# *COMP 210*
# *Lab 5*

*February 14, 2017*

This week we'll work out some of the implementation for a basic node-based Binary Search Tree. We'll finish up the implementation in class; there is no homework this week due to the take home exam.

## *Lab 5*

Work up the following design by documenting, declaring, stubbing, and writing tests for the following. Use Eclipse to auto-generate

1. A *BST* interface for a Binary Search tree containing primitive integer type data. The interface should provide an isEmpty predicate, a predicate to check if a specific value is contained in the tree, mutators for insertion and removal of values, and a mutator that will balance the tree. Be sure to utilize and declare exceptions as needed.

2. A NodeBST class that implements the BST interface using the BinTree type from last week. Make constructors private and provide a static factory method for constructing a search tree from an array of unsorted integers. The factory method will construct an ideally balanced initial tree given the numbers.

Once the design is setup, turn your attention to the implementation. This time around we will implement NodeBST without making any changes to the BinTree hierarchy. This means adding private helper methods or static procedures to NodeBST rather than functionality to BinTree. For this implementation you are to take a top down approachby doing the following for each NodeBST method:

1. Analyze the problem and decide if any helpers could be used to make the implementation simpler.

2. Declare and stub each helper in the NodeBST class.

3. Complete the implementation of the method using the helpers.

The goal here is to have a complete enough picture of the specification for each helper than you could complete the method they're helping under the assumption that the helpers work to spec. At the end of lab, submit your work as assignment *lab5*.