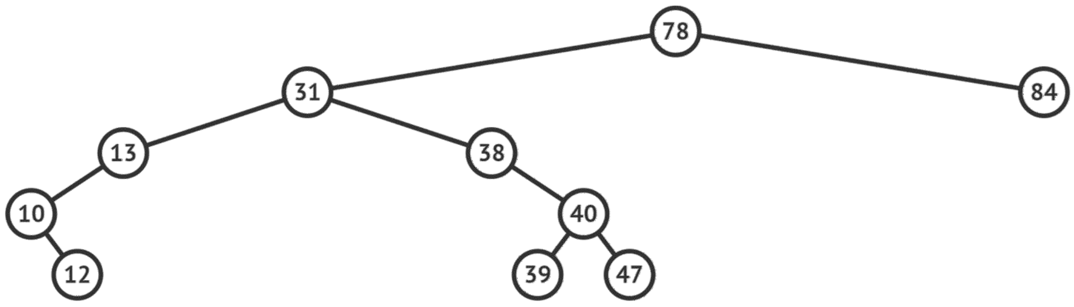


Task 2: Remove

In this task, you are required to complete the **BST<T> remove (T element)** method, and the **T findMin()** method. The **remove** method should remove the given element from the tree if it is present, and either return the possibly mutated tree after removal or an empty tree. The **findMin** method should return the minimum value in the tree.

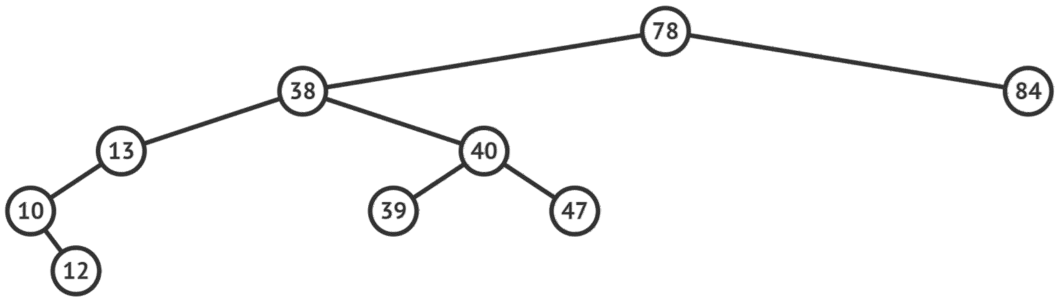
Note, *for selecting a successor node, you should choose the smallest from the right subtree*.

Example:



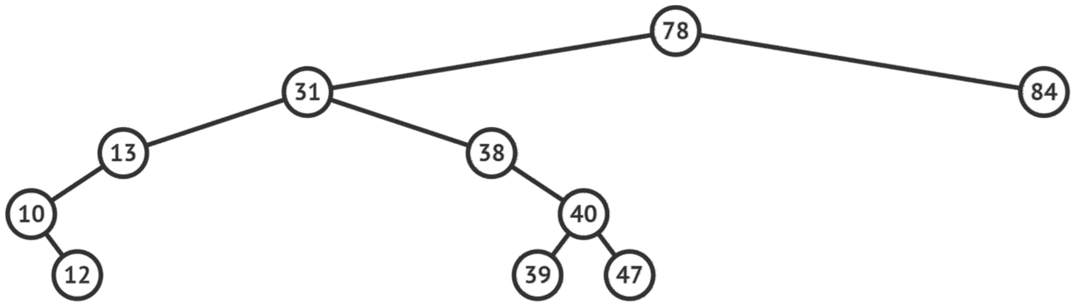
Remove (31)

Return:



Task 3:

In this task, you will be asked to complete the **printPreOrderTraversal** method, which should print the tree in using (**depth-first)** pre-order traversal. **Your method should print the elements all in one line with a space *after* each element.** Example:

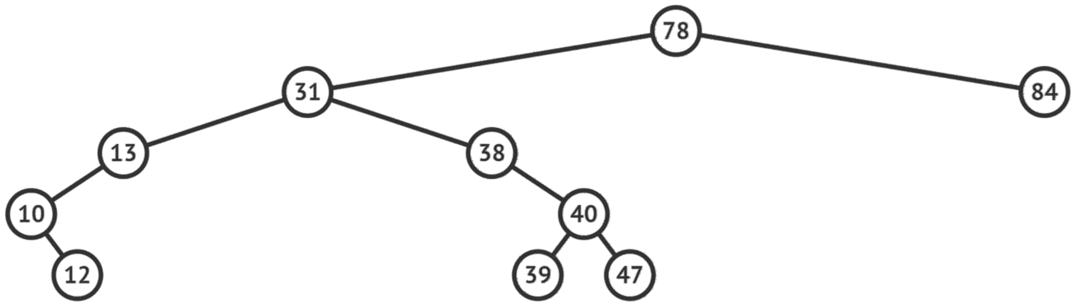


Return: 78 31 13 10 12 38 40 39 47 84

Task 4:

In this task, you will be asked to complete the **printPostOrderTraversal** method, which should print the tree using *post-order traversal*. **Your method should print the elements all in one line with a space after each element.**

Example:



Return: 12 10 13 39 47 40 38 31 84 78

2. Submit to Gradescope for Grading

It's time to submit your work on Gradescope for evaluation!

Before the deadline, you can submit your assignment multiple times without any penalties. We encourage this approach, especially since the assignments are autograded, offering nearly instant feedback. We aim for you to leverage these resubmissions to achieve full autograding credit.

Creating a Zip Archive in IntelliJ:
Let’s return to IntelliJ to prepare your zip archive for submission:

Mac Users

Along the bottom of your window, you should see an option to open a terminal integrated into IntelliJ. Type the following command (all on a single line, including the ‘.’):

zip -r assn04_submission.zip assn04 .
(You need to zip the whole file, including src file)

In the file explorer pane, look for the zip file named “assn04_submission.zip”. If you right click on this file “Open in -> Finder” on Mac, the zip file’s location on your computer will open. Upload this file to Gradescope to submit your work for this assignment.

Windows Users

Please navigate to your course workspace in the File Explorer window. Then right click on the src folder in your exercises directory and compress the directory into a zip folder. You can name it “assn04_submission.zip” Please pay attention, the level of the folder needs to be the same as the description. Compression software on Windows may introduce additional folder structure, and the file naming must be the same.

Before uploading the zip folder to Gradescope, please delete any files that showed up in the src/ folder that were not actually part of assn04. Specifically, in this assignment, the files that need to be kept are the BST.java, Main.java, EmptyBST.java, and NonEmptyBST.java. Then:

1. Log in to Gradescope.
2. Choose the assignment titled “**Assignment 4 – Binary Search Trees**”.
3. You'll find an option to upload a zip file.

Autograding will require a short while to process. For this exercise, there isn't a "human-graded" component. Consequently, you should aim to achieve the full score of 100 points. If you encounter any reported issues, you should try to resolve these and then continue to resubmit until all issues are cleared.