# Algorithms and Data Structures

AVL Tree

Robert Horvick

www.pluralsight.com

# Outline

- **Binary Tree Overview**
  - Unbalanced and Balanced
- **AVL Tree**
  - Base Tree Implementation
- **Balancing Algorithms**
  - Right Rotation
  - Left Rotation
  - Right-Left Rotation
  - Left-Right Rotation
- **Demo: Balanced Tree Viewer**

# Binary Tree Overview

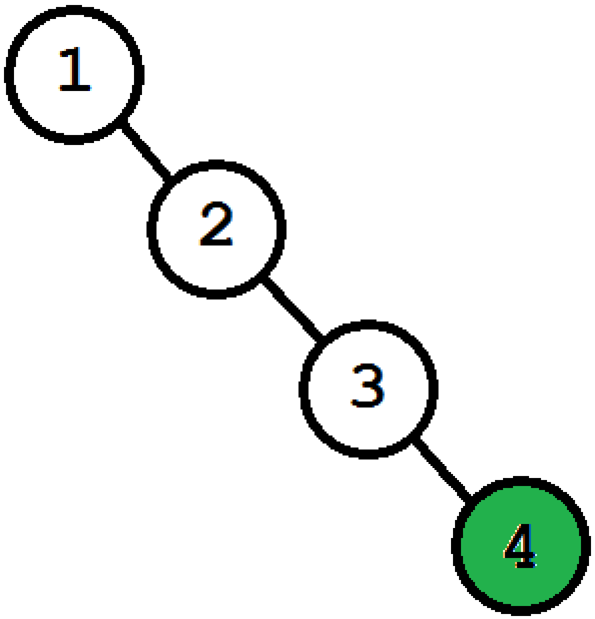
- **Collection that stores data in a tree structure**
- **Each node in the tree contains**
  - Value
  - Left and Right pointer
- **Navigation Rules**
  - Smaller values on left
  - Equal or larger values on right
- **Standard tree operations**
  - Insertion
  - Deletion
  - Search
  - Clear
  - Enumeration

# AVL Tree Overview

- **Self-balancing binary tree Invented by Adelson-Velsky & Landis (1962)**
- **Similar to Binary Tree**
  - Follows all binary tree structural constraints
  - Search and Enumeration are identical to Binary Tree
  - Insertion and Deletion differ only in running the Balance algorithm
- **New tree concepts**
  - Self-Balancing
  - Height
  - Balance Factor
  - Right/Left Heavy

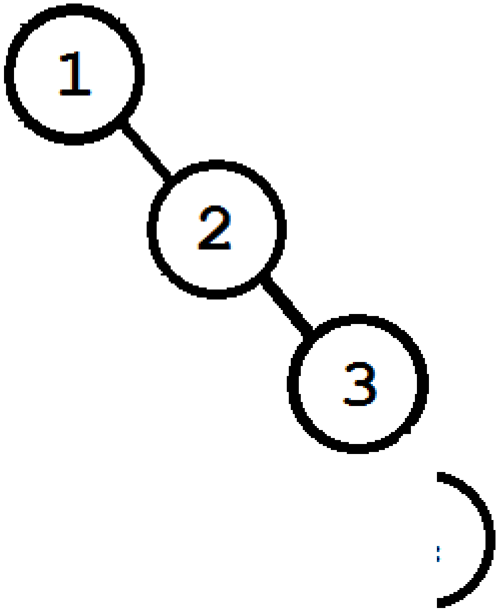
# Unbalanced Binary Tree

- **Can become a singly linked list in worst case**
  - O(n) search performance
- **Example**
  - Inserting values 1, 2, 3, 4 into binary tree
- **Binary Tree became Linked Listed**
- **Example:**
  - Searching for value 4
- **4 comparisons were necessary**
  - O(n) search performance



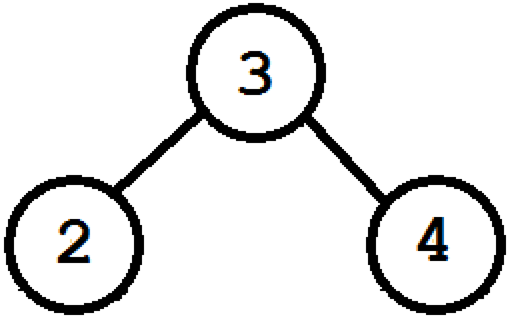
# Balanced Binary Tree

- **The tree remains balanced as nodes are inserted or deleted**
  - O(log n) search performance
- **Height of left and right tree differ by at most 1**
- **Example**
  - Inserting values 1, 2, 3, 4 into binary tree
- **Example:**
  - Searching for value 4
- **3 comparisons were necessary**
  - O(log n) becomes clearer with larger data sets



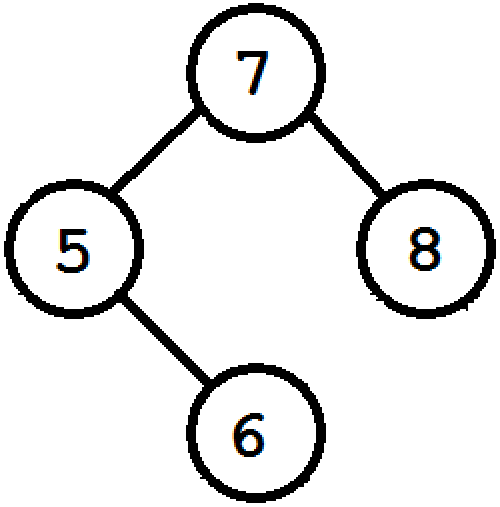
# Insertion

- **Insertion is identical to Binary Tree insertion**
  - Lesser values added to the left
  - Greater or Equal values added to the right
- **The balance algorithm runs for every parent node after the insertion**
- **Example: Inserting 4, 2, 3**
  - 4 becomes the root
  - 2 becomes a child of 4
  - Balancing is performed



# Deletion

- **Deletion is identical to Binary Tree deletion**
  - The node to delete is found
  - Child nodes are moved to retain tree rules
- **The balance algorithm runs for every parent node after the deletion**
- **Example: Deleting 4**
  - 4 is found
  - 4 is deleted
  - Balancing is performed



# AVL Tree

```
public class AVLTree<T> : IEnumerable<T>
    where T : IComparable
{
    public AVLTreeNode<T> Head { get; }

    public void Add(T value);
    public bool Remove(T value);

    public bool Contains(T value);
    public void Clear();
    public int Count { get; }

    public void PreOrderTraversal(Action<T> action);
    public void PostOrderTraversal(Action<T> action);
    public void InOrderTraversal(Action<T> action);
    public IEnumerator<T> InOrderTraversal();

    public IEnumerator<T> GetEnumerator();
    System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator();
}
```

- **Modify Contents**
  - Add
  - Remove
- **Informational**
  - Contains
  - Count
- **Traversal/Enumeration**
  - In, Pre, Post Order
  - GetEnumerator

# AVL Tree Node

```
public class AVLTreeNode<TNode> : IComparable<TNode>
    where TNode : IComparable
{
    public AVLTreeNode(TNode value, AVLTreeNode<TNode> parent, AVLTree<TNode> tree);

    public AVLTreeNode<TNode> Left  { get; private set; }
    public AVLTreeNode<TNode> Right { get; private set; }
    public AVLTreeNode<TNode> Parent { get; private set; }
    public TNode Value { get; private set; }

    public int CompareTo(TNode other);

    // Balancing Methods
    internal void Balance();
    private void LeftRotation();
    private void RightRotation();
    private void LeftRightRotation();
    private void RightLeftRotation();

    // Support properties and methods
    private int MaxChildHeight(AVLTreeNode<TNode> node);
    private int LeftHeight { get; }
    private int RightHeight { get; }
    private TreeState State { get; }
    private int BalanceFactor { get; }
}
```

- **Binary Tree Properties**
  - Left, Right, Value
- **Balance Operations**
- **Support Properties**

# Binary Tree Properties

- **Left & Right**

```
public AVLTreeNode<TNode> Left
{
    get
    {
        return _left;
    }
    internal set
    {
        _left = value;
        if (_left != null)
        {
            _left.Parent = this;
        }
    }
}
```

- **Value**

```
public TNode Value { get; private set; }
```

- **Parent**

```
public AVLTreeNode<TNode> Parent { get; internal set; }
```

# AVL Node Height

- **Node Height**

```
private int LeftHeight
{
    get
    {
        return MaxChildHeight(Left);
    }
}

private int RightHeight
{
    get
    {
        return MaxChildHeight(Right);
    }
}

private int MaxChildHeight(AVLTreeNode<TNode> node)
{
    if (node != null)
    {
        return 1 + Math.Max(MaxChildHeight(node.Left), MaxChildHeight(node.Right));
    }

    return 0;
}
```

# Balance Factor and Left/Right Heavy

- ## Balance Factor

```
private int BalanceFactor
{
    get
    {
        return RightHeight - LeftHeight;
    }
}
```

- ## Heavy or Balanced?

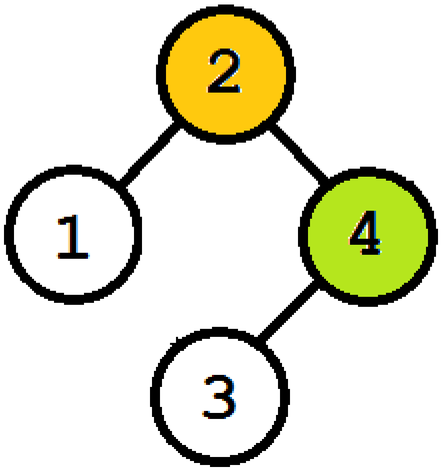
```
private TreeState State
{
    get
    {
        if (LeftHeight - RightHeight > 1)
        {
            return TreeState.LeftHeavy;
        }

        if (RightHeight - LeftHeight > 1)
        {
            return TreeState.RightHeavy;
        }

        return TreeState.Balanced;
    }
}
```

# Balancing Algorithms

- **Balancing is done using node rotation**
- **Rotation occurs at the insertion and deletion point (and parents)**
- **Rotation changes the physical structure of the tree within the constraints of the binary tree**
  - Smaller values on the left, larger and equal on the right
- **Rotation algorithms**
  - Right Rotation
  - Left Rotation
  - Right-Left Rotation
  - Left-Right Rotation

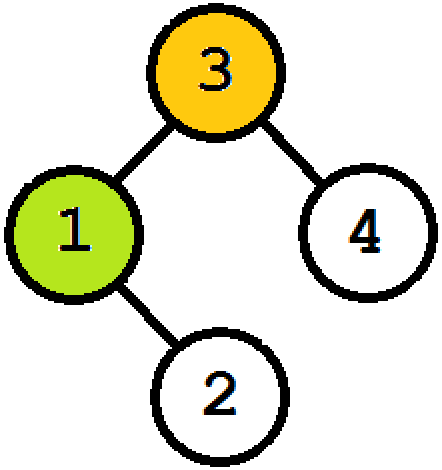
# Right Rotation

- **Algorithm to rotate a node to the right**
  - Left child becomes the new root
  - Right child of new root is assigned to left child of old root
  - Previous root becomes the new root's right child
- **Example rotation rooted at value 1**
  - Right height at node "4" is 0
  - Left height at node "4" is 2
  - "2" becomes new root
  - "4" becomes right child of "2"
  - "3" becomes left child of "4"



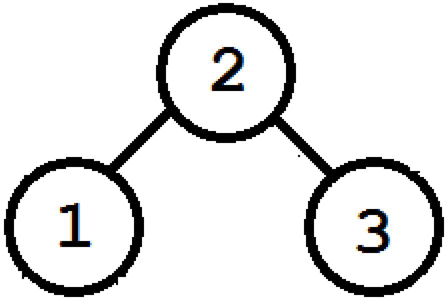
# Left Rotation

- **Algorithm to rotate a node to the left**
  - Right child becomes the new root
  - Left child of new root is assigned to right child of old root
  - Previous root becomes the new root's left child
- **Example rotation rooted at value 1**
  - Left height at node "1" is 0
  - Right height at node "1" is 2
  - "3" becomes new root
  - "2" becomes left child of "1"
  - "1" becomes left child of "3"



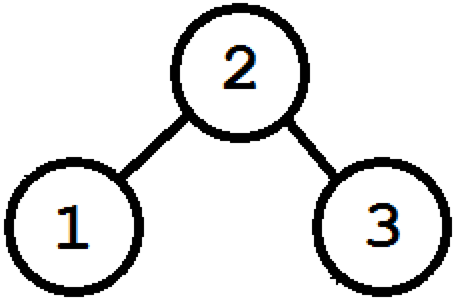
# Right-Left Rotation

- **Right rotation can leave a tree unbalanced**
- **Solution**
  - Left Rotate the left child
  - Right Rotate the updated tree
- **Example**
  - Perform left rotation at “1”
  - Perform a right rotation at “3”



# Left-Right Rotation

- **Left rotation can leave a tree unbalanced**
- **Solution**
  - Right Rotate the right child
  - Left Rotate the updated tree
- **Example**
  - Perform right rotation at “3”
  - Perform a left rotation at “1”

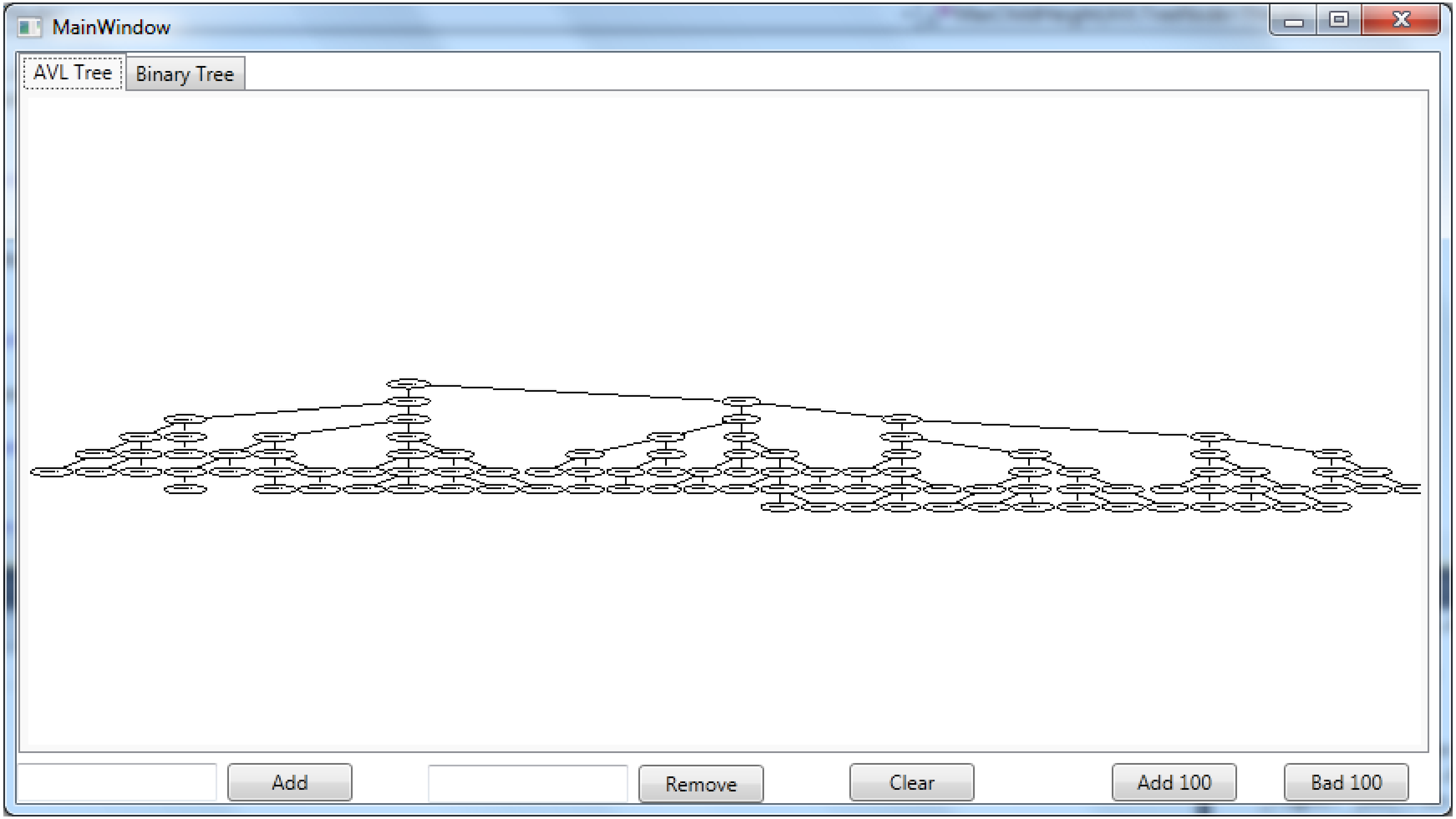


# Choosing The Proper Rotation

- **Right Heavy Tree**
  - If right child is left heavy
    - Left-Right Rotation
  - Else
    - Left Rotation
- **Left Heavy Tree**
  - If left child is right heavy
    - Right-Left Rotation
  - Else
    - Right Rotation

```
internal void Balance()
{
    if (State == TreeState.RightHeavy)
    {
        if (Right != null && Right.BalanceFactor < 0)
        {
            LeftRightRotation();
        }
        else
        {
            LeftRotation();
        }
    }
    else if (State == TreeState.LeftHeavy)
    {
        if (Left != null && Left.BalanceFactor > 0)
        {
            RightLeftRotation();
        }
        else
        {
            RightRotation();
        }
    }
}
```

# Sample Application



# Summary

- **Binary Tree Overview**
  - Unbalanced and Balanced
- **AVL Tree**
  - Base Tree Implementation
- **Balancing Algorithms**
  - Right Rotation
  - Left Rotation
  - Right-Left Rotation
  - Left-Right Rotation
- **Demo: Balanced Tree Viewer**

# References

- AVL Tree
  - http://en.wikipedia.org/wiki/AVL_tree
- Tree Rotation
  - http://en.wikipedia.org/wiki/Tree_rotation
- Binary Tree
  - http://www.pluralsight-training.net/microsoft/Courses/TableOfContents?courseName=ads-part1
  - http://en.wikipedia.org/wiki/Binary_tree
- GLEE Library (Microsoft Automated Graph Layout)
  - http://research.microsoft.com/en-us/projects/msagl/default.aspx