# CSE 326 Summer 2006
# Assignment 3
Due: Wednesday, July 12

For all algorithm and data structure design problems, please provide elegant pseudocode and an adequate explanation of your methods. It is often helpful to include small examples demonstrating the method. Put your name at the top of each sheet of paper that you turn in.

1. Recommended: You should know how to do these problems, but you don't need to turn in a solution. Weiss, 4.2, 4.9, 4.19.

2. Weiss, 4.6. Use induction for your proof.

3. In class we saw how to implement the binary tree data structure using pointers. The data structure can also be implemented using arrays. Consider storing a complete binary tree of size $N$ in an array of size $N$. The nodes are laid out in level order in the array: the root is at index 1, its left and right children are at index 2 and 3, the left child of 2 is at 4, and so on, with each level in the tree written out left to right.

   This extends to non-complete trees by placing nodes at the index where they would be if the tree was complete. Non-existent nodes are given a special marker value in the array, say -1. For example, the tree on the right hand side of Figure 4.15 in Weiss, would be represented in an array of size $N = 15$ as:
   6 2 8 1 4 -1 -1 -1 -1 3 7 -1 -1 -1 -1

   (a) Give pseudocode for iterative implementations of the Search ADT operations `find`, `insert`, and `delete` using an array for the underlying binary tree data structure. Your methods should be of the form `find( array A, integer N, integer key ) => integer`.

   (b) How does the pointer implementation compare to the array implementation in terms of cache behavior? Your answer should depend on the size of the tree.

4. In this problem you will design a recursive AVL deletion procedure and execute it on a simple example.

   (a) Design a recursive AVL tree deletion procedure. You may assume that rotation and height calculation procedures are available to you. Your procedure should be modeled on the recursive deletion procedure in Figure 4.25 of the text. After each recursive call and before returning, rotations must be done to rebalance the tree. See the way this is done for AVL tree insertion in Figure 4.39 of the text.

(b) Demonstrate your algorithm by doing all the rotations needed to delete 10 from the following AVL tree. If there is more than one rotation show the result after each single or double rotation.