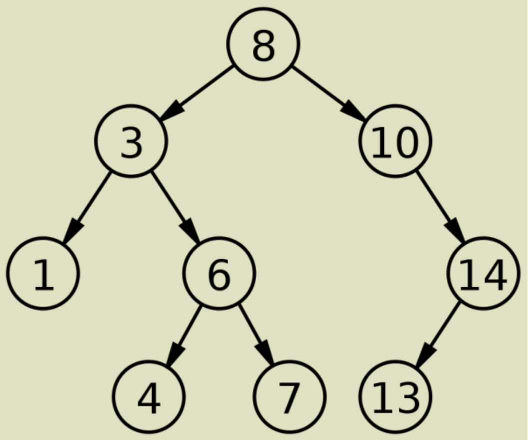
Binary Trees: Search & Insert

Computer Science Department
University of Central Florida

COP 3502 – Computer Science I

Binary Search Tree



- Binary Search Trees
  - Ordering Property:
    - For each node N, all the values stored in the left subtree of N are LESS than the value stored in N.
    - Also, all the values stored in the right subtree of N are GREATER than the value stored in N.
    - Why might this property be a desireable one?
      - **Searching for a node is super fast!**
    - Normally, if we search through n nodes, it takes O(n) time
    - But notice what is going on here:
      - This ordering property of the tree tells us where to search
      - We choose to look to the left or look to the right of a node
      - We are **HALVING** the search space …**O(log n)** time

Binary Search Tree: Searching

- Binary Search Trees
  - Searching for a node:
    - Algorithm:

1) **IF** the tree is NULL, return false.

   ELSE

2) Check the root node. If the value we are searching for is in the root, return true.
3) If not, if the value is less than that stored in the root node, recursively search in the left subtree.
4) Otherwise, recursively search in the right subtree.

Binary Search Tree: Searching

- Binary Search Trees
  - Searching for a node (Code):

```
int find (struct tree_node *current_ptr, int val) {
        // Check if there are nodes in the tree.
        if (current_ptr != NULL) {
                // Found the value at the root.
                if (current_ptr->data == val)
                        return 1;
                // Search to the left.
                if (val < current_ptr->data)
                        return find(current_ptr->left, val);
                // Or...search to the right.
                else
                        return find(current_ptr->right, val);
        }
        else
                return 0;
}
```

Binary Search Tree: Creation

- Insertion into a Binary Search Tree
  - Before we can insert a **node** into a BST, what is the one obvious thing that we must do?
  - We have to actually **create** the node that we want to insert
    - malloc space for the node
    - And save appropriate data value(s) into it
  - Here's our struct from last time:

```
struct tree_node {
        int data;
        struct tree_node *left_child;
        struct tree_node *right_child;
}
```

Binary Search Tree: Creation

- Creating a Binary Search Tree
  - In main, we simply make a **pointer of type struct tree_node** and initialize it to NULL
    - `struct tree_node *my_root = NULL;`
    - So this is the ROOT of our tree
  - You then get your values to insert into the tree
    - This could be automated
    - You could have the user enter a value(s)
    - However you want (this really isn't that important)
  - We then call the `create_node` function to create a new node with this specific value

Binary Search Tree: Creation

- Creating a Binary Search Tree
  - `create_node` function:

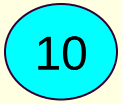
```
struct tree_node* create_node(int val) {

       // Allocate space for the node
       struct tree_node* temp;
       temp = (struct tree_node*)malloc(sizeof(struct tree_node));

       // Initialize the fields
       temp->data = val;
       temp->left = NULL;
       temp->right = NULL;

        // Return a pointer to the created node.
       return temp;
}
```

Binary Search Tree: Insertion

- Insertion (of nodes) into a Binary Search Tree
  - Now that we have nodes, it is time to insert!
  - Binary Trees must maintain their ordering property
    - Smaller items to the left of any given root
    - And greater items to the right of that root
  - So when we insert, we MUST follow these rules
  - You simply start at the root and either
    1) Go right if the new value is greater than the root
    2) Go left if the new value is less than the root
  - Keep doing this till you come to an empty position
  - An example will make this clear…

Binary Search Tree: Insertion

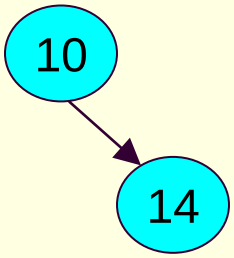
- Insertion into a Binary Search Tree
  - Let's assume we insert the following data values, in their order of appearance into an initially empty BST:
    - 10, 14, 6, 2, 5, 15, and 17



  - **Step 1**:
    - Create a new node with value 10
    - Insert node into tree
    - The tree is currently empty
    - New node becomes the root

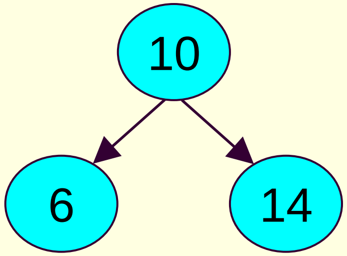
Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - 10, 14, 6, 2, 5, 15, and 17
  - **Step 2**:
    - Create a new node with value 14
    - This node belongs in the right subtree of node 10
      - Since 14 > 10
    - The right subtree of node 10 is empty
      - So node 14 becomes the right child of node 10



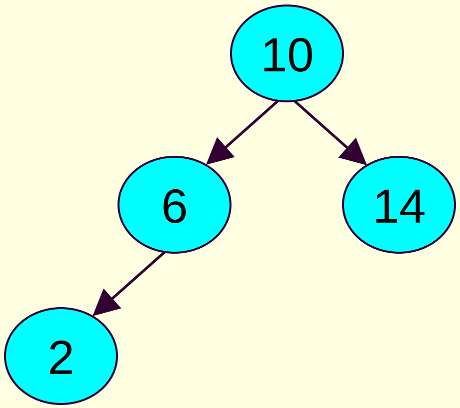
Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - 10, 14, 6, 2, 5, 15, and 17
  - **Step 3**:
    - Create a new node with value 6
    - This node belongs in the left subtree of node 10
      - Since 6 < 10
    - The left subtree of node 10 is empty
      - So node 6 becomes the left child of node 10



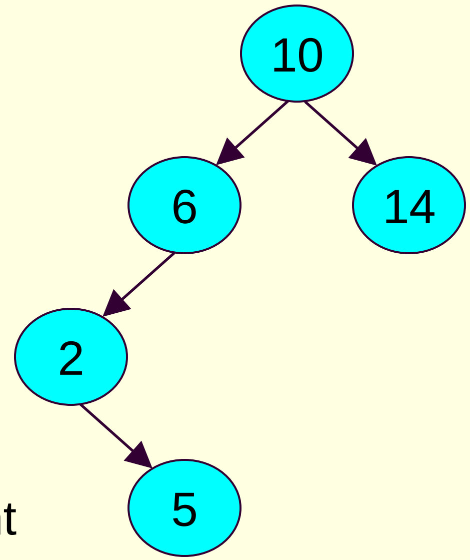
Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - 10, 14, 6, 2, 5, 15, and 17
  - **Step 4**:
    - Create a new node with value 2
    - This node belongs in the left subtree of node 10
      - Since 2 < 10
    - The root of the left subtree is 6
    - The new node belongs in the left subtree of node 6
      - Since 2 < 6
    - So node 2 becomes the left child of node 6



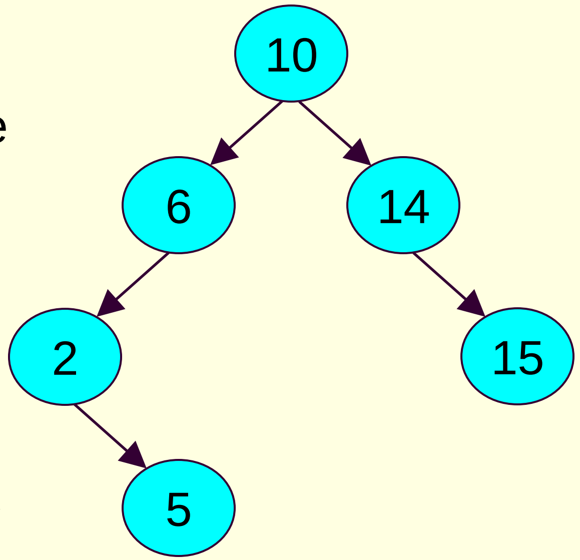
Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - 10, 14, 6, 2, 5, 15, and 17
  - **Step 5**:
    - Create a new node with value 5
    - This node belongs in the left subtree of node 10
      - Since 5 < 10
    - The new node belongs in the left subtree of node 6
      - Since 5 < 6
    - And the new node belongs in the right subtree of node 2
      - Since 5 > 2



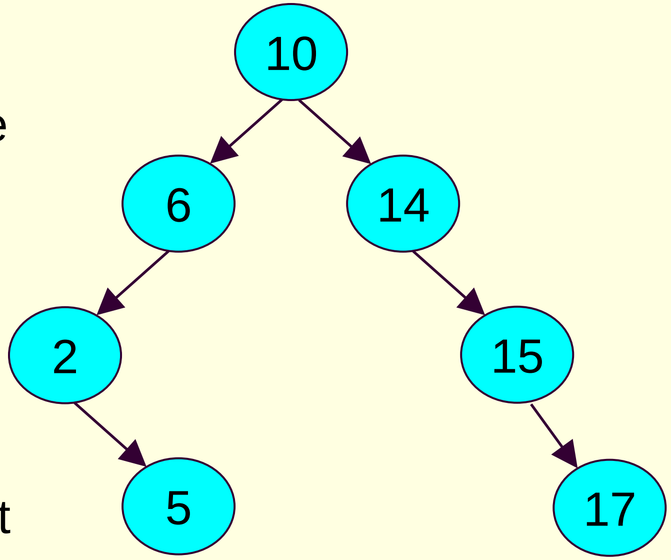
Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - 10, 14, 6, 2, 5, 15, and 17
  - **Step 6**:
    - Create a new node with value 15
    - This node belongs in the right subtree of node 10
      - Since 15 > 10
    - The new node belongs in the right subtree of node 14
      - Since 15 > 14
    - The right subtree of node 14 is empty
    - So node 15 becomes right child of node 14



Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - 10, 14, 6, 2, 5, 15, and 17
  - **Step 7**:
    - Create a new node with value 17
    - This node belongs in the right subtree of node 10
      - Since 17 > 10
    - The new node belongs in the right subtree of node 14
      - Since 17 > 14
    - And the new node belongs in the right subtree of node 15
      - Since 17 > 15



Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - Here's our basic plan to do this recursively:
    1) If the tree is empty, just return a pointer to a node containing the new value.
    2) Otherwise, see which subtree the node should be inserted by comparing the value to insert with the value stored at the root.
    3) Based on this comparison, **<u>recursively</u> either insert into the right subtree, or into the left subtree.**

Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - And here's the matching code:

```
struct tree_node* insert(struct tree_node *root, struct tree_node *element) {

        // Inserting into an empty tree.
        if (root == NULL)
                return element;
        else {
                // element should be inserted to the right.
                if (element->data > root->data)
                        root->right = insert(root->right, element);
                // element should be inserted to the left.
                else
                        root->left = insert(root->left, element);
                // Return the root pointer of the updated tree.
                return root;
        }
}
```

Binary Search Tree: Creation

- Creating a Binary Search Tree
  - What we get from this:
    - Creating a BST is really nothing more than a series of insertions (calling the insert function over and over)
  - You simply get the values
  - Create the nodes
  - And then call this insert function over and over
    - For every node

Brief Interlude: Human Stupidity

Binary Search Tree: Sum Nodes

- Summing the Nodes of a Binary Search Tree
  - How would you do this?
  - If it is not clear, think about how you did this with linked lists.
    - How did you sum the nodes in a linked list?
    - You simply traversed the list and summed the values
  - Similarly, we traverse the tree and sum the values
  - How do we traverse the tree?
    - We already went over that
    - You have three traversal options: preorder, inorder, postorder...so choose one

Binary Search Tree: Sum Nodes

- Summing the Nodes of a Binary Search Tree
  - But it's really even easier than this!
  - All we do is add the values (root, left, and right) and then return the answer
  - Here's the code, and notice how succinct it is:

```
int add(struct tree_node *current_ptr) {
        if (current_ptr != NULL)
                return current_ptr->data +
                add(current_ptr->left)+ add(current_ptr->right);
        else
                return 0;
}
```

Binary Search Tree: Search

- Search of an **<u>Arbitrary</u>** Binary Tree
  - We've seen how to search for a node in a binary search tree
  - Now consider the problem if the tree is NOT a binary search tree
    - It <u>does not have the ordering property</u>
  - You could simply perform one of the traversal methods, checking each node in the process
    - Unfortunately, this won't be O(log n) anymore
    - It degenerates to O(n) since we possibly check all nodes

Binary Search Tree: Search

- Search of an **Arbitrary** Binary Tree
  - Here's another way we could do this
  - The whole idea here is to be comfortable with binary trees:

```
int Find(struct tree_node *current_ptr, int val) {
        if (current_ptr != NULL)  {
                if (current_prt->data == val)
                        return 1;
                return (Find(current_ptr->left, val) ||
                         Find(current_ptr->right, val))
        }
        else
                return 0;
}
```

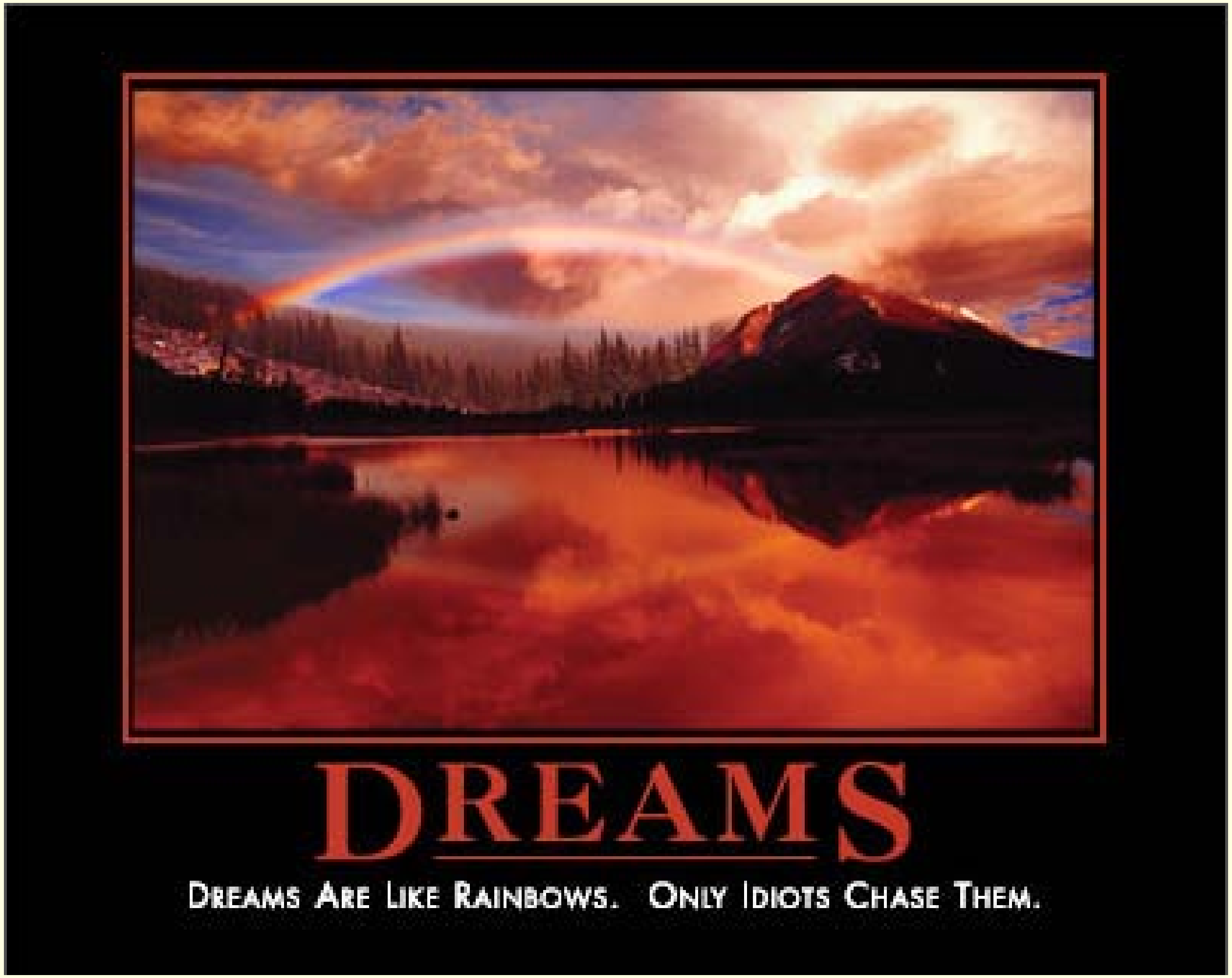
Binary Trees: Search & Insert

- Class Exercise:
  - Write a function that prints out all the values in a **binary tree** that are greater than or equal to a value passed to the function.
  - Here is the prototype:
  - `void PrintBig(struct tree_node *current_ptr, int value);`

Binary Trees: Search & Insert

WASN'T THAT FABULOUS!

Daily Demotivator



Binary Trees: Search & Insert

Computer Science Department
University of Central Florida

COP 3502 – Computer Science I