Chapter 15 -2 Trees – Properties and Traversals of a Binary Tree

Properties of a Binary Tree

- The maximum height of an n-node binary tree is n
- The minimum height of an n-node binary tree is log (n+1)

Example) Full binary tree

Properties of a Binary Tree

If $n = the\ total\ number\ of\ nodes\ in\ a\ binary\ tree\ T$,

$n_E = the\ total\ number\ of\ external\ nodes\ in\ a\ binary\ tree\ T$,

$n_I = the\ total\ number\ of\ internal\ nodes\ in\ a\ binary\ tree\ T$, and

$$h = the\ height\ of\ a\ binary\ tree\ T$$

1. $h \leq n \leq 2^h - 1$
2. $1 \leq n_E \leq 2^{h-1}$
3. $h - 1 \leq n_I \leq 2^{h-1} - 1$
4. $\log(n+1) \leq h \leq n$

The ADT Binary Tree

- Operations of ADT binary tree
  - Add, remove
  - Set, retrieve data
  - Test for empty
  - Traversal operations that visit every node
- Traversal can visit nodes in several different orders

Traversals of a Binary Tree

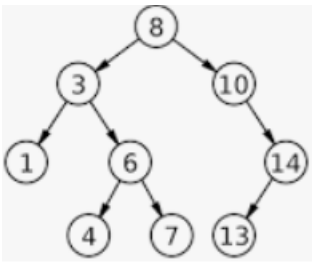
- Pseudocode for general form of a recursive traversal algorithm

```
if (T is not empty)
{
   Display the data in T's root
   Traverse T's left subtree
   Traverse T's right subtree
}
```

- Options for when to visit the root
  - Preorder: before it traverses both subtrees
  - Inorder: after it traverses left subtree, before it traverses right subtree
  - Postorder: after it traverses both subtrees
- Note traversal is O(*n*)

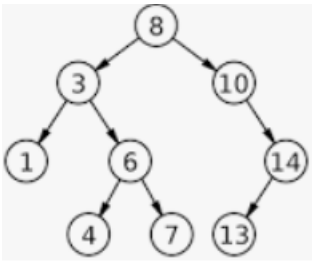
Preorder Traversal Algorithm

```
// Traverses the given binary tree in preorder.
// Assumes that "visit a node" means to process the node's data item.
preorder(binTree: BinaryTree): void
{
   if (binTree is not empty)
   {
      Visit the root of binTree
      preorder(Left subtree of binTree's root)
      preorder(Right subtree of binTree's root)
   }
}
```



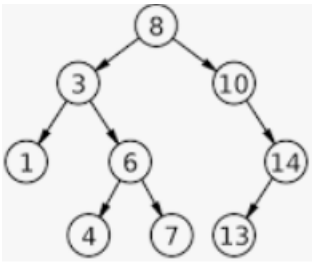
Inorder Traversal Algorithm

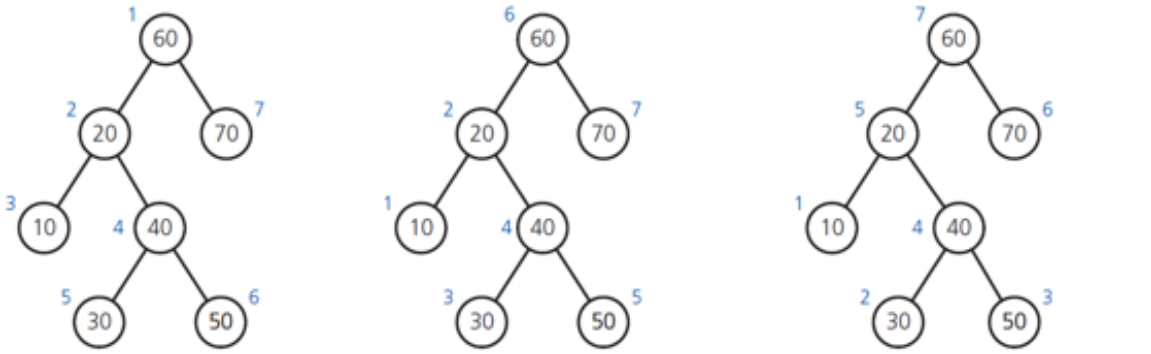
```
// Traverses the given binary tree in inorder.
// Assumes that "visit a node" means to process the node's data item.
inorder(binTree: BinaryTree): void
{
   if (binTree is not empty)
   {
      inorder(Left subtree of binTree's root)
      Visit the root of binTree
      inorder(Right subtree of binTree's root)
   }
}
```



Postorder Traversal Algorithm

```
// Traverses the given binary tree in postorder.
// Assumes that "visit a node" means to process the node's data item.
postorder(binTree: BinaryTree): void
{
   if (binTree is not empty)
   {
      postorder(Left subtree of binTree's root)
      postorder(Right subtree of binTree's root)
      Visit the root of binTree
   }
}
```





(a) Preorder: 60, 20, 10, 40, 30, 50, 70 (b) Inorder: 10, 20, 30, 40, 50, 60, 70 (c) Postorder: 10, 30, 50, 40, 20, 70, 60

(Numbers beside nodes indicate traversal order.)

Quiz