# COP4530: Data Structures, Algorithms, and Generic Programming
# Spring 2008

## Proving Binary Search Tree Properties: An example

Properties of binary trees, or of algorithms that use them, are often proven using arguments, which may be different from what you are typically used to. We give an example below.

*Prove that an inorder traversal of a binary search tree visits nodes in ascending order.*

**Proof**

Assume that this statement is not true. The there exists a binary search tree with nodes x and y, such that $y < x$, but x is visited before y in an inorder traversal.

We will consider the following three cases, which cover all possibilities: (i) x is an ancestor of y, (ii) y is an ancestor of x, and (iii) neither x nor y is the ancestor of the other.

Case (i): In this case, y is in the left subtree of x, since $y < x$. But an inorder traversal visits all nodes in the left subtree of a node, before visiting that node. So y should be visited before x. Since that did not happen, x cannot be the ancestor of y.

Case (ii): Since $y < x$, x must be in the right subtree of y. But an inorder traversal visits a node before visiting any node in its right subtree. So y should be visited before x. Since that did not happen, y cannot be the ancestor of x.

Case (iii): Let z be their common ancestor at the highest level. (A common ancestor does exists, since the root is a common ancestor of all other nodes, and neither x nor y is the root, from the assumption for this case.) x and y must lie on different subtrees of z. (Otherwise, there will be a node at a higher level that separates x and y into different subtrees; in which case z would not be the highest level common ancestor.) Since $y < x$, y must be in the left subtree of z and x must be in the right subtree of z. An inorder traversal would visit all the nodes in the left subtree before it visited any node in the right subtree, and so y would be visited before x is. Thus case (iii) is not possible.

None of the three cases is possible, which is impossible. So our assumption is false and an inorder traversal visits the nodes of a binary search tree in ascending order. Q.E.D.

## Proving Impossibility Results: An example

It is often useful to prove that certain algorithms are impossible, within a specified time complexity. In this course, we will use the fact that comparison based sorting cannot be done in less than $\Theta(n \log n)$ time. You will prove this fact in COP4531, but we will assume it is true, in order to prove that certain operations cannot be performed very fast on certain data structures (*under this model*). We give an example below.

*It is impossible to develop a max-heap DeleteMax algorithm that takes constant time.*

**Proof**

Assume that a constant time algorithm exists for this operation. We will show that we can then develop a sorting algorithm that takes O(n) time, as shown below. Since O(n) sorting is impossible (under our model), our assumption must be false.

```
ImpossibleSort(Array, n)
{
      InitializeHeap(Array, n); // O(n) time
      for(i=0; i<n; i++)
      {
            cout << FindMax() // O(1) time
            DeleteMax() // O(1) time
      }
}
```

The loop above executes n times for O(n) time complexity, and the heap initialization takes O(n) time. So the above algorithm outputs the data in descending order in O(n) time, which is impossible. Q.E.D.

Note that in the above algorithm, it is clear that the output is in ascending order. If this were not obvious, then we would need to prove this fact too.