1. [10 Points] Sorting with Trees. (This is a short review problem.)
   
   (a) Imagine that we have a binary search tree with \( n \) elements in it. Recall that an inorder traversal of the tree is a recursive traversal which takes a node \( v \) as input and does the following: Recursively traverse the left subtree of \( v \), print out the value of \( v \), recursively traverse the right subtree of \( v \). Note that in a binary search tree, an inorder traversal beginning at the root prints out the contents of the tree in increasing order. Explain briefly why an inorder traversal of a tree with \( n \) elements takes \( O(n) \) time regardless of how imbalanced the tree may be.

   (b) Professor Lai, who now teaches at the Massachusetts Institute of Typography, wants to use inorder traversals to get a fast sorting algorithm. Here is his idea: Given an unsorted array of \( n \) elements, first insert them one-at-a-time into an initially empty binary search tree. Then do an inorder traversal of the binary search tree. Each time the inorder traversal prints out a number, put that in the next available position in a new array. The new array will be the sorted version of the original array! What is the worst-case running time of Professor Lai’s algorithm? Poor Professor Lai. No tenure again!

   (c) Assume that instead of using regular binary search trees in Professor Lai’s approach, we were to use red-black trees. Analyze the running time of this “Red-Black Sorting algorithm”. Explain your analysis.

   (d) How about if B-Trees had been used instead of red-black trees? What would the running time be for the “B-Tree Sorting Algorithm”?

2. [65 Points] Implementing B-Trees! In this assignment you will implement B-Trees. In particular, your implementation should assume that \( t = 2 \). Your code should have a `Node` element similar to the one suggested in the lecture notes. Your program should support the operations `insert(x)`, which inserts the integer \( x \) into the tree, and `find(x)` which returns a boolean indicating whether or not \( x \) is in the tree. For insertion, use the second method described in class (in which insertion is done in a single pass down the tree). In addition, your program should have a method called `height()` which returns the height of the tree. (A tree with a single node has height 0.) Please do this programming exercise without looking at the book. The book’s pseudocode isn’t particularly clean or clear.

You should turn in a copy of your code and a sample run of a test program which inserts the even numbers from 0 to 20 (in order) into the tree. Next, invoke the `height()` method and print the height of your tree. Then, the test program tests to see if each number from 0 to 20 (odd and even) is in the tree and prints “true” or “false” for each such query.