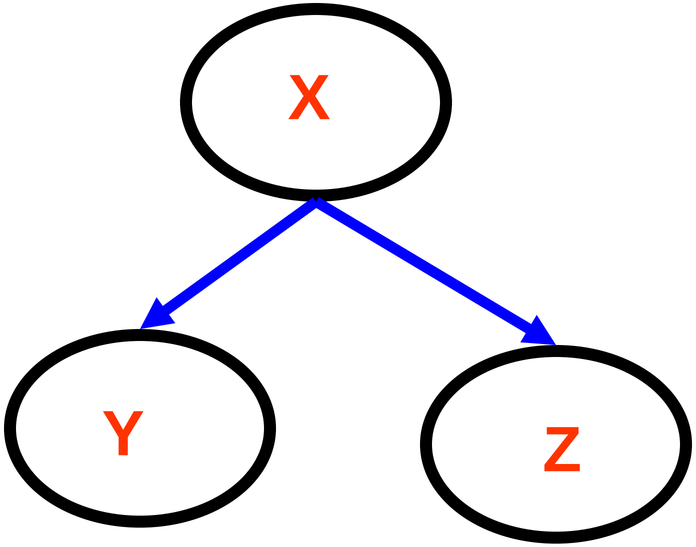
National University of Computer & Emerging Sciences

Trees – Binary Search Trees

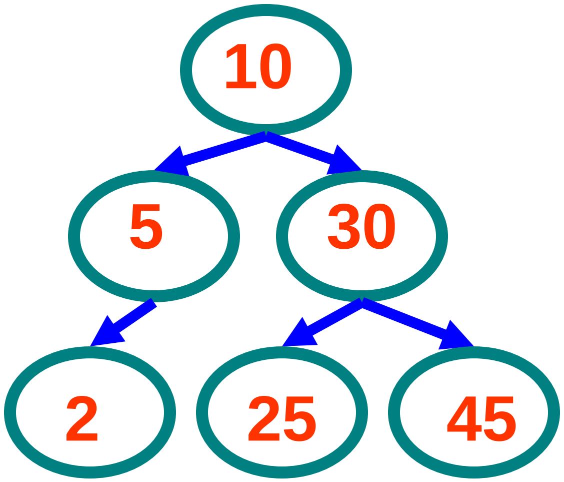
Binary Search Tree

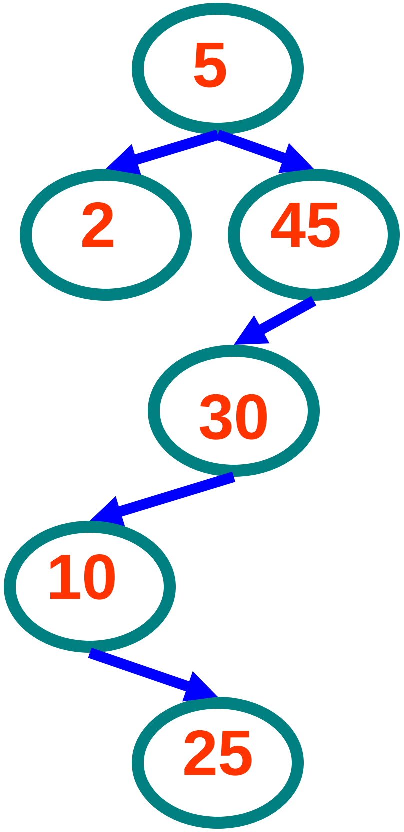
- For every node, X, in the tree,
  - the values of all the keys in its left subtree are *smaller than* the key value of X,
  - the values of all the keys in its right subtree are *larger than* the key value of X.
  - Example
    - X > Y
    - X < Z



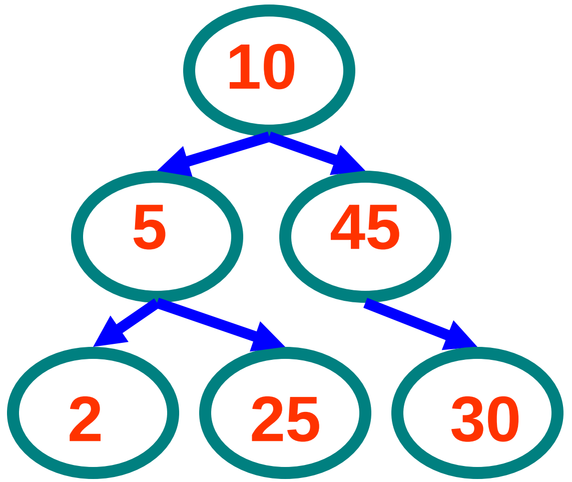
Binary Search Trees

- Examples



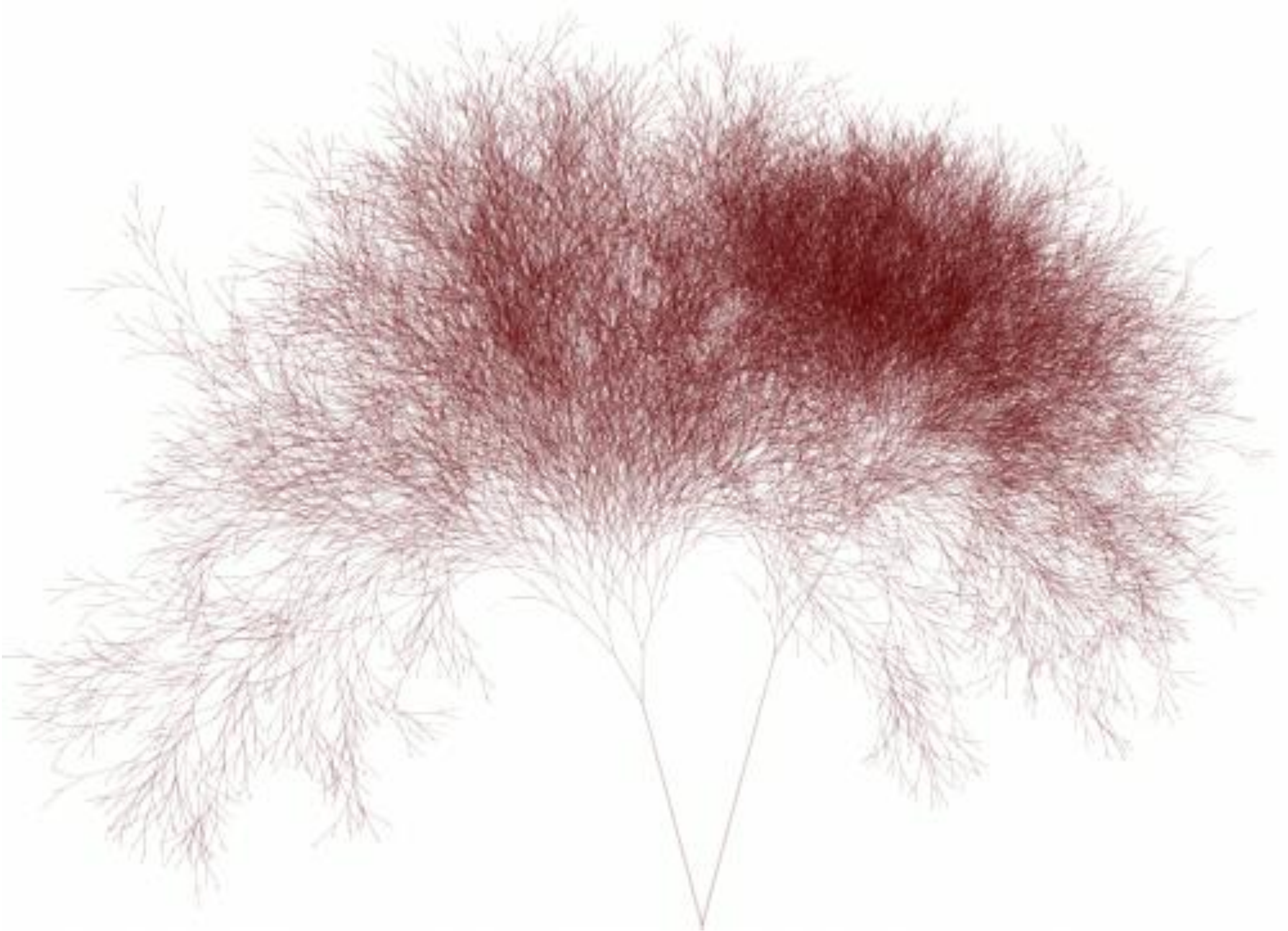


Binary search trees



Not a binary search tree. Why?

2,147,483,647 Nodes



Binary Search Tree Operations

There are many operations one can perform on a *binary search tree*.

a) **Creating** a binary search tree
b) **Finding** a node in a binary search tree

c) **Inserting** a node into a binary search tree
d) **Deleting** a node in a binary search tree.

e) **Traversing** a binary search tree.

We will briefly cover all of these operations with their general algorithms, implementation, and examples.

Creating a Binary (Search) Tree

- We will use a simple class that implements a binary tree to store integer values.
- We create a class called *IntBinaryTree.*

The basic node of our binary tree has the following ***struct*** declaration.

```
struct TreeNode
{
    int value;
    TreeNode *left;
    TreeNode *right;
}
```

The class IntBinaryTree declaration is -

IntBinaryTree.h

```
class IntBinaryTree
{
private:
        struct TreeNode
        {
                int value;
                TreeNode *left;
                TreeNode *right;
        };
```

```
        TreeNode *root;
        void destroySubTree(TreeNode *);
        void deleteNode(int, TreeNode *&);
        void makeDeletion(TreeNode *&);
        void displayInOrder(TreeNode *);
        void displayPreOrder(TreeNode *);
        void displayPostOrder(TreeNode *);

    public:

        IntBinaryTree()        // Constructor
                { root = NULL; }
        ~IntBinaryTree()       // Destructor
                { destroySubTree(root); }
        void insertNode(int);
        bool searchNode(int);
        void remove(int);
        void showNodesInOrder(void)
                {       displayInOrder(root); }
        void showNodesPreOrder()
                {       displayPreOrder(root); }
        void showNodesPostOrder()
                {       displayPostOrder(root); }
};
```

The *root* pointer is the pointer to the binary tree. This is similar to the *head* pointer in a linked list.

The *root* pointer will point to the first node in the tree, or to NULL (if the tree is empty).

It is initialized in the constructor.

The destructor calls *destroySubTree*, a private member function, that **recursively** deletes **all** the nodes in the tree.

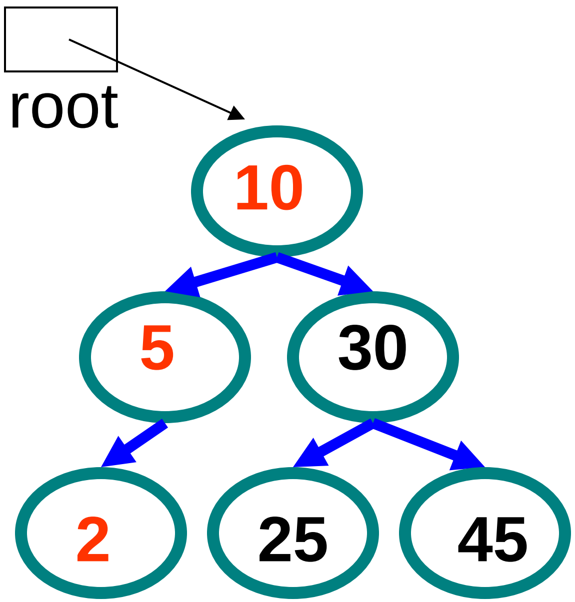
Finding a node in a binary search tree

- Recall that a BST has the following key property (invariant):
  - Smaller values in left sub-tree
  - Larger values in right sub-tree

- To search a node, we use this property!

Example Binary Searches

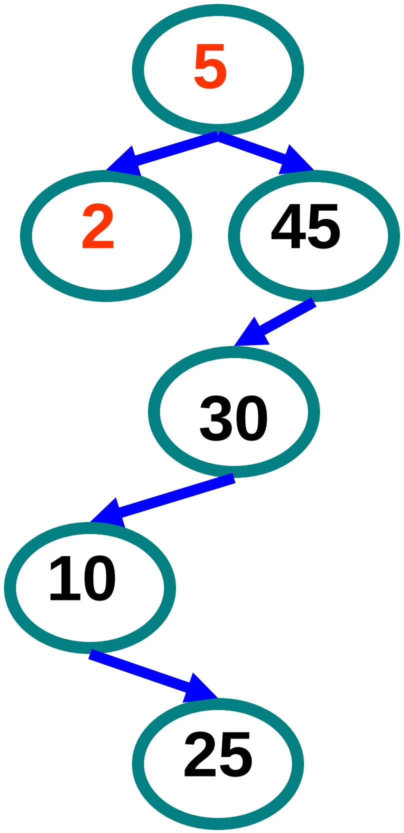
- Find (2)



10 > 2, left

5 > 2, left

2 = 2, found

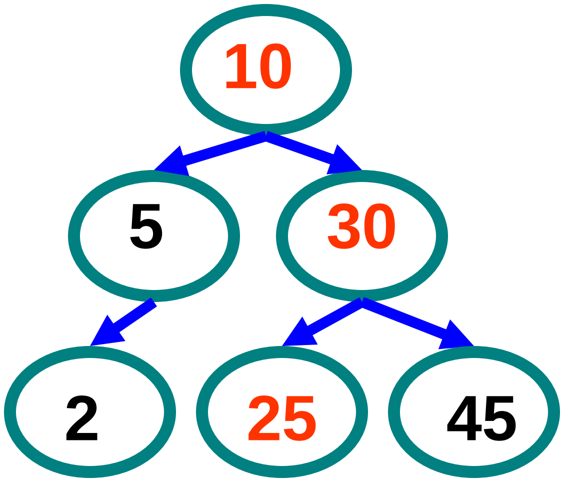


5 > 2, left

2 = 2, found

Example Binary Searches

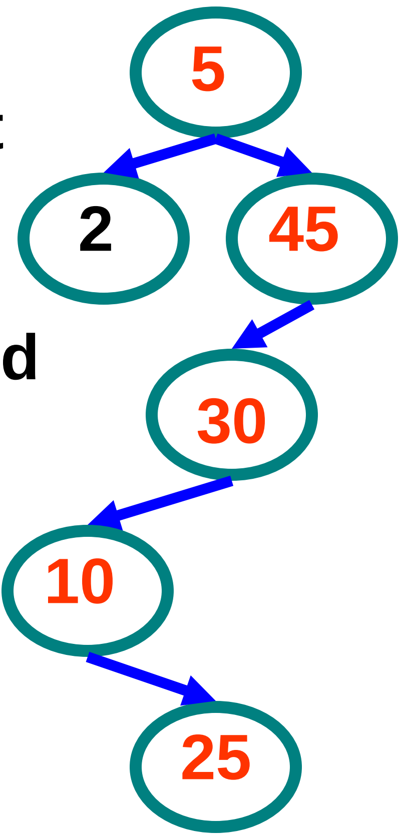
- Find (25)



10 < 25, right

30 > 25, left

25 = 25, found



5 < 25, right

45 > 25, left

30 > 25, left

10 < 25, right

25 = 25, found

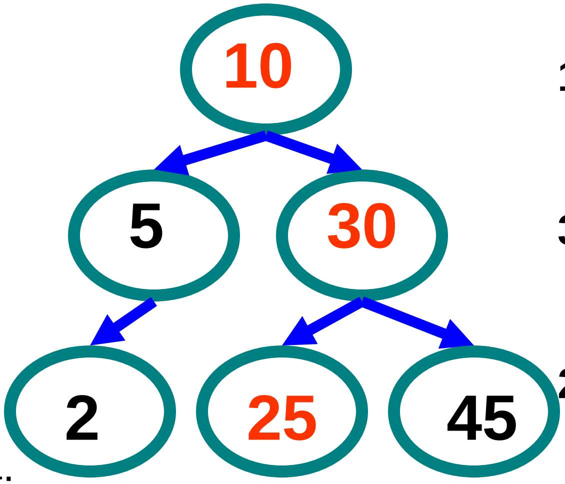
Searching the Tree

The IntBinaryTree class has a public member function called searchNode, that returns true if a value is found in the tree, or false otherwise.

The function *starts* at the root node, and traverses the tree, until it finds the search value, or *runs out* of nodes.

```
bool IntBinaryTree::searchNode(int num)
{
        TreeNode *nodePtr = root;

        while (nodePtr)
        {
                if (nodePtr->value == num)
                        return true;
                else if (num < nodePtr->value)
                        nodePtr = nodePtr->left;
                else
                        nodePtr = nodePtr->right;
        }
        return false;
}
```



10 < 25, right

30 > 25, left

25 = 25, found

Inserting a node into a binary search tree

- The code to insert a new value in the tree is fairly straightforward.

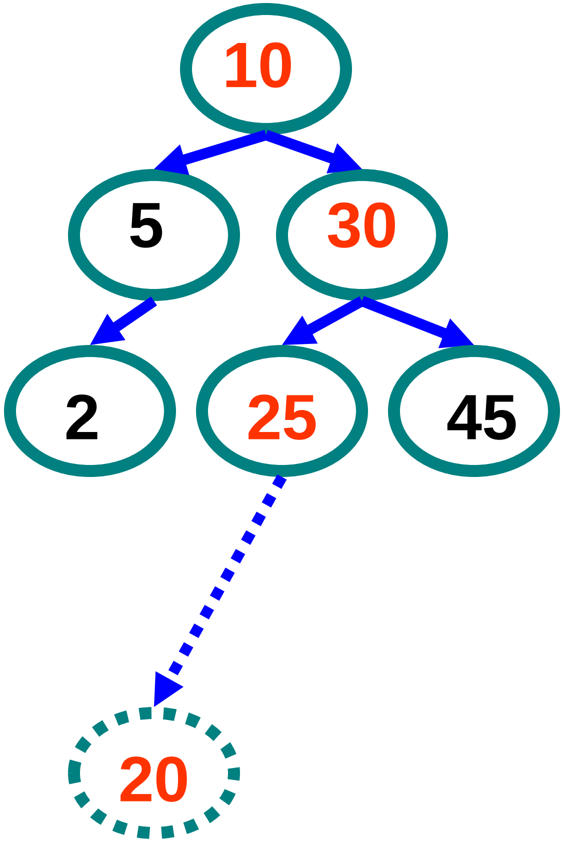
- First, a new node is allocated, and its *value* member is initialized with the new value.

Binary Search Tree – Insertion

- Algorithm
  1. Perform search for value X
  2. Search will end at node Y (if X not in tree)
  3. If X < Y, insert new leaf X as new left subtree for Y
  4. If X > Y, insert new leaf X as new right subtree for Y
- Observations
  - Insertions may unbalance tree

Example Insertion

- Insert (20)



10 < 20, right

30 > 20, left

25 > 20, left

Insert 20 on left

Note, we assume that our binary tree will store no duplicate values

```
void IntBinaryTree::insertNode(int num)
{
	TreeNode *newNode,	// Pointer to a new node
		*nodePtr;	// Pointer to traverse the tree

	// Create a new node
	newNode = new TreeNode;
	newNode->value = num;
	newNode->left = newNode->right = NULL;

	if (!root)	// Is the tree empty?
		root = newNode;
	else
	{
		nodePtr = root;
```

```
while (nodePtr != NULL)
{       if (num < nodePtr->value)
        {   if (nodePtr->left)
                    nodePtr = nodePtr->left;
                else
                {   nodePtr->left = newNode;
                    break;
                }
        }

        else if (num > nodePtr->value)
        {   if (nodePtr->right)
                    nodePtr = nodePtr->right;
                else
                {   nodePtr->right = newNode;
                    break;
                }
        }
        else
        { cout << "Duplicate value found in tree.\n";
      break;
        }
}
}
}
```

Program

```
// This program builds a binary tree with 5 nodes.
// The SearchNode function determines if the
// value 3 is in the tree.
#include <iostream.h>
#include "IntBinaryTree.h“

void main(void)
{
        IntBinaryTree tree;

        cout << "Inserting nodes.\n";
        tree.insertNode(5);
        tree.insertNode(8);
        tree.insertNode(3);
        tree.insertNode(12);
        tree.insertNode(9);
```

```
        if (tree.searchNode(3))
                cout << "3 is found in the tree.\n";
        else
                cout << "3 was not found in the tree.\n";
}
```

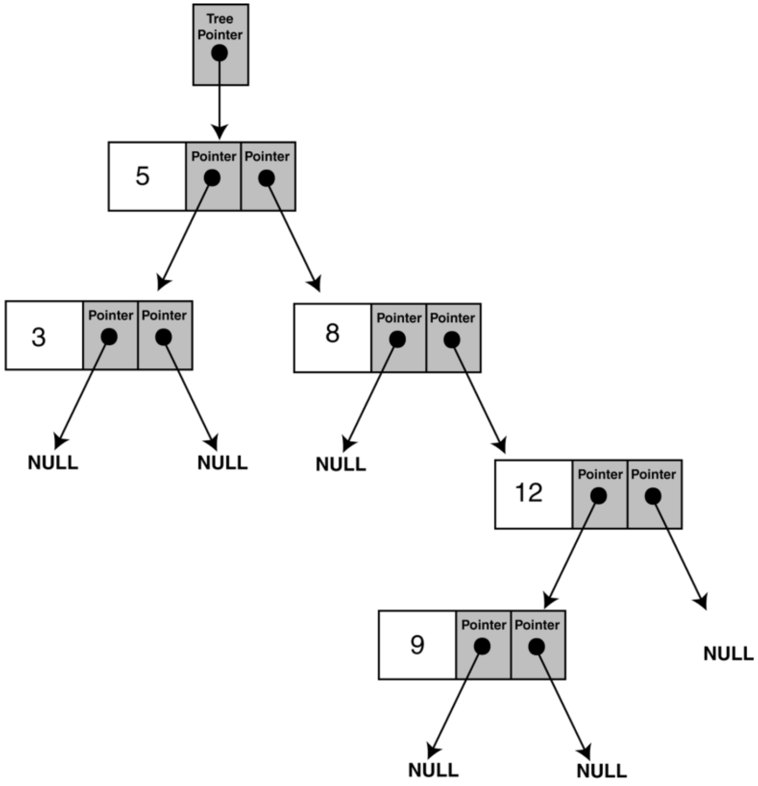
Program Output

Inserting nodes.

3 is found in the tree.

Program

Figure shows the structure of the binary tree built by the program.



Note: The <u>shape</u> of the tree is determined by the order in which the values are <u>inserted</u>. *The root node in the diagram above holds the value 5 because that was the <u>first value inserted</u>.*