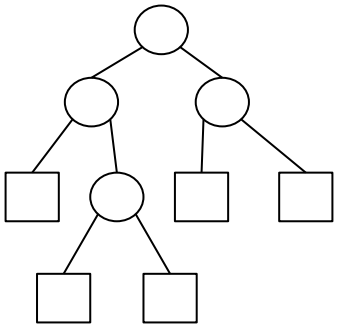
Trees (General):



Uses:

-
-
-
-

Terminology:

Tree:

Root:

Path:

Parent:

Child:

Ascendants:

Descendants:

Node:

Internal Nodes:

Leaves:

Subtree:

Level:

Depth:

Height:

Complete Binary Trees:

Def:

Level	Nodes per level	Sum all nodes

_____ leaves

_____ internal nodes

Height =

Levels =

Drawing:

Ex:

Complete Binary Trees:

Def:

Drawing:

_____ < leaves < _____

_____ internal nodes

Height =

Levels =

Ex:

Full (Binary) Tree:

Def:

Drawing:

If contains X internal nodes:

_____ external nodes

_____ edges/links

_____ total nodes

_____ < height < _____

Ex:

Binary Trees:

Traversal:

```
typedef struct TreeNode * TreeNodePT;
struct TreeNode {
   int data;
   TreeNodePT left;
   TreeNodePT right;
};
```

Depth-First Order Traversal using Recursion:

```
Preorder(TreeNodePT h) {



}
```

For a tree with N nodes:

TC:

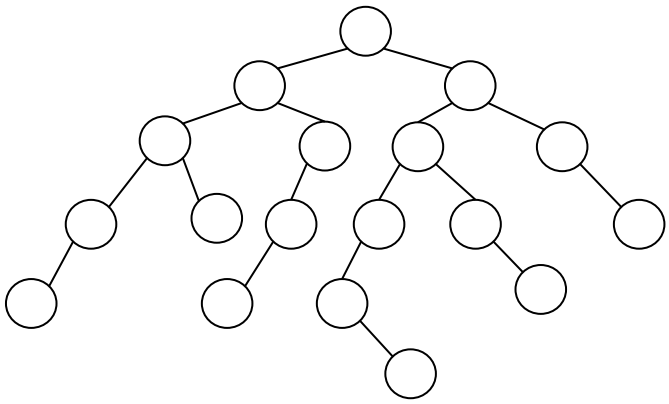
SC:

Inorder(TreeNodePT h) {

}

Postorder(TreeNodePT h) {

}



Preorder: _____

Inorder: _____

PostOrder: _____

Level-Order Traversal:

BreadthFirstTraversal(TreeNodePT h) {

}

Count number of Nodes in :

int count(TreeNodePT h) {

}

Height of Tree:

int height(TreeNodePT h) {

}

Why can we not assume the height of any binary tree?

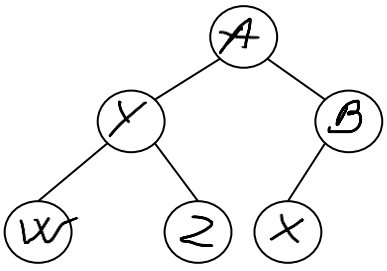
Best Case for Binary Tree:

Worst Case for Binary Tree:

<u>Binary Search Tree:</u>

General Search Tree:

Binary Search Tree:



A _____ X _____ B

Y _____ A _____ B

W _____ Y _____ A

Y _____ Z _____ A

Valid Search Path:

Searching for _____

Given Path: _____

Valid? If not, why?

Searching for _____

Given Path: _____

Valid? If not, why?

Searching for _____

Given Path: _____

Valid? If not, why?

Search:

TreeNodePT search(TreeNodePT tree, int data) {

}

Naive Insertion:

```
TreeNodePT new_tree_node(int data_in) {
   TreeNodePT ndp = malloc(sizeof(struct TreeNode));
   ndp->data = data_in;
   ndp->left = NULL;
   ndp->right = NULL;
   return ndp;
}

TreeNodePT insert(TreeNodePT h, int data) {

}
```

Deleting a Node:

Performance of BST:

How does order of insertion affect performance of searching and future insertions? Why do we want to keep the tree balanced?

How do we randomize the order of insertion?

How should we handle duplicates to balance the tree?

Time Complexity for a Tree with N Nodes:

Finding Min: Leftmost Node (from the root keep going left):

O(_____)

Finding Max: Rightmost Node (from the root keep going right):

O(_____)

Printing in order:

Increasing: Left, Root, Right (inorder traversal)

Decreasing: Right, Root, Left

O(_____) Can we give Theta? Θ(_____)

Finding Successor of Node X with Key K (go right):

O(_____)

Finding Predecessor of Node X with Key K (go left):

O(_____)

Searching for a Value (and not found):

O(_____)

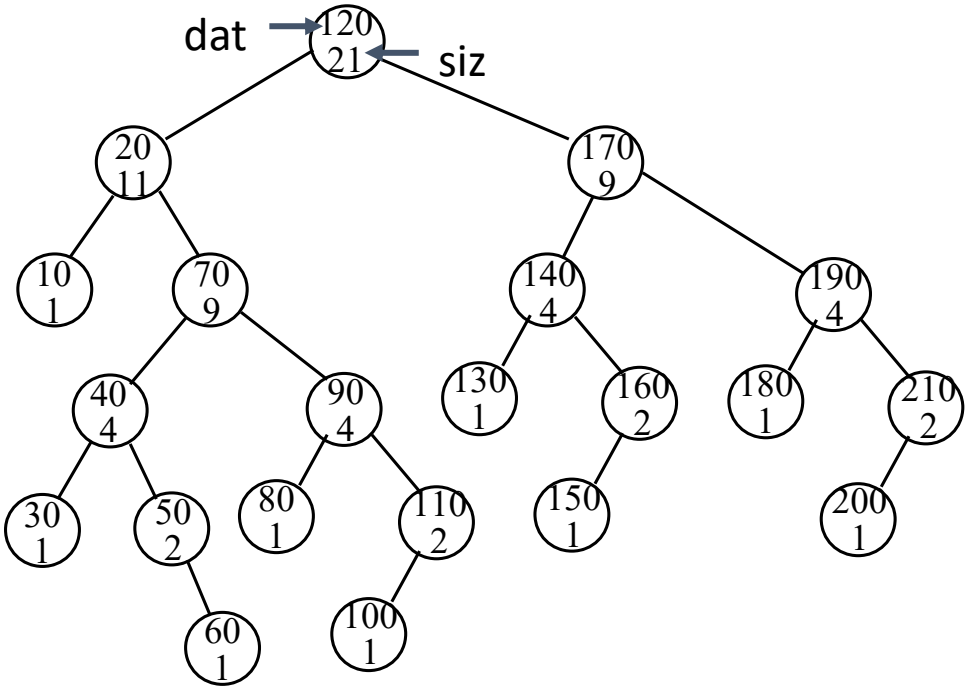
Build the Tree via N Repeated Insertions:

O(_____) Best: Θ(_____) Worst: Θ(_____)

Deletion of a Node:

O(_____) Best: Θ(_____) Worst: Θ(_____)

How about space complexity?

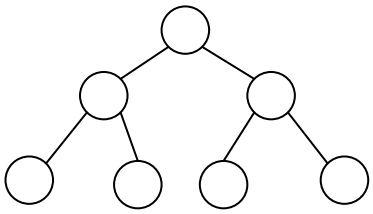

dat
120
21
siz
20
11
170
9
10
1
70
9
140
4
190
4
40
4
90
4
130
1
160
2
180
1
210
2
30
1
50
2
80
1
110
2
150
1
200
1
60
1
100
1

Rotation:

Purpose:

Implementation:

Original:



Left rotation:

Code:

Drawing:

Right rotation:

Code:

Drawing: