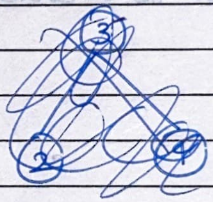
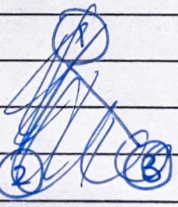
lecture 5
Binary Search Tree

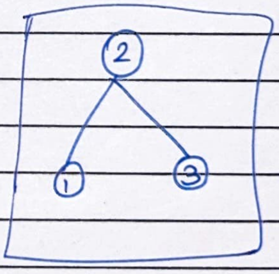
- Each node has a key, which determines the node's position in the tree.

- We generally work of key, Address

Let x be a node in BST,
If y is in LEFT subtree of x, $y.key \leq x.key$ and vice versa.





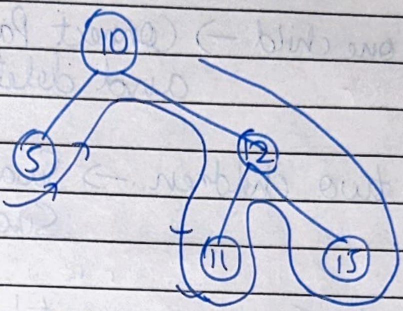


Total Possible BSTs $= \boxed{\frac{1}{n+1}\binom{2n}{n}}$

In C, BST is a struct with int key (value) and node *left and *right (pointers to next nodes)

Search → O(n)
Insert → O(n)
Height → N in worst case

Inorder Traversal



Output = 5, 10, 11, 12, 15
Flag is at the bottom

Post order Traversal

Output = 5, 11, 15, 12, 10

Pre - Order Traversal

Output - 10, 5, 12, 11, 15

Deletion of a Node

Case 1: It is a leaf node → Delete and Parent points to Null

Case 2: It has one child → Connect Parent to grandchild and delete

Case 3: It has two children → Swap with Inorder Successor, Delete then

Note : Inorder Successor cannot have LEFT child. It is paradoxical.

nt points to

grand chid

Inorder
Delete then

child.

Balanced Binary S Tree (Doubts)

If |height of LEFT subtree - RIGHT subtree| > 1

↓

Imbalanced BST

Note: Height is from leaf node

Solution: Rotate the BST to the RIGHT or LEFT

$f(h) = 1 + f(h-1) + f(h-2)$, h = height of tree

Height = $O(\log n)$

~~Code~~

~~int** dbl;~~

Arrays in C (Under-The-Hood)

Eg ~~int arr~~

Eg. int arr[] = {5, 6, 8};

Here, arr is the POINTER to the first location but arr[i] will get the address of the value

☆ Another Way of Defining Arrays in C