# 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 8**. 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, or method signatures.
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.LinkedList` for a Linked List 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 non-efficient unless that is absolutely required (and that case is extremely rare).
7. You must submit your source code, the `.java` files, not the compiled `.class` files.
8. 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.

# Concept Applications

For this assignment, you will be implementing five different coding questions that will allow you to apply your knowledge of data structures and algorithms. Descriptions of the expected outcome of each method are included in the Javadocs as well as the time and space complexity constraints. Be sure to consider the efficiencies of the data structures you have learned in this course when making your implementation.

## Space Complexity

While most of this course has primarily been about time complexity, in the real world, space complexity is also a concern. The way it's defined is very similar to how we measure time but with some notable caveats:

- The space used to store the inputs is not taken into account when determining the space used. For example, in a sorting algorithm like insertion sort, the space complexity is $O(1)$ because there is no **extra** space used to sort the array.
- The space complexity is the maximum amount of space being used at any given time, not overall throughout the algorithm.

# 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:** | |
|---|---|
| countAllPairs | 15pts |
| reverse | 15pts |
| isSymmetric | 15pts |
| findKLargest | 15pts |
| matchingBrackets | 15pts |
| **Other:** | |
| Checkstyle | 10pts |
| Efficiency | 15pts |
| **Total:** | 100pts |

# A note on JUnits

We have provided a **very basic** set of tests for your code, in `ConceptApplicationsStudentTests.java`. These tests do not guarantee the correctness of your code (by any measure), nor does it 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 T-Square under Resources, to help you run JUnits on the command line or in IntelliJ.

# Style and Formatting

It is important that your code is not only functional but is also written clearly and with good style. We will be checking your code against a style checker that we are providing. It is located in T-Square, under Resources, along with instructions on how to use it. We will take off a point for every style error that occurs. If you feel like what you wrote is in accordance with good style but still sets off the style checker please email Raymond Ortiz (rortiz9@gatech.edu) with the subject header of "CheckStyle XML".

## Javadocs

Javadoc any helper methods you create in a style similar to the existing Javadocs. Like the existing Javadocs, the Javadocs for your helper method(s) must describe well what the method does, what each parameter means (if any), and what the returned value is (if any). If a method is overridden or implemented from a superclass or an interface, you may use `@Override` instead of writing Javadocs.

## 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:

```
throw new PDFReadException("Did not read PDF, will lose points.");

throw new IllegalArgumentException("Cannot insert null data into data structure.");
```

## 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.

- `break` may only be used in switch-case statements
- `continue`
- `package`
- `System.arraycopy()`
- `clone()`
- `assert()`
- `Arrays` class
- `Array` class
- `Objects` class
- `Collections` class
- `Collection.toArray()`
- Reflection APIs
- Inner, nested, or anonymous classes

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

## Provided

The following file(s) have been provided to you. There are several, but you will only edit one of them.

1. `ConceptApplications.java`

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

2. `ConceptApplicationsStudentTests.java`

   This is the test class that contains a set of tests covering the basic operations on the `ConceptApplications` 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. The graphs used for these tests are shown above in the pdf.

3. `LinkedListNode.java`

   This class represents a single node in a linked list. It encapsulates `data` and the `next` reference. **Do not alter this file.**

4. `BinaryNode.java`

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

## Deliverables

You must submit **all** of the following file(s). Please make sure the filename matches the filename(s) below, and that *only* the following file(s) are present. T-Square does **not** delete files from old uploads; you must do this manually. Failure to do so may result in a penalty.

After submitting, be sure you receive the confirmation email from T-Square, and then download your uploaded files to a new folder, copy over the interfaces, recompile, and run. It is your responsibility to re-test your submission and discover editing oddities, upload issues, etc.

1. `ConceptApplications.java`