Important

There are general homework guidelines you must always follow. If you fail to follow any of the following guidelines, you risk receiving a **0** for the entire assignment.

1. All submitted code must compile under **JDK 11**. This includes unused code, so don't submit extra files that don't compile. Any compile errors will result in a 0.
2. Do not include any package declarations in your classes.
3. Do not change any existing class headers, constructors, instance/global variables, or method signatures. For example, do not add `throws` to the method headers since they are not necessary.
4. Do not add additional public methods.
5. Do not use anything that would trivialize the assignment. (e.g. Don't import/use `java.util.ArrayList` for an `ArrayList` assignment. Ask if you are unsure.)
6. Always be very conscious of efficiency. Even if your method is to be $O(n)$, traversing the structure multiple times is considered inefficient unless that is absolutely required (and that case is extremely rare).
7. You are expected to implement all methods on the homework. Each unimplemented method will result in a deduction.
8. You must submit your source code, the `.java` files, not the compiled `.class` files.
9. Only the last submission will be graded. Make sure your last submission has **all** required files. Resubmitting will void all previous submissions.
10. After you submit your files, redownload them and run them to make sure they are what you intended to submit. You are responsible if you submit the wrong files.

Collaboration Policy

Every student is expected to read, understand and abide by the Georgia Tech Academic Honor Code.

When working on homework assignments, you **may not** directly copy code from any source (other than your own past submissions). You are welcome to collaborate with peers and consult external resources, but you **must** personally write all of the code you submit. **You must list, at the top of each file in your submission, every student with whom you collaborated and every resource you consulted while completing the assignment**.

You may not directly share any files containing assignment code with other students or post your code publicly online. If you wish to store your code online in a personal private repository, you can use Github Enterprise to do this for free.

The only code you may share is JUnit test code on a pinned post on the official course Piazza. Use JUnits from other students at your own risk; **we do not endorse them**. See each assignment's PDF for more details. If you share JUnits, they **must** be shared on the site specified in the Piazza post, and not anywhere else (including a personal GitHub account).

Violators of the collaboration policy for this course will be turned into the Office of Student Integrity.

Style and Formatting

It is important that your code is not only functional, but written clearly and with good programming style. Your code will be checked against a style checker. The style checker is provided to you, and is located on Canvas. It can be found under Files, along with instructions on how to use it. A point is deducted for every style error that occurs. If there is a discrepancy between what you wrote in accordance with good style and the style checker, then address your concerns with the Head TA.

Javadocs

Javadoc any helper methods you create in a style similar to the existing javadocs. If a method is overridden or implemented from a superclass or an interface, you may use `@Override` instead of writing javadocs. Any javadocs you write must be useful and describe the contract, parameters, and return value of the method. Random or useless javadocs added only to appease checkstyle will lose points.

Vulgar/Obscene Language

Any submission that contains profanity, vulgar, or obscene language will receive an automatic zero on the assignment. This policy applies not only to comments/javadocs, but also things like variable names.

Exceptions

When throwing exceptions, you must include a message by passing in a String as a parameter. **The message must be useful and tell the user what went wrong.** "Error", "BAD THING HAPPENED", and "fail" are not good messages. The name of the exception itself is not a good message.
For example:
Bad: `throw new IndexOutOfBoundsException(‘‘Index is out of bounds.’’);`
Good: `throw new IllegalArgumentException(‘‘Cannot insert null data into data structure.’’);`

In addition, you may not use try catch blocks to catch an exception unless you are catching an exception you have explicitly thrown yourself with the `throw new ExceptionName(‘‘Exception Message’’);` syntax (replacing `ExceptionName` and `Exception Message` with the actual exception name and message respectively).

Generics

If available, use the generic type of the class; do **not** use the raw type of the class. For example, use `new LinkedList<Integer>()` instead of `new LinkedList()`. Using the raw type of the class will result in a penalty.

Forbidden Statements

You may not use these in your code at any time in CS 1332.

- `package`
- `System.arraycopy()`
- `clone()`
- `assert()`
- `Arrays` class
- `Array` class

- `Thread` class
- `Collections` class
- `Collection.toArray()`
- Reflection APIs
- Inner or nested classes
- Lambda Expressions
- Method References (using the :: operator to obtain a reference to a method)

If you're not sure on whether you can use something, and it's not mentioned here or anywhere else in the homework files, just ask.

Debug print statements are fine, but nothing should be printed when we run your code. We expect clean runs - printing to the console when we're grading will result in a penalty. If you submit these, we will take off points.

JUnits

We have provided a **very basic** set of tests for your code. These tests do not guarantee the correctness of your code (by any measure), nor do they guarantee you any grade. You may additionally post your own set of tests for others to use on the Georgia Tech GitHub as a gist. Do **NOT** post your tests on the public GitHub. There will be a link to the Georgia Tech GitHub as well as a list of JUnits other students have posted on the class Piazza.

If you need help on running JUnits, there is a guide, available on Canvas under Files, to help you run JUnits on the command line or in IntelliJ.

Binary Search Tree

You are to code a binary search tree, `BST`, which is a collection of nodes, each having a data item and a reference pointing to a left and a right child nodes. The `BST` must follow the order property: for any given node, its left child's data and all of its children's data must be less than the current node while its right child's data and all of its children's data must be greater than the current node. In order to compare the data, all elements added to the tree must implement Java's generic `Comparable` interface.

It will have two constructors: a no-argument constructor (which should initialize an empty tree), and a constructor that takes in a collection of data to be added to the tree, and initializes the tree with this collection of data.

You may import Java's `LinkedList`/`ArrayList` classes for the 4 traversal methods, but only use them in these methods.

Recursion

Since trees are naturally recursive structures, all methods that are not $O(1)$ **must be implemented recursively**, except for level order traversal. You'll also notice that a lot of the public method stubs we've provided do not contain the parameters necessary for recursion to work, so these public methods act as "wrapper methods" for the user to use. You will have to write private recursive helper methods and call them in these wrapper methods. All of these helper methods must be private. To reiterate, **do not change the method headers for the provided methods**.

For methods that change the structure of the tree in some way, we highly recommend you use a technique taught in class called pointer reinforcement. It is not required, but it will make the homework cleaner, and it'll also help greatly when we get to a later homework.

Nodes

The binary search tree consists of nodes. A class `BSTNode` is provided to you. `BSTNode` has getter and setter methods to access and mutate the structure of the nodes.

Methods

You will implement all standard methods for a Java data structure (add, remove, etc.) in addition to a few other methods (such as traversals). You must follow the requirements stated in the javadocs of each method you implement.

Traversals

You will implement 4 different ways of traversing a tree: pre-order traversal, in-order traversal, post-order traversal, and level-order traversal. The first 3 MUST be implemented recursively; level-order is best implemented iteratively. For a level-order traversal, you may use Java's `Queue` interface (and an implementing class for it such as `LinkedList`).

Height

You will implement a method to calculate the height of the tree. The height of the tree is defined as the height of its root. The height of any given node is `max(left node's height, right node's height) + 1`. When doing this calculation, a leaf node has a height of 0 and a `null` node has a height of -1.

Comparable

As stated, the data in the BST must implement the `Comparable` interface. As you'll see in the files, the generic typing of the `BST` and `BSTNode` classes will enforce this `Comparable` data requirement. You use the interface by making a method call like `data1.compareTo(data2)`. This will return an `int`, and the value tells you how `data1` and `data2` are in relation to each other

- If the int is positive, then `data1` is larger than `data2`.
- If the int is negative, then `data1` is smaller than `data2`.
- If the int is zero, then `data1` equals `data2`.

Note that the returned value can be any integer in Java's `int` range, not just -1, 0, 1.

Grading

Here is the grading breakdown for the assignment. There are various deductions not listed that are incurred when breaking the rules listed in this PDF and in other various circumstances.

Methods:	
constructor	4pts
add	14pts
remove	20pts
get	5pts
contains	5pts
preorder	3pts
inorder	3pts
postorder	3pts
levelorder	3pts
height	3pts
clear	2pts
findPathBetween	10pts
Other:	
Checkstyle	10pts
Efficiency	15pts
Total:	100pts

Provided

The following file(s) have been provided to you. There are several, but we've noted the ones to edit.

1. `BST.java`

   This is the class in which you will implement the `BST`. Feel free to add private helper methods but **do not add any new public methods, inner/nested classes, instance variables, or static variables**.

2. `BSTNode.java`

   This class represents a single node in the tree. It encapsulates the `data`, and the `left` and `right` references. **Do not alter this file.**

3. `BSTStudentTest.java`

   This is the test class that contains a set of tests covering the basic operations on the `BST` class. It is not intended to be exhaustive and does not guarantee any type of grade. **Write your own tests to ensure you cover all edge cases.**

Deliverables

You must submit **all** of the following file(s) to the course Gradescope. Make sure all file(s) listed below are in each submission, as only the last submission will be graded. Make sure the filename(s) matches the filename(s) below, and that *only* the following file(s) are present. Do **NOT** submit `BSTNode.java` for this homework; if you do, your homework will not compile on Gradescope. If you resubmit, be sure only one copy of each file is present in the submission. If there are multiple files, do not zip up the files before submitting; submit them all as separate files.

Once submitted, double check that it has uploaded properly on Gradescope. To do this, download your uploaded file(s) to a new folder, copy over the support file(s), recompile, and run. It is your sole responsibility to re-test your submission and discover editing oddities, upload issues, etc.

1. `BST.java`