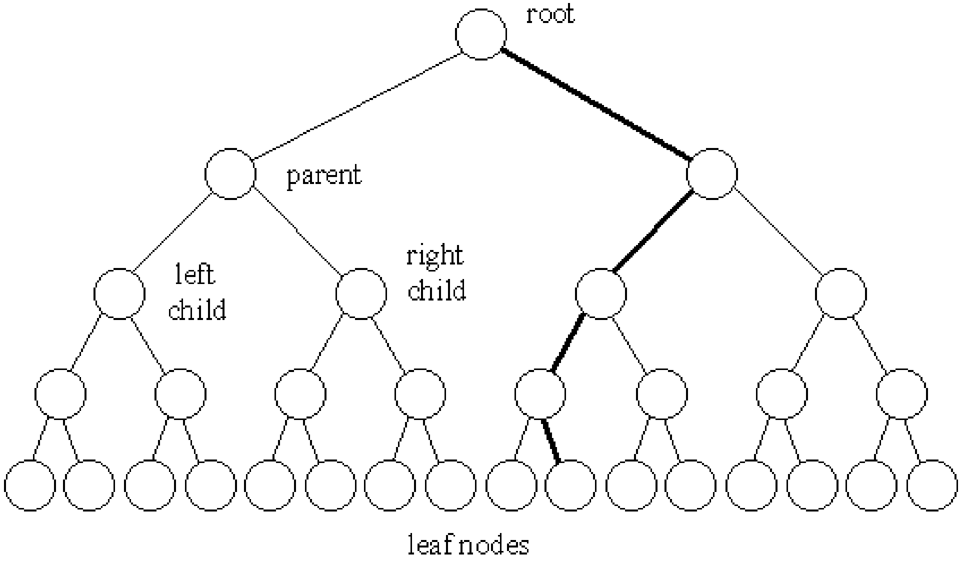
Traverse Binary Trees

SEW3



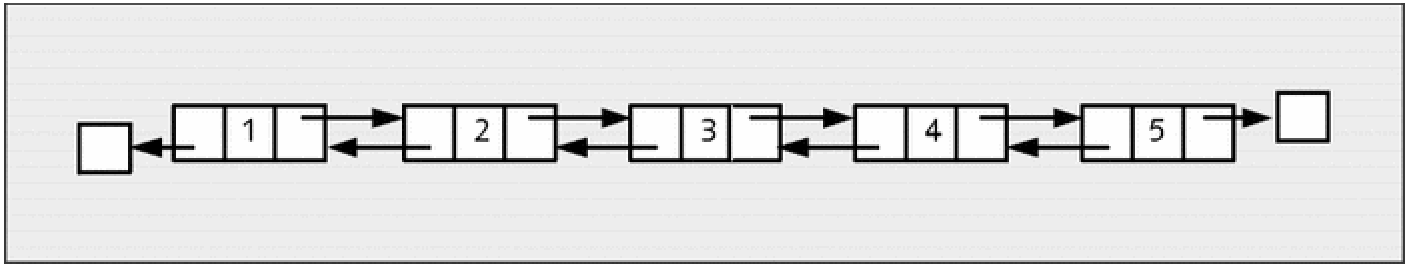
Overview

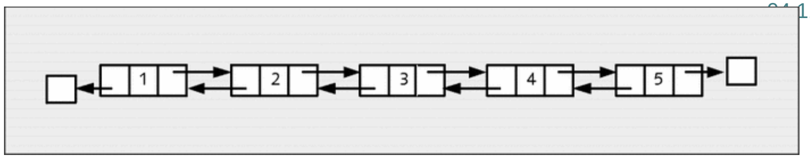
- Linked List vs Binary Trees
- Advantage of Binary Search Trees
- Definitions
- Traverse through Binary Trees

- Code to
  - create a Binary Tree Node
  - insert a Node
  - traverse through Trees

Binary Trees vs Linked Lists

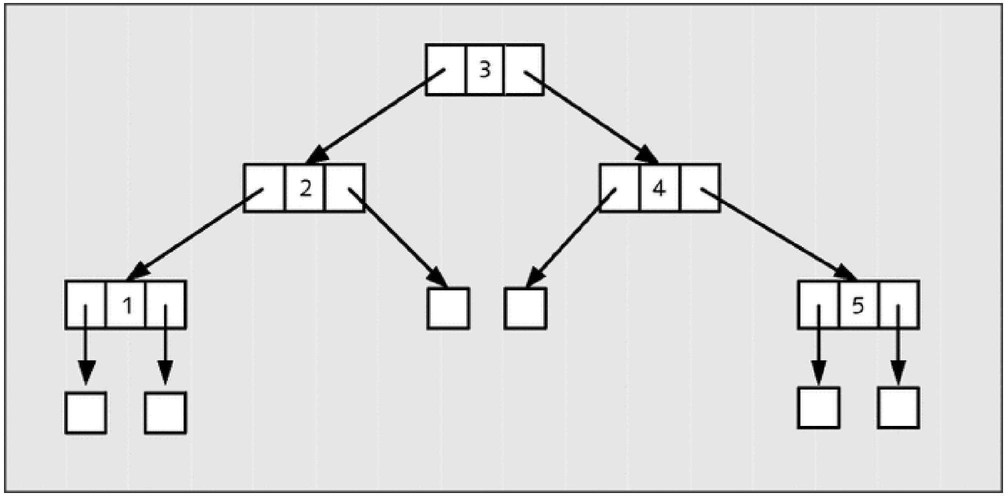
- Linked List
  - Head: Reference to the first element
  - Node has a
    - Value (Data)
    - Reference to the next node (Next)
    - Reference to the previous node (Previous)





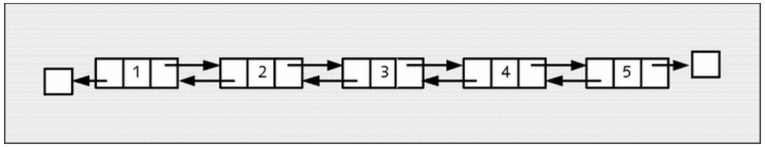
Binary Tree

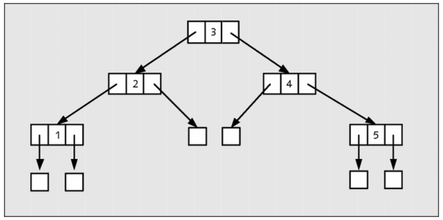
- Root: Reference to the first Node
  - Node has a
    - Value (Data)
    - Reference to a left Node (less than)
    - Reference to a right Node (greater than)



Advantage of binary trees

- Steps to get from the first element to element 5



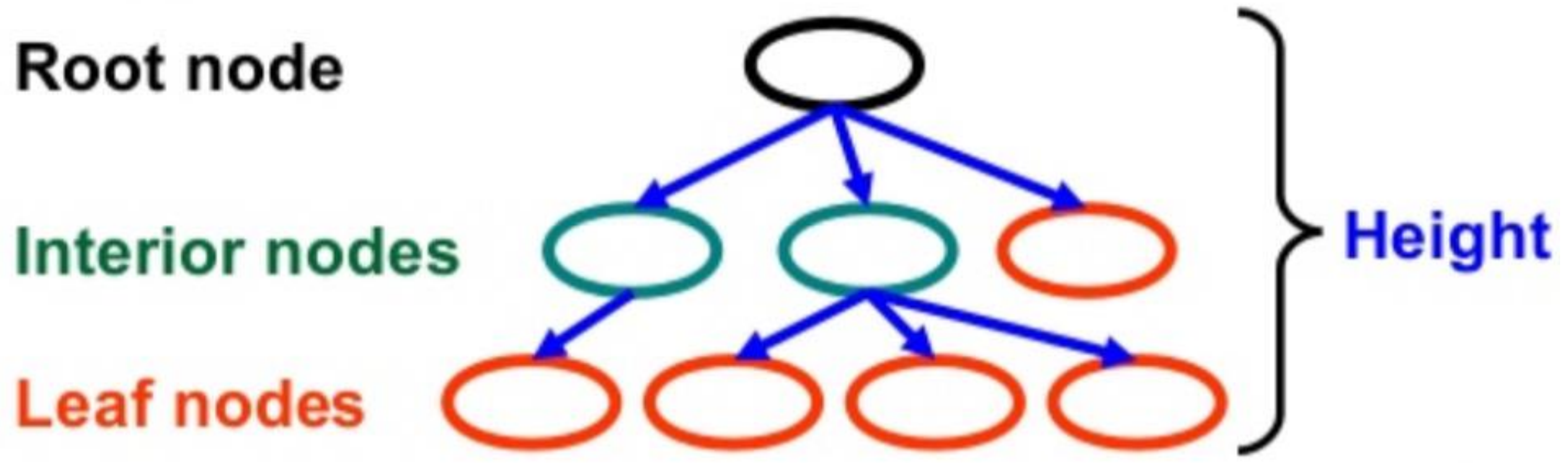


  - Linked List -> 5 steps
  - Binary Tree -> 3 steps

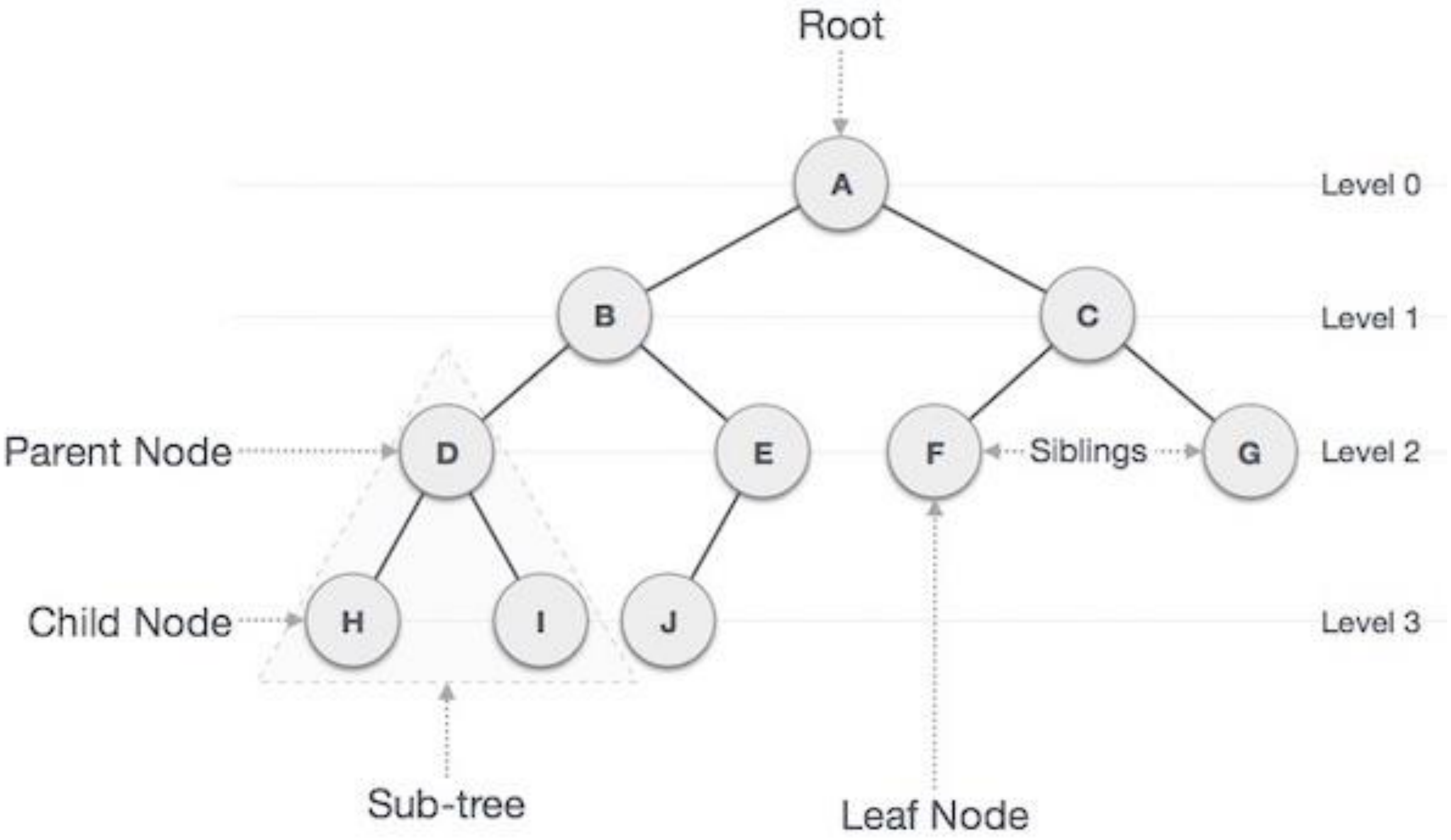
- Binary tree uses less steps to find an element
- Performance gets even better in huge data sets

Terminology

- Root – no parent
- Leaf – no child
- Interior – non leaf
- Height – distance from root to leaf



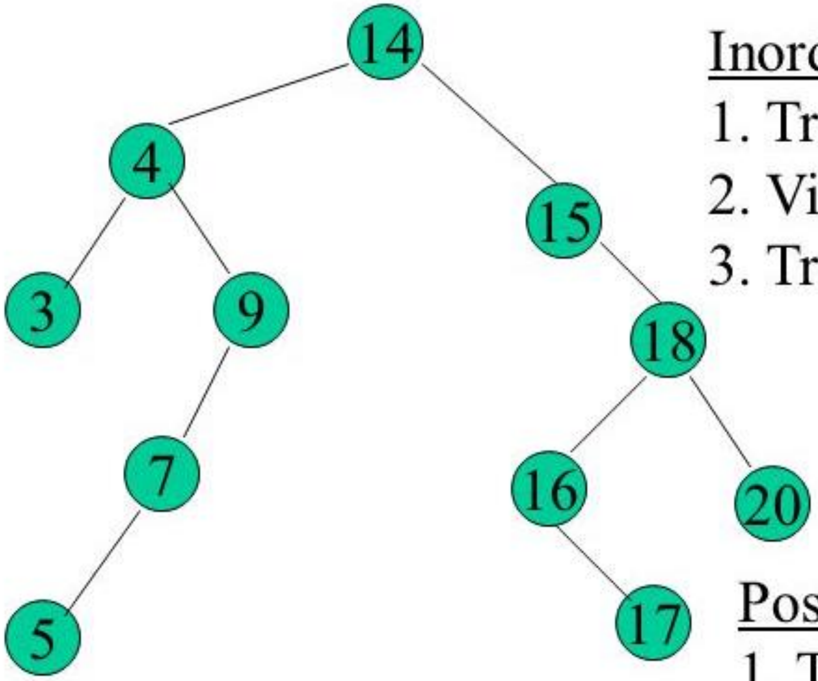
Terminology



Traverse a binary tree

Three methods:

- 1. preorder
- 2. inorder
- 3 postorder



Preorder

1. Visit the root
2. Traverse the left subtree in preorder
3. Traverse the right subtree in preorder

Inorder

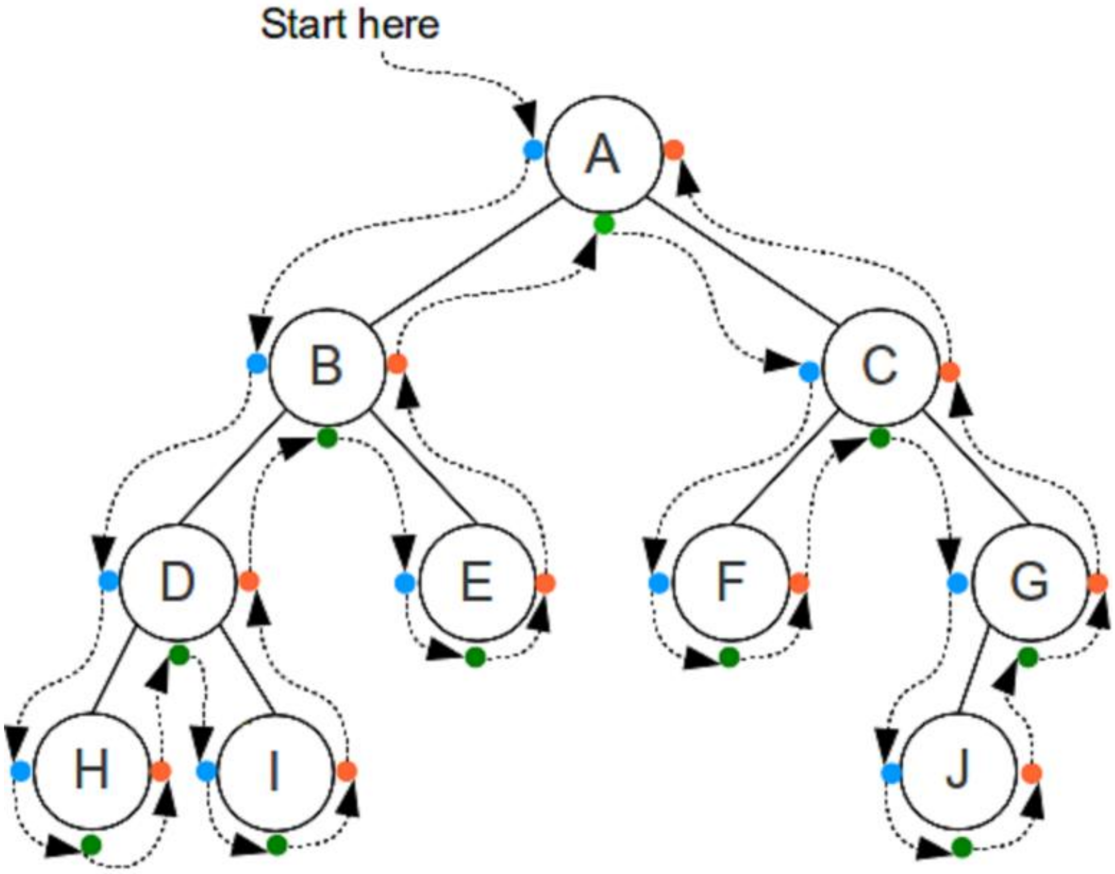
1. Traverse the left subtree in inorder
2. Visit the root
3. Traverse the right subtree in inorder

Postorder

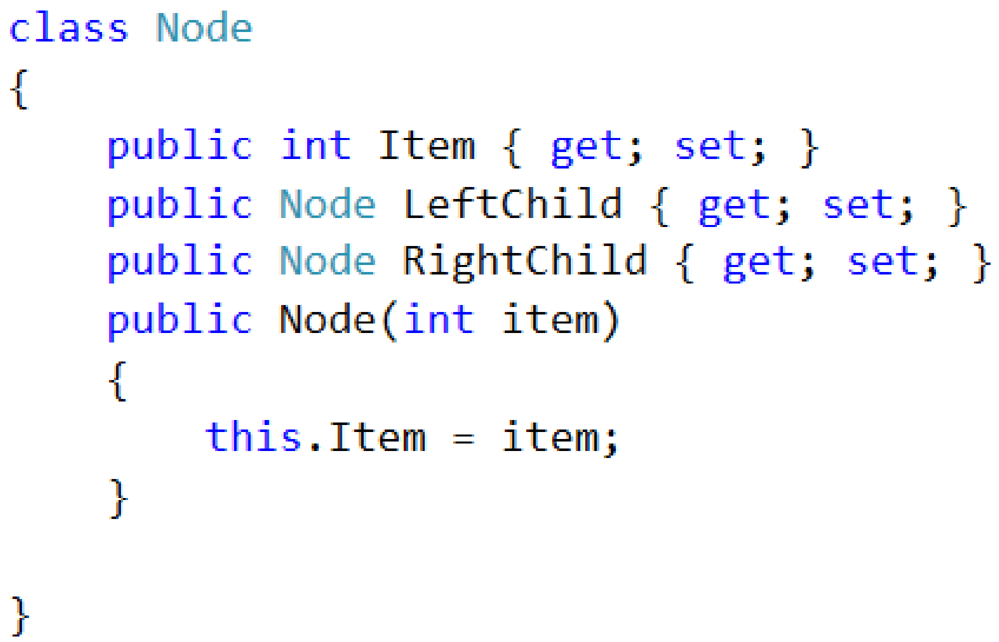
1. Traverse the left subtree in postorder
2. Traverse the right subtree in postorder
3. Visit the root

Traverse

- Inorder
  - HDIBEAFCJG
- Preorder
  - ABDHIECFGJ
- Postorder
  - HIDEBFJGCA

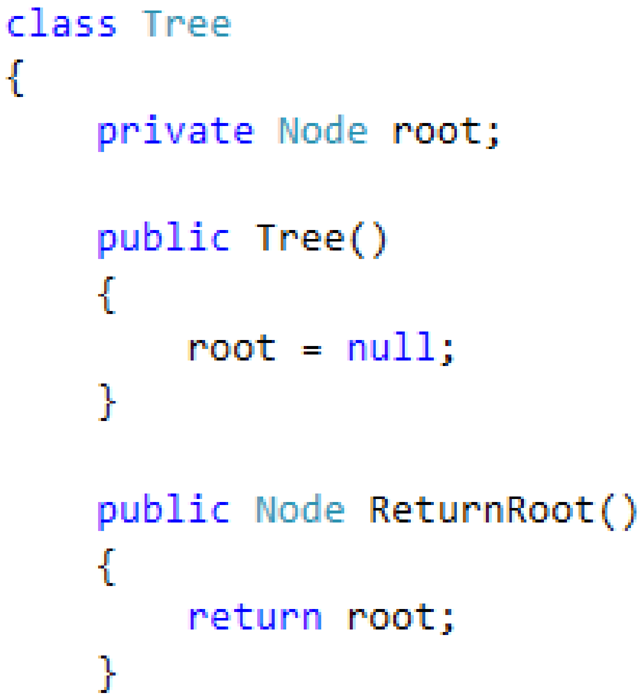


Node

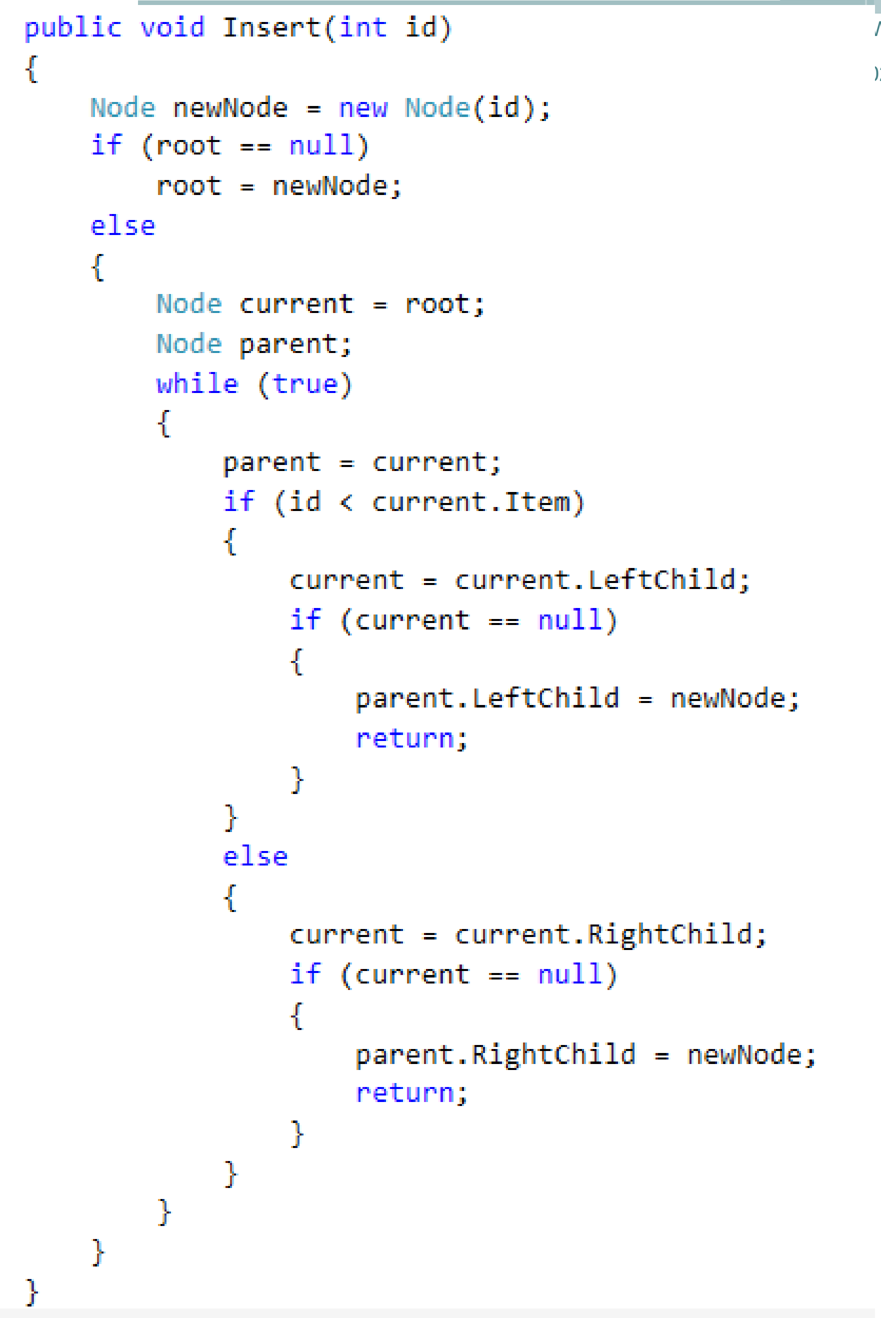


```
class Node
{
    public int Item { get; set; }
    public Node LeftChild { get; set; }
    public Node RightChild { get; set; }
    public Node(int item)
    {
        this.Item = item;
    }

}
```

Tree



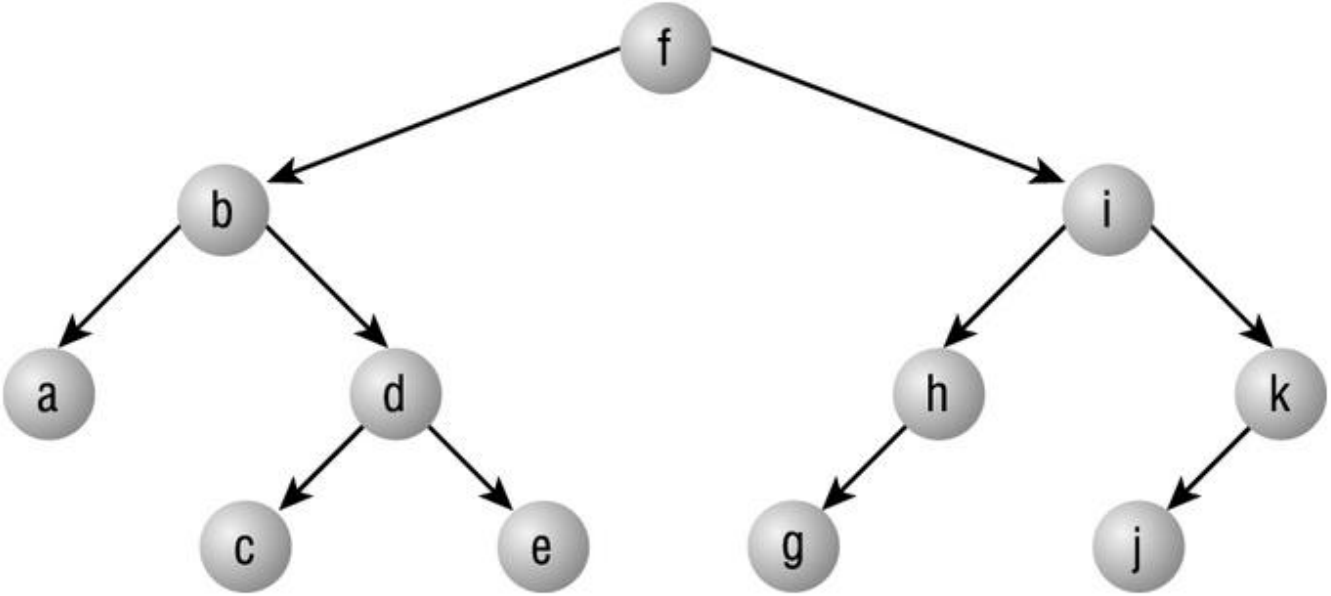
```
class Tree
{
    private Node root;

    public Tree()
    {
        root = null;
    }

    public Node ReturnRoot()
    {
        return root;
    }
```



```
public void Insert(int id)
{
    Node newNode = new Node(id);
    if (root == null)
        root = newNode;
    else
    {
        Node current = root;
        Node parent;
        while (true)
        {
            parent = current;
            if (id < current.Item)
            {
                current = current.LeftChild;
                if (current == null)
                {
                    parent.LeftChild = newNode;
                    return;
                }
            }
            else
            {
                current = current.RightChild;
                if (current == null)
                {
                    parent.RightChild = newNode;
                    return;
                }
            }
        }
    }
}
```

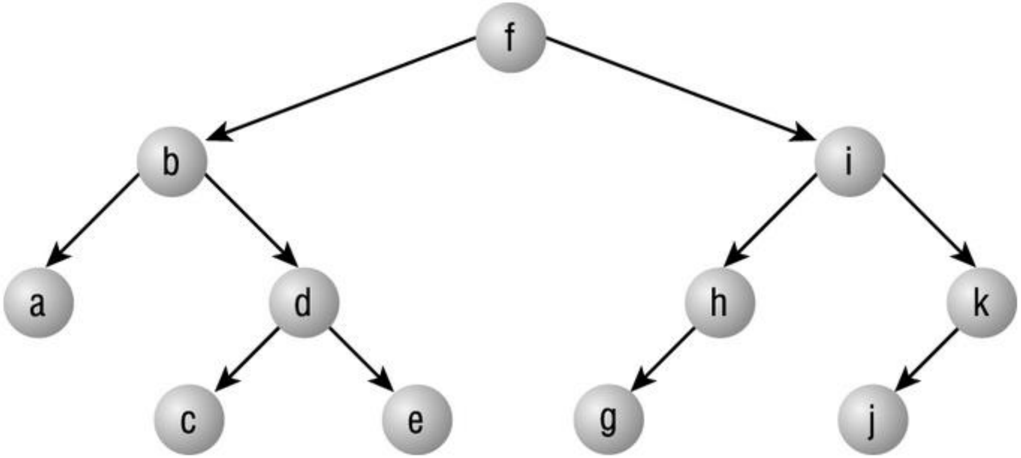
Traverse Trees

- Inorder
- Preorder
- Postorder



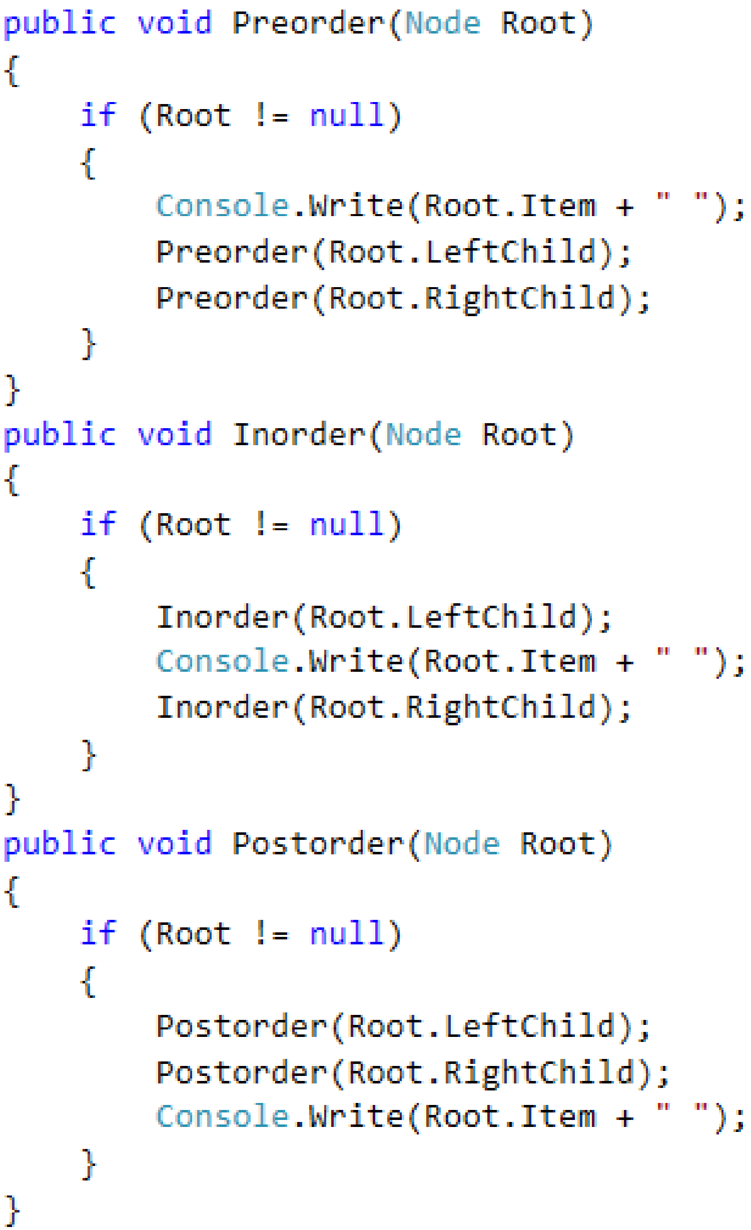
Traverse Trees

- Inorder Traversal:
  - Left Middle Right
  - ▫ a b c d e f g h i j k
- Preorder Traversal:
  - Middle Left Right
  - ▫ f b a d c e i h g k j
- Postorder
  - Left Right Middle
  - ▫ a c e d b g h j k i f



Trees

- Traverse Preorder
- Traverse Inorder
- Traverse Postoder



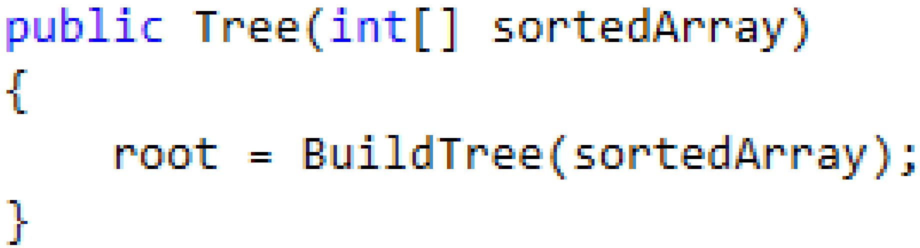
```
public void Preorder(Node Root)
{
    if (Root != null)
    {
        Console.Write(Root.Item + " ");
        Preorder(Root.LeftChild);
        Preorder(Root.RightChild);
    }
}
public void Inorder(Node Root)
{
    if (Root != null)
    {
        Inorder(Root.LeftChild);
        Console.Write(Root.Item + " ");
        Inorder(Root.RightChild);
    }
}
public void Postorder(Node Root)
{
    if (Root != null)
    {
        Postorder(Root.LeftChild);
        Postorder(Root.RightChild);
        Console.Write(Root.Item + " ");
    }
}
```

Create a Constructor

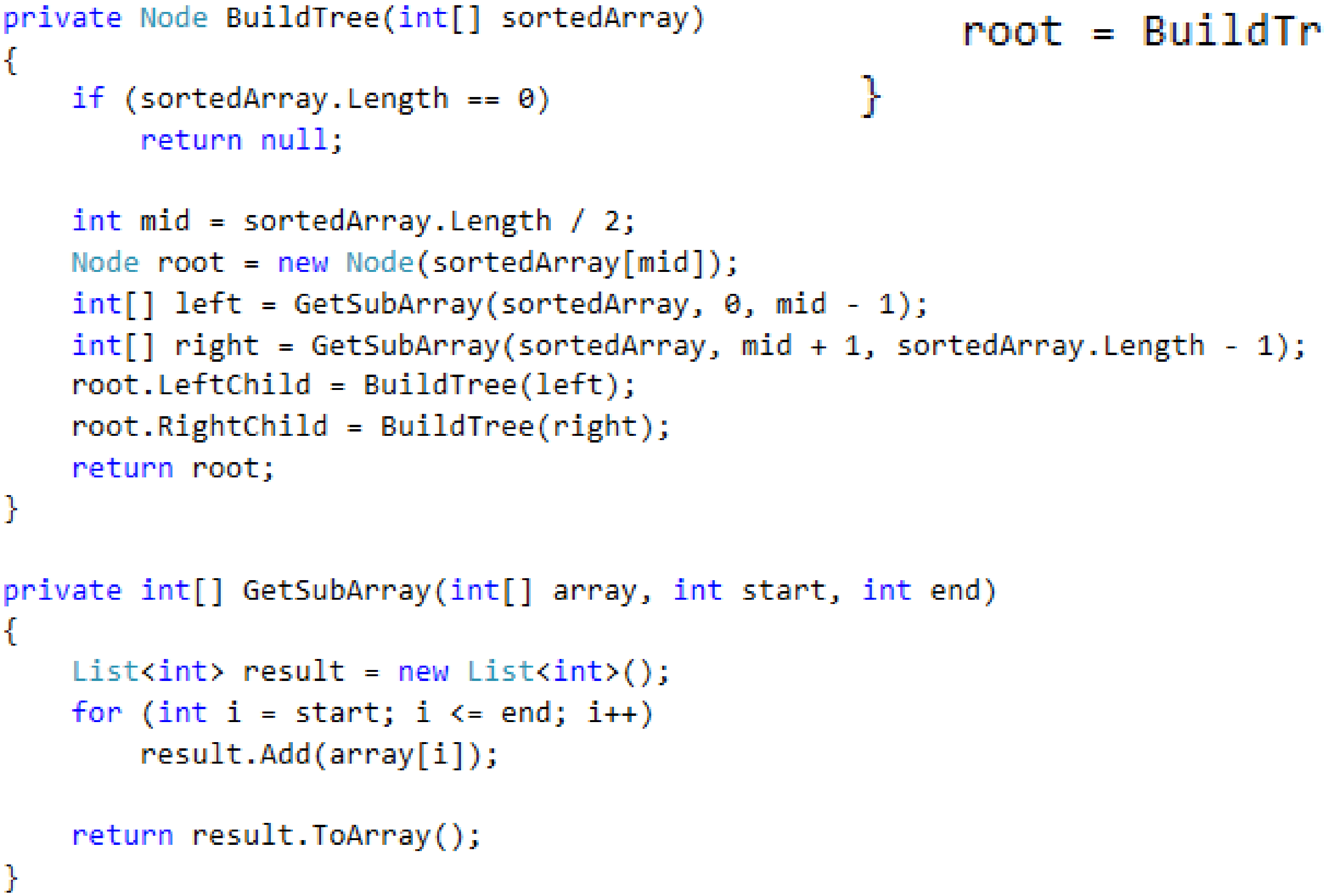
- Create a constructor for unsorted arrays
  - using a array to initialise the tree
  - use a unsorted array

- Create a constructor for sorted arrays
  - Insert the middle element as root element
  - Use then the „left middle“ and the „right middle“

Constructor for sorted arrays



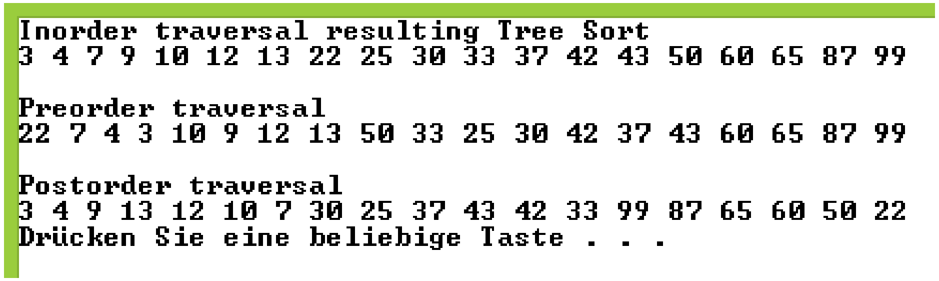
```
public Tree(int[] sortedArray)
{
    root = BuildTree(sortedArray);
}
```



```
private Node BuildTree(int[] sortedArray)
{
    if (sortedArray.Length == 0)
        return null;

    int mid = sortedArray.Length / 2;
    Node root = new Node(sortedArray[mid]);
    int[] left = GetSubArray(sortedArray, 0, mid - 1);
    int[] right = GetSubArray(sortedArray, mid + 1, sortedArray.Length - 1);
    root.LeftChild = BuildTree(left);
    root.RightChild = BuildTree(right);
    return root;
}

private int[] GetSubArray(int[] array, int start, int end)
{
    List<int> result = new List<int>();
    for (int i = start; i <= end; i++)
        result.Add(array[i]);

    return result.ToArray();
}
```

Teste den Quellcode in der Main

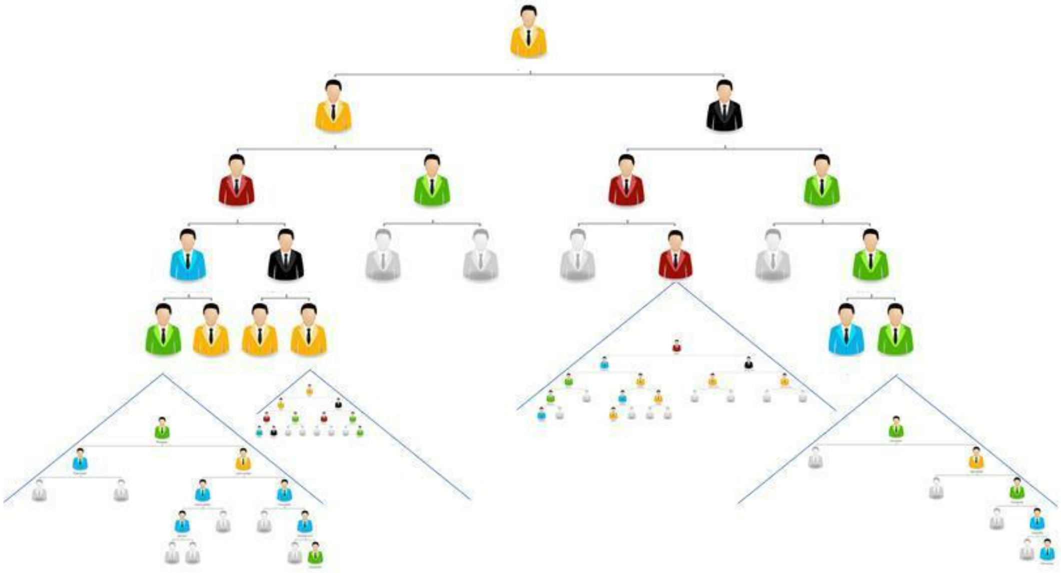
```
static void Main(string[] args)
{
    int[] arr = { 3, 4, 7, 10, 22, 33, 50, 60 };
    Tree theTree = new Tree(arr);

    theTree.Insert(42);
    theTree.Insert(25);
    theTree.Insert(65);
    theTree.Insert(12);
    theTree.Insert(37);
    theTree.Insert(13);
    theTree.Insert(30);
    theTree.Insert(43);
    theTree.Insert(87);
    theTree.Insert(99);
    theTree.Insert(9);

    Console.WriteLine("Inorder traversal resulting Tree Sort");
    theTree.Inorder(theTree.ReturnRoot());
    Console.WriteLine(" ");

    Console.WriteLine();
    Console.WriteLine("Preorder traversal");
    theTree.Preorder(theTree.ReturnRoot());
    Console.WriteLine(" ");

    Console.WriteLine();
    Console.WriteLine("Postorder traversal");
    theTree.Postorder(theTree.ReturnRoot());
    Console.WriteLine(" ");

}
```



```
Inorder traversal resulting Tree Sort
3 4 7 9 10 12 13 22 25 30 33 37 42 43 50 60 65 87 99

Preorder traversal
22 7 4 3 10 9 12 13 50 33 25 30 42 37 43 60 65 87 99

Postorder traversal
3 4 9 13 12 10 7 30 25 37 43 42 33 99 87 65 60 50 22
Drücken Sie eine beliebige Taste . . .
```

Exercise

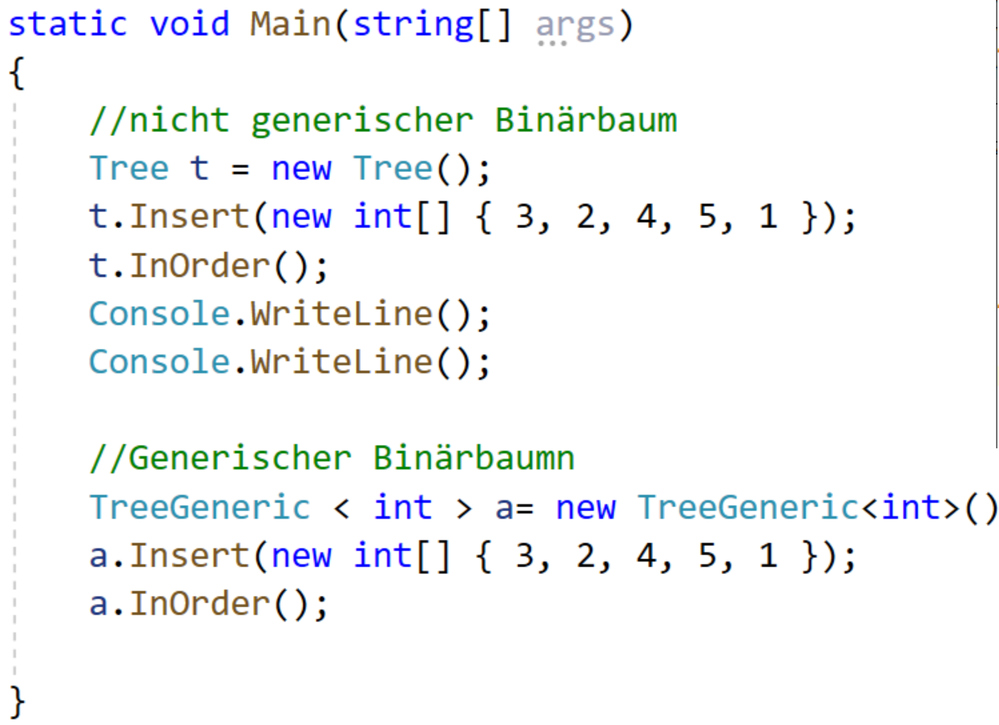
- Write a search method
  - iterative
  - recursive
- Write a Print Method
  - Inorder
  - Preorder
  - Postorder
- Write a Init Method with random values



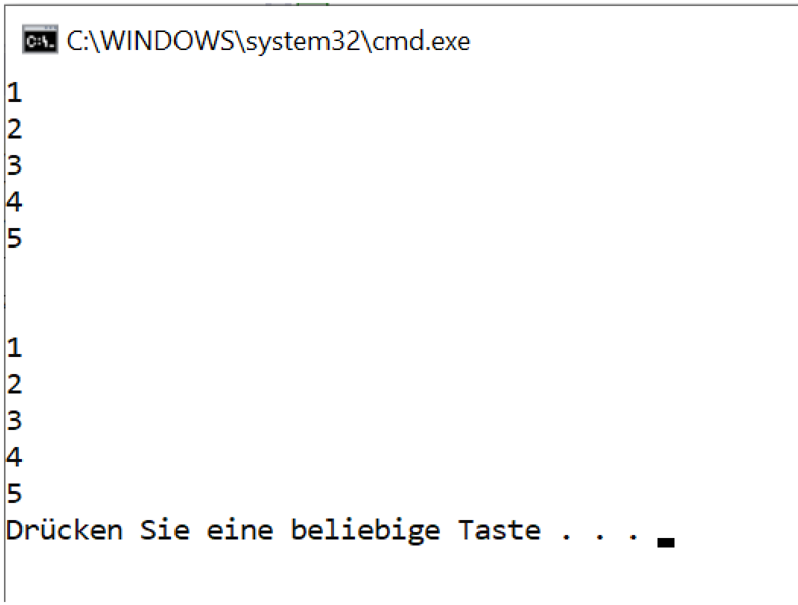
Generic Tree

Use the Tree with Int-Values – change it to a generic tree. Make sure, that you can compare items – using ICompareable

Main



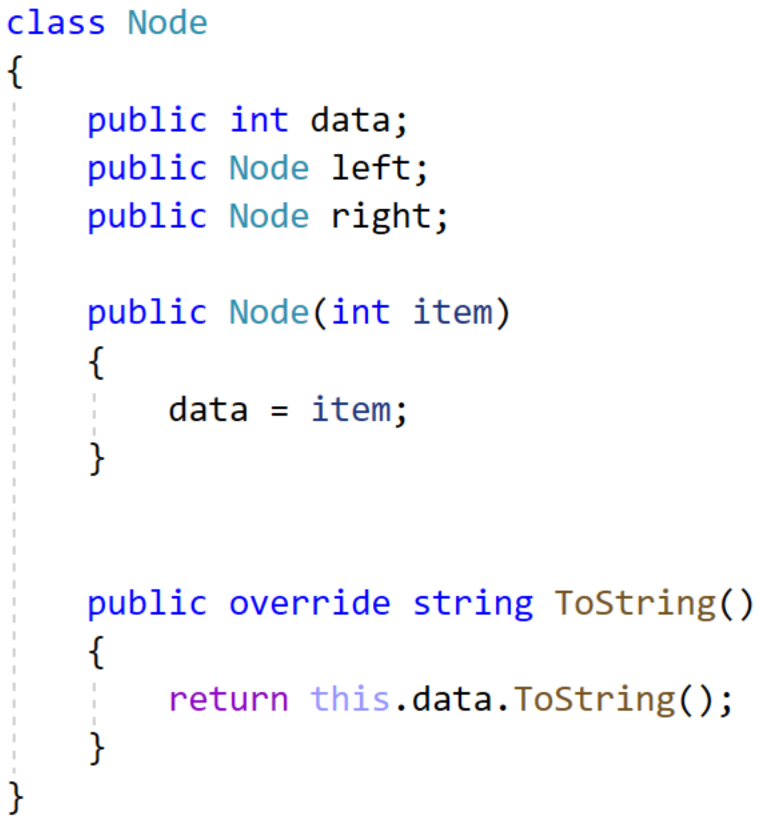
```
static void Main(string[] args)
{
    //nicht generischer Binärbaum
    Tree t = new Tree();
    t.Insert(new int[] { 3, 2, 4, 5, 1 });
    t.InOrder();
    Console.WriteLine();
    Console.WriteLine();

    //Generischer Binärbaumn
    TreeGeneric < int > a= new TreeGeneric<int>();
    a.Insert(new int[] { 3, 2, 4, 5, 1 });
    a.InOrder();

}
```



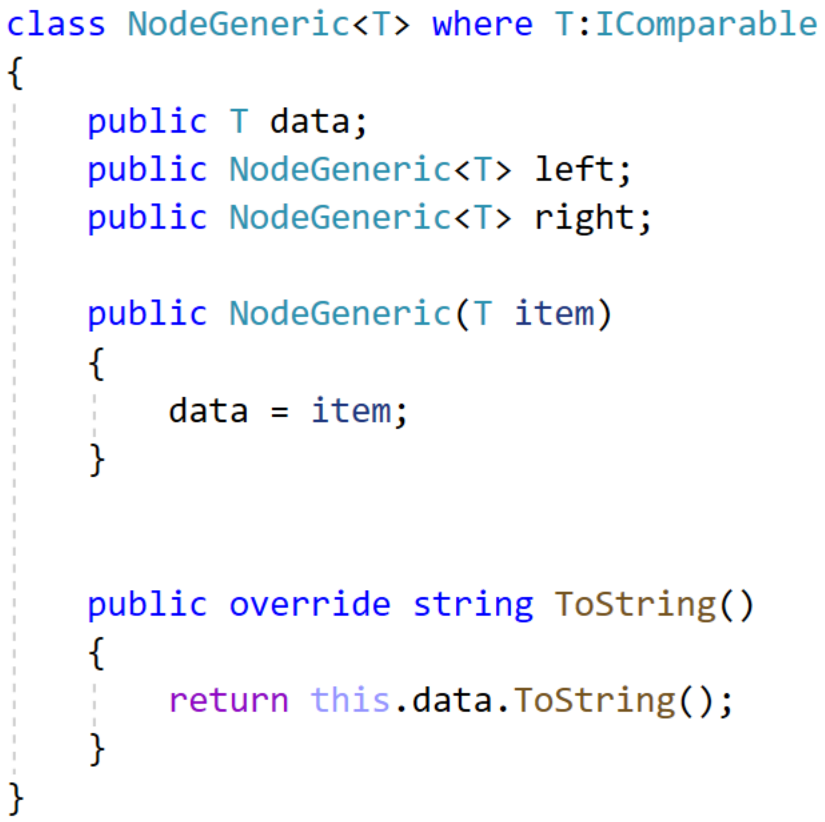
C:\WINDOWS\system32\cmd.exe

```
1
2
3
4
5


1
2
3
4
5
Drücken Sie eine beliebige Taste . . .
```

Node vs Generic Node

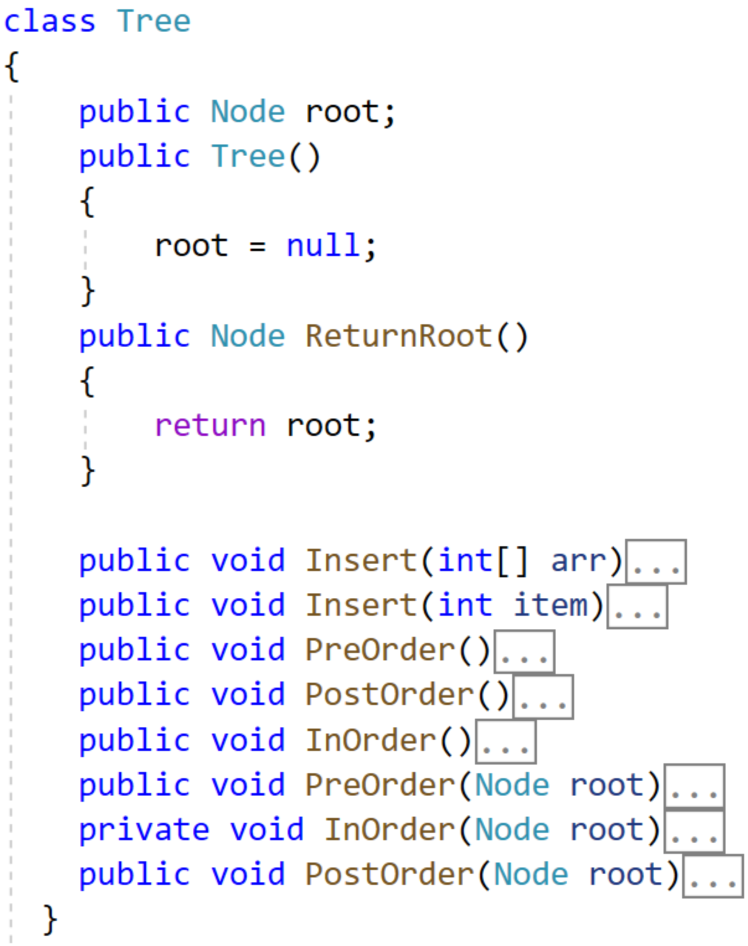


```
class Node
{
    public int data;
    public Node left;
    public Node right;

    public Node(int item)
    {
        data = item;
    }


    public override string ToString()
    {
        return this.data.ToString();
    }
}
```

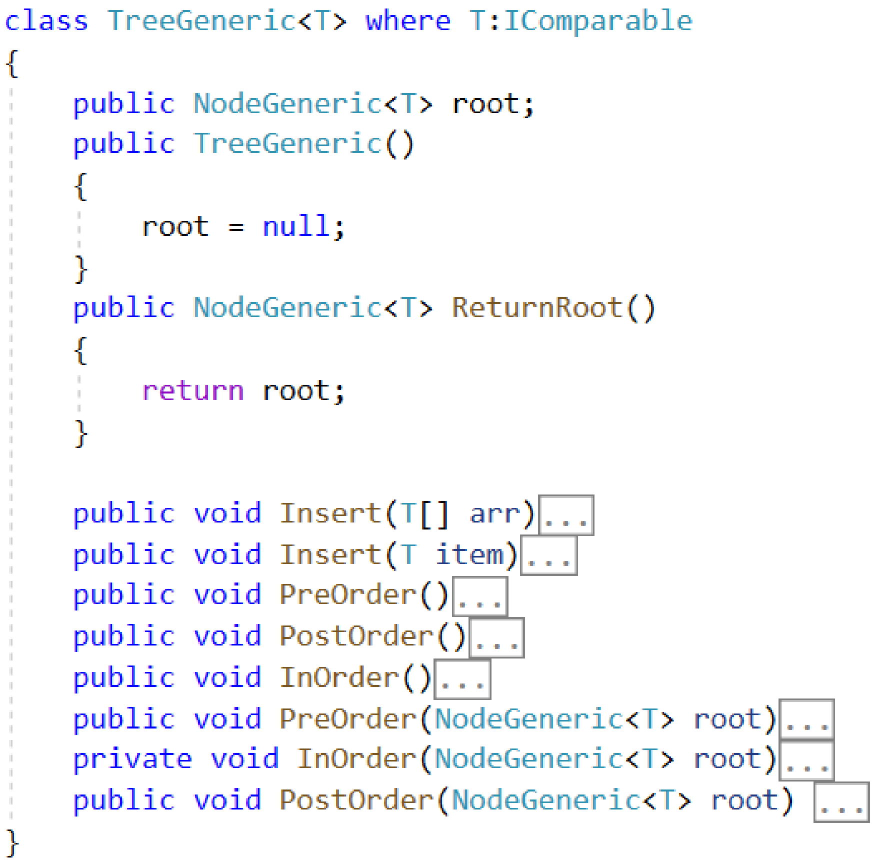


```
class NodeGeneric<T> where T:IComparable
{
    public T data;
    public NodeGeneric<T> left;
    public NodeGeneric<T> right;

    public NodeGeneric(T item)
    {
        data = item;
    }


    public override string ToString()
    {
        return this.data.ToString();
    }
}
```

Tree vs Generic Tree



```
class Tree
{
    public Node root;
    public Tree()
    {
        root = null;
    }
    public Node ReturnRoot()
    {
        return root;
    }

    public void Insert(int[] arr)...
    public void Insert(int item)...
    public void PreOrder()...
    public void PostOrder()...
    public void InOrder()...
    public void PreOrder(Node root)...
    private void InOrder(Node root)...
    public void PostOrder(Node root)...
}
```



```
class TreeGeneric<T> where T:IComparable
{
    public NodeGeneric<T> root;
    public TreeGeneric()
    {
        root = null;
    }
    public NodeGeneric<T> ReturnRoot()
    {
        return root;
    }

    public void Insert(T[] arr)...
    public void Insert(T item)...
    public void PreOrder()...
    public void PostOrder()...
    public void InOrder()...
    public void PreOrder(NodeGeneric<T> root)...
    private void InOrder(NodeGeneric<T> root)...
    public void PostOrder(NodeGeneric<T> root) ...
}
```