Binary Search Trees

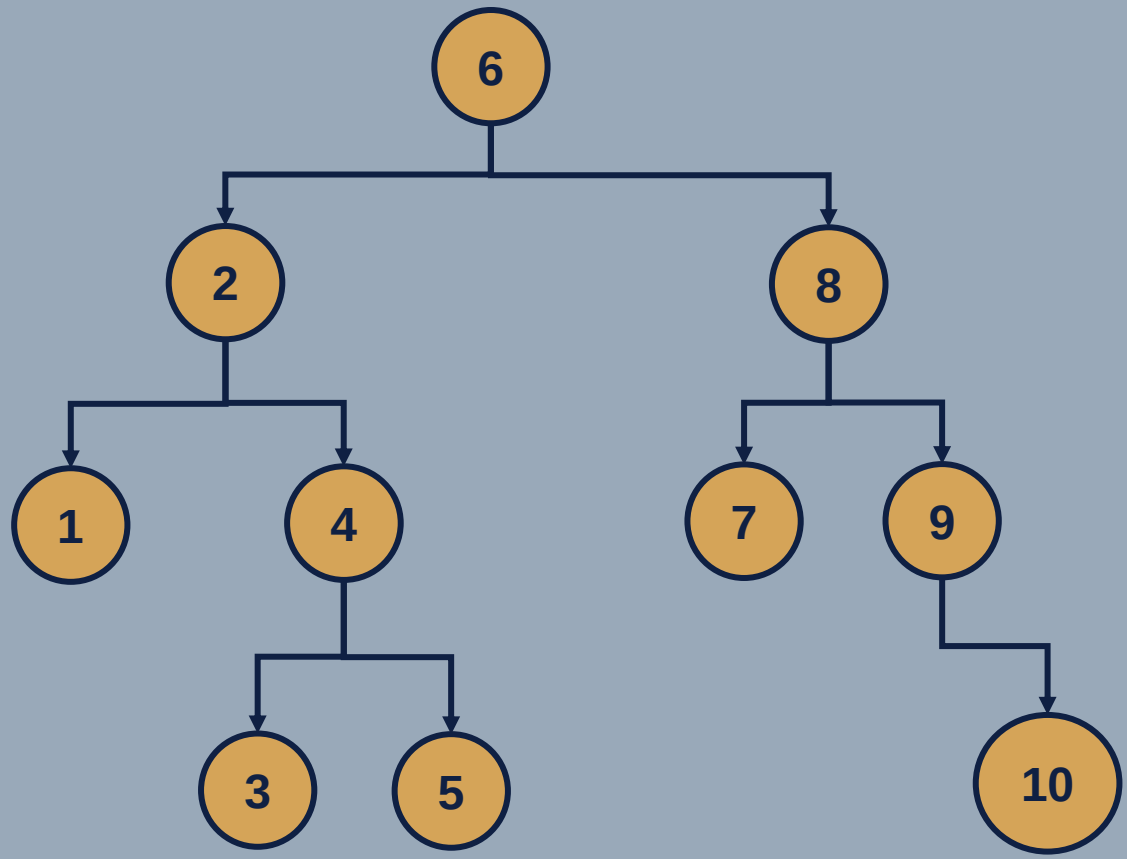
Lesson 6.2

Learning Objectives

LO 6.2.1	**Insert** correctly an element into a binary search tree
LO 6.2.2	**Perform** element search in a binary search tree
LO 6.2.3	**Delete** an element successfully in a binary search tree
LO 6.2.4	**State** the advantage of BST over a generic binary tree

Binary Search Tree

- A binary search tree (BST) is a binary tree with the following properties:
    1. All items in the left subtree are less than the root
    2. All items in the right subtree are greater than or equal to the root.
    3. Each subtree is itself a binary search tree.



Binary Search Tree Operations

- *Inserting an element into the BST*

```
algorithm insertBST (root, newNode)
    Insert node containing new data into BST using recursion.
    Pre:    root is address of current node in a BST.
            newNode is address of node containing data.
    Post:   newNode inserted into the tree
    Return: address of potential new tree root.
    if (empty tree)
        set root to newNode
        return newNode
    end if
    if (data of newNode < data of root)
        return insertBST (left subtree, newNode)
    else
        return insertBST (right subtree, newNode)
    end if
end insertBST
```

LO 6.2.1 Insert correctly an element into a binary search tree

Insert the following respectively into an initially empty binary search tree:

14, 23, 7, 10, 56, 70, 80, 66, 33

Binary Search Tree Operations

- *Searching an element into the BST*

```
algorithm searchBST (root, elem)
    Search node containing the data elem in the BST using recursion.
    Pre:    root is address of current node in a BST.
            elem is the data searched for.
    Return: address of node containing elem if it exists, otherwise null.
    if (root is null)
        return null
    end if
    if (elem < data of root)
        return searchBST (left subtree, elem)
    else if (elem > data of root)
        return searchBST (right subtree, elem)
    else
        return root
    end if
end searchBST
```

LO 6.2.2 Perform element search in a binary search tree

Conduct an element search on the binary search tree that you have just constructed earlier. How many comparisons did it take to search:

1. 80
2. 33
3. 100
4. 20
5. 12

Binary Search Tree Operations

- *Deleting an element from the BST*

```
algorithm deleteBST (root, dltKey)
    This algorithm deletes a node from a BST.
    Pre:      root is reference to node to be deleted
              dltKey is key of node to be deleted
    Post:     node deleted; if dltKey not found, root unchanged
    Return:   true if node deleted, false if not found
    if (empty tree)
        return false
    end if
    if (dltKey < root)
        return deleteBST (left subtree, dltKey)
    else if (dltKey > root)
        return deleteBST (right subtree, dltKey)
    else
        Delete node found--test for leaf node
        if (no left subtree)
            make right subtree the root
            return true
    ……
                else if (no right subtree)
                    make left subtree the root
                    return true
                else
                    Node to be deleted not a leaf.
                    Find largest node on left subtree.
                    set root data to the largest data in
                        the left subtree
                    return deleteBST (left subtree,
                        data of root)
                end if
            end if
    end deleteBST
```

LO 6.2.3 Delete an element successfully in a binary search tree

Delete the elements below sequentially on the binary search tree you have constructed earlier:

1. 14
2. 23
3. 80

LO 6.2.4 State the advantage of BST over a generic binary tree

What are the advantages of using BST over a generic binary tree?