# Homework 4

EXPERIMENTATION WITH BSTS

VERSION 1.0

The objective of this assignment is to reinforce concepts we learned about BSTs. The programming assignment should be implemented in Java. You are required to work alone on this assignment. This is an extra credit assignment worth 5 points.

This assignment may be modified to clarify any questions (and the version number incremented), but the crux of the assignment and the distribution of points will not change. If there are any changes to the assignment, all changes will be documented in the "Change History" section of this assignment.

## 1 Description of Task

As part of this assignment, you will be responsible for designing programs that implement a Binary Search Tree data structure. The programming assignment should be developed in Java version 8, and your classes must reside in the package "cs250.hw4".

You will be designing two classes that implement the functionality of storing data in memory using a BST data structure.

We will compile and run your program using the commands below, replacing "<`filename` >" with the appropriate filename as described in the tasks. See the "Example Outputs" section for more examples on how your program will be run. Your code must compile and run on the Linux machines in the CSB 120 computer lab, using the versions of Java provided by those machines (either version 8 or version 11 is acceptable). It is highly recommended to develop your code using VS Code, either directly on the machines or through the Remote SSH extension in VS Code. Other IDE's or text editors may not be supported by the instructors.

### 1.1 Testing your code

We will provide an input file containing many random integers separated by new lines. You should read in this file inserting each line as an entry into the tree.

The Driver code defined below will handle all the execution of reading from the file and inserting/removing from the tree. If your output matches ours (not counting the timestamps since those will be different) then your tree is most likely implemented correctly.

### 1.2 Task 1: Build a Binary Search Tree (5 pts)

You should create a class called: **BinaryTree** that implements the **TreeStructure** interface that is provided below. Each of the methods part of the **TreeStructure** interface should be implemented using the same logic discussed in the lecture slides.

**Command line compilation**: `javac cs250/hw4/BinaryTree.java`

**Command line execution format**: `java cs250.hw4.BinaryTree <filename>`
**Command line execution example**: `java cs250.hw4.BinaryTree data.txt`

**Points Breakdown**

**1 point** for correctly implementing the "**insert**" operation.
**1 point** for correctly implementing the "**remove**" operation.
**1 point** for correctly implementing the "**get**" operation.
**1 point** for correctly implementing the "**findMaxDepth**" operation.
**1 point** for correctly implementing the "**findMinDepth**" operation.

## 1.4 TreeStructure interface explanation

Each of your classes should implement the **TreeStructure** interface we have provided below.

Recall in Java you implement an interface by specifying so in the class declaration.

Example:

```
public class ExampleTree implements TreeStructure {}
```

There are 5 methods in the **TreeStructure** interface which will interact with your underlying data structure.

1. **Insert(num)**: When called this method should insert a node into the tree with the value **num** into the tree along with the current **time** from **System.nanoTime()**
2. **Remove(num)**: When called this method should find and remove the node in the tree with the value **num**. If **num** did not exist in the tree this method should return **False** and if **num** did exist in the tree this method should return **True**.
3. **Get(num)**: When called this method should find the node with the value **num** then return the **time** that the number was inserted at.
4. **FindMaxDepth()**: When called this method should return the maximum depth of the current tree.
5. **FindMinDepth()**: When called this method should return the minimum depth of the current tree.

TreeStructure Code:

```
package cs250.hw4;

public interface TreeStructure {

    public void insert(Integer num);

    public Boolean remove(Integer num);

    public Long get(Integer num);

    public Integer findMaxDepth();

    public Integer findMinDepth();

}
```

### 1.5 Example output

Use the Driver Code provided to test your code without having to manually create and test the tree

Note: Outputs should match mainly on min and max depth, the insertion times will not be exact and will change depending on computer and implementation of methods.

**Binary Tree Driver Code**

```
public static void main(String[] args) throws FileNotFoundException, IOException {
File file = new File(args[0]);
        FileReader fReader = new FileReader(file);
        BufferedReader bufferedReader = new BufferedReader(fReader);
        TreeStructure tree = new BinaryTree();
        Random rng = new Random(42);
        ArrayList<Integer> nodesToRemove = new ArrayList<>();
        ArrayList<Integer> nodesToGet = new ArrayList<>();
        String line = bufferedReader.readLine();
while (line != null) {
            Integer lineInt = Integer.parseInt(line);
tree.insert(lineInt);
            Integer rand = rng.nextInt(10);
            if (rand < 5) nodesToRemove.add(lineInt);
else if (rand >= 5) nodesToGet.add(lineInt);
line = bufferedReader.readLine();
        }
        bufferedReader.close();
for (int i = 0; i < 10; i++) {
            System.out.println(nodesToGet.get(i) + " inserted at " + tree.get(nodesToGet.get(i)));
        }
        System.out.println("Max depth: " + tree.findMaxDepth());
System.out.println("Min depth: " + tree.findMinDepth());
for (Integer num : nodesToRemove) {
tree.remove(num);
        }
        for (int i = 0; i < 10; i++) {
            System.out.println(nodesToGet.get(i) + " inserted at " + tree.get(nodesToGet.get(i)));
        }
        System.out.println("Max depth: " + tree.findMaxDepth());
        System.out.println("Min depth: " + tree.findMinDepth());
    }
```

**Binary Tree Output**

```
-1643699032 inserted at 13017969196679
1614093892 inserted at 13017969230068
-1000272748 inserted at 13017969239743
-1907667521 inserted at 13017969249045
1912735862 inserted at 13017969258249
-1142304848 inserted at 13017969303508
1564342588 inserted at 13017969321896
624763728 inserted at 13017969341361
-200006731 inserted at 13017969372713
-284011743 inserted at 13017969401098
```

```
Max depth: 33
Min depth: 5
```

```
-1643699032 inserted at 13017969196679
1614093892 inserted at 13017969230068
-1000272748 inserted at 13017969239743
-1907667521 inserted at 13017969249045
1912735862 inserted at 13017969258249
-1142304848 inserted at 13017969303508
1564342588 inserted at 13017969321896
624763728 inserted at 13017969341361
-200006731 inserted at 13017969372713
-284011743 inserted at 13017969401098
Max depth: 28
Min depth: 6
```

## 2 What to Submit

Use the CS250 *Canvas* to submit a single .tar or .zip file that contains your Java source code, plus a README.txt file. When the archive is opened, the directory structure should be as follows:

- cs250 ○ hw4
    - BinaryTree.java
    - Any other .java files needed for the program
    - README.txt

The README.txt file must contain a description of each file submitted and any other information you feel that the TAs will need to grade your program.

Other files (such as the ".class" files from compiling your code, or your "hw1" folder from the previous assignment) are allowed to be present and will be ignored.

**Filename Convention:** The archive file should be named as <FirstName>-<LastName>-HW4.tar. E.g., if you are Cameron Doe then the tar file should be named Cameron-Doe-HW4.tar. Note: Canvas sometimes adds an extra "-1", "-2", etc. to the file name, which is OK.

**Tar File Command Format:** `tar -cf <FirstName>-<LastName>-HW4.tar cs250`
**Tar File Command Example:** `tar -cf Cameron-Doe-HW4.tar cs250`

**Zip File Command Format:** `zip -r <FirstName>-<LastName>-HW4.zip cs250`
**Zip File Command Example:** `zip -r Cameron-Doe-HW4.zip cs250`

Note: we highly recommend using one of the commands above for creating your archive for submission. Creating an archive through your operating system's GUI (or another GUI program) may not zip the file correctly, potentially resulting in a deduction on the assignment.

## 3 Grading

The assignments must compile and function correctly on machines in the CSB-120 Lab. Assignments that work on your laptop on your particular flavor of Linux, but not on the Lab machines are considered unacceptable.

This assignment will contribute a maximum of 5 extra credit points towards your final grade. The grading will also be done on a 5 point scale. The points breakdown is as follows:

**Points Breakdown**

**5 points** for correctly performing Task 1.

**Deductions:**

**2 points each** for not naming your classes what we specified.
**2 points** for any error requiring manual intervention including:

- Incorrect package name
- Incorrect class name
- Not implementing the interface provided - Etc.

You are required to **work alone** on this assignment.

## 4 Late Policy

Please check the class policy on submitting late assignments. You are allowed to submit assignments up to 7 days late with a 10% deduction for every day late.

## 5 Version Change History

This section will reflect the change history for the assignment (if needed) after the first public release of the assignment. It will list the version number, the date it was released, and the changes that were made to the preceding version. Any changes to the first public release are made to clarify the assignment; the spirit or the crux of the assignment will not change.

| **Version** | **Date** | **Comments** |
|---|---|---|
| | | |