

Example #7: Invalid argument exception with stoi

56
5p-24
Sat.

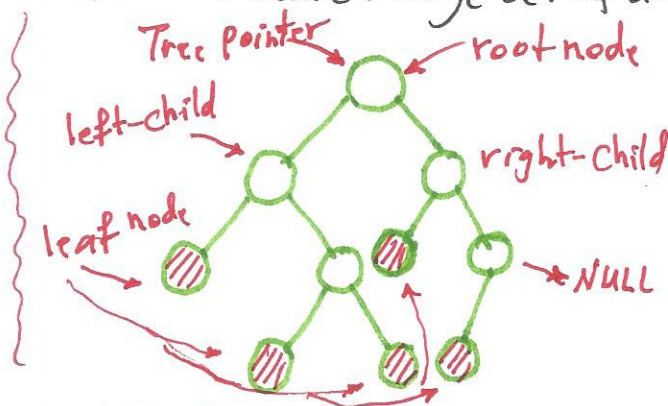
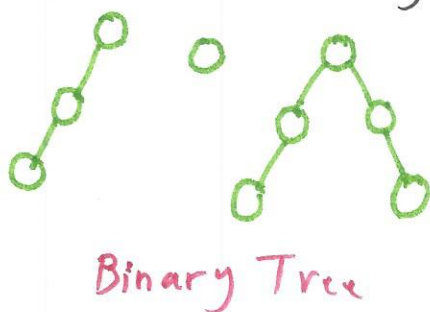
```
try { String str = "123abc";  
      int value = stoi(str); // Converting invalid string to integer  
    }  
    catch (const invalid_argument &err)  
    { cout << "Exception Caught: " << err.what() << endl;
```

Week 13, Sat:

Binary Tree: Each node can have at most \geq children.

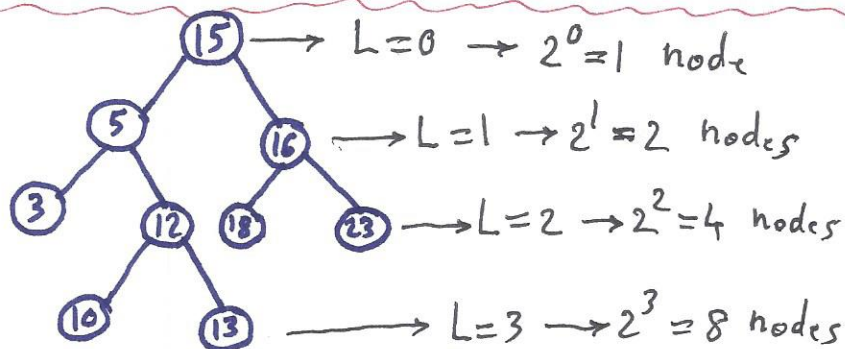
at most 2 $\equiv 0, 1, 2$

It is a non-linear data structure. Array and linked list are linear data structures. We use binary tree to store and search large set of data.

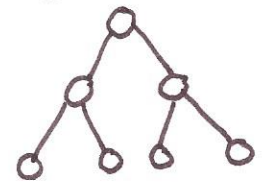


- A node with no children is called **leaf** node. Both left and right child are NULL.
- **Strict Binary Tree (or proper):** Each node can have 2 or 0 children. We have different levels. Max. # of nodes at level i is 2^i .
- Depth of a given node is the level it is on.
- Height of the trees: Max. depth of the tree is height of the tree.
- **Perfect Binary Tree:** All levels are filled.

Example for strict
Binary Tree →



- The height of a perfect binary tree is the length of the longest path between root and any of the leaf nodes or number of edges in the longest path from root to any of the leaf nodes.




- Max # of nodes in a tree with height h : $2^{h+1} - 1$

$$\text{Max \# of nodes} = 2^0 + 2^1 + 2^2 + \dots + 2^h$$

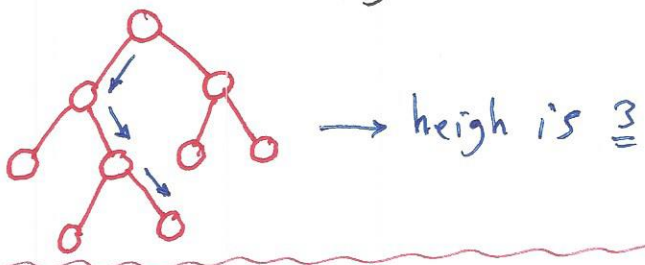
$$S = \frac{a(r^n - 1)}{r - 1} = \frac{2^0(2^{h+1} - 1)}{2 - 1} = 2^{h+1} - 1$$

ex #1, Max # of nodes with $h=3 \rightarrow 2^0 + 2^1 + 2^2 + 2^3 = 15$ or $2^{3+1} - 1 = 15$

One node:  \rightarrow height = 0, $h=0$, $2^{0+1} - 1 = 2 - 1 = 1$ node

no node: empty tree \rightarrow height = -1 $\rightarrow h = -1 \rightarrow 2^{-1+1} - 1 = 1 - 1 = 0$ node

example:



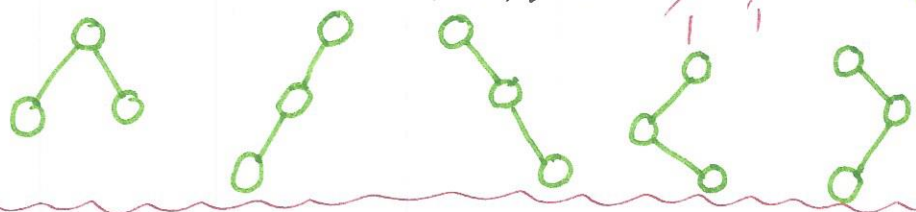
- Balanced Binary Tree: Difference between height of left and right subtree for every node is not more than one.

- Catalan Number: The # of distinct binary tree with n nodes is given by the Catalan number:

$$f(n) = \frac{(2n)!}{(n+1)! n!}$$

Example: $n=3$, $f(3) = \frac{6!}{(3+1)! 3!} = \frac{4! \times 5 \times 6}{4! \times 6} = 5$

\rightarrow 5 distinct binary tree

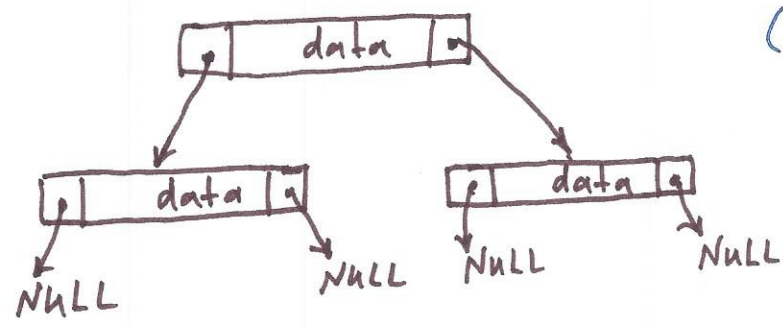


An application of Binary Tree: Binary Expression Tree

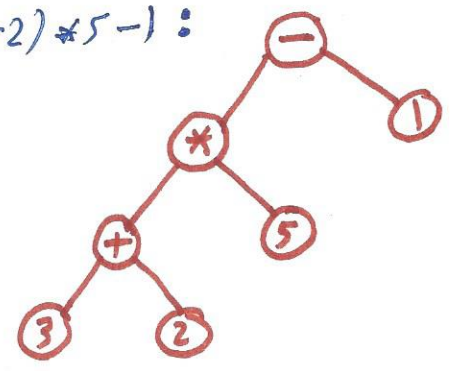
$$(3 + 2) * 5 - 1$$

1- Arithmetic expressions can be represented by Binary Trees.

2- Data Structure



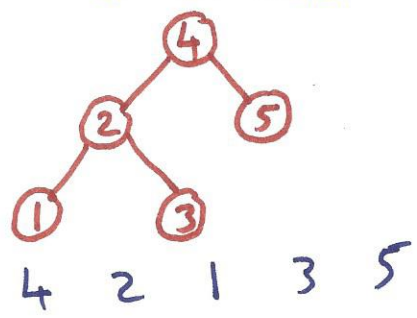
$(3+2) * 5 - 1 :$



- Binary Tree Traversal: 1- pre-order 2- In-Order 3- post-order

1- pre-order: Root-Left-Right

- Visit the root
- Traverse left subtree
- " right "



2- In-Order: Left-Root-Right

- Traverse left subtree
- Visit the root
- Traverse right subtree



3- post-order: Left-Right-Root

- Traverse left subtree
- " right "
- visit root



Class Activity:

pre-order: Root-Left-Right

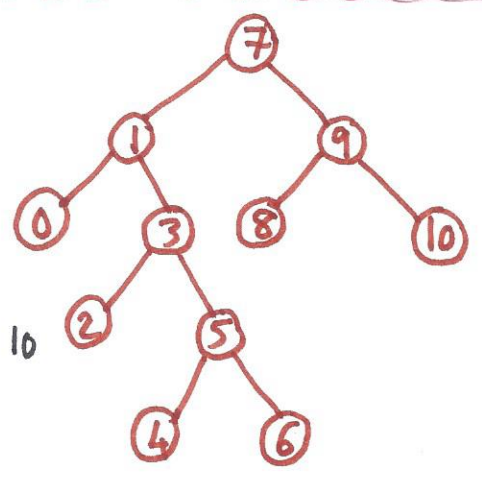
7 1 0 3 2 5 4 6 9 8 10

In-Order: L-Root-R

0 1 2 3 4 5 6 7 8 9 10

post-order: L-R-Root

0 2 4 6 5 3 1 8 10 9 7



Binary Tree: We want a data structure with the following operations:

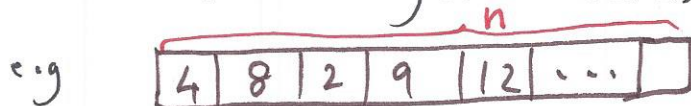
59
Sp. 24
50t

1- Search (x)

2- Insert (x)

3- Delete (x)

- We can use an array or a linked list.



Array
Search (x) $\rightarrow O(n)$

Insert (x) $\rightarrow O(1)$

Delete (x) $\rightarrow O(n)$

Linked List
To check all elements $\rightarrow O(n)$

Insert x at the end $\rightarrow O(1)$

We shift to left $\rightarrow O(n)$

- **Binary Search (Array is sorted):**

Search (x) $\rightarrow O(\log n)$

Insert (x) $\rightarrow O(n)$

Delete (x) $\rightarrow O(n)$

- If one Comparison takes 10^{-6} sec. then 100,000,000 will take 100 Seconds which is too much.

- **Binary Search Example:** We want to search for 10,
(We have sorted array.)

$$\text{mid} = \frac{0+8}{2} = 4 \text{ index}$$

$$\text{mid} = \frac{0+3}{2} = 1.5 \approx 2$$

$$\text{mid} = \frac{0+1}{2} = 0.5 \approx 1$$

0	1	2	3	4	5	6	7	8
6	10	14	17	25	30	35	40	50

$$\frac{n}{2^k} = 1$$

$$\log_2 2^k = \log_2 n$$

$$\log_2 2^k = \log_2 n$$

$$k \cdot \log_2 2 = \log_2 n$$

$$k = \log_2 n \rightarrow O(\log_2 n)$$

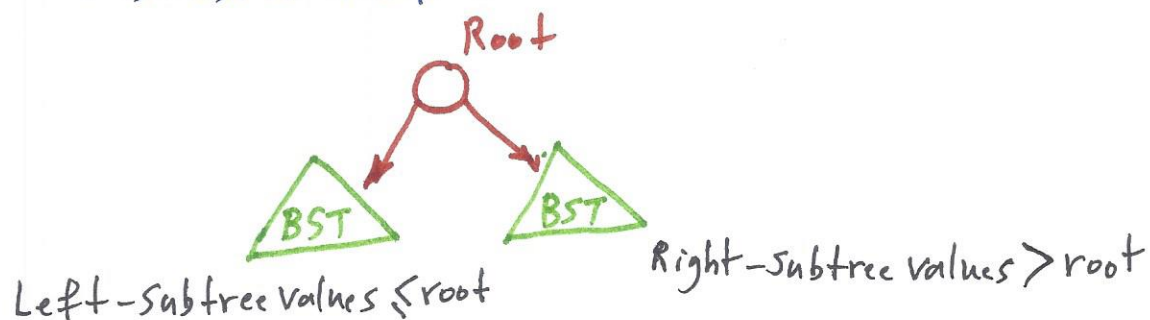
$$\begin{aligned} n &\rightarrow \frac{n}{2^0} \\ \downarrow n/2 &\rightarrow \frac{n}{2^1} \\ \downarrow n/4 &\rightarrow \frac{n}{2^2} \\ \downarrow n/8 &\rightarrow \frac{n}{2^3} \\ \vdots &\vdots \\ \downarrow 1 &\rightarrow \frac{n}{2^k} \end{aligned}$$

$\log n < n \rightarrow$ Then the binary search is faster than regular (linear) search.

- Binary Search Tree (BST):

60
Sp. 24
Sat

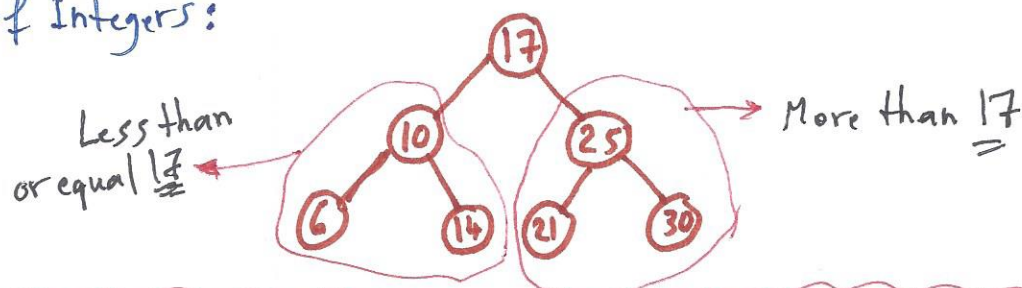
BST is a binary tree in which for each node, values of all nodes in left subtree are lesser or equal and values of all nodes in right subtree are greater than the root. This must be true for all nodes.



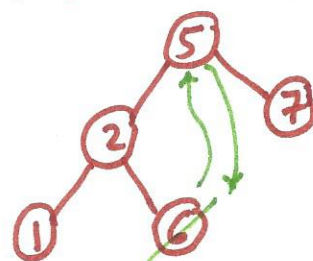
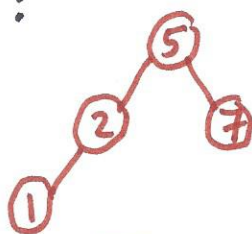
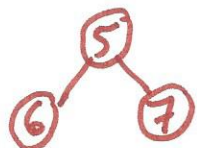
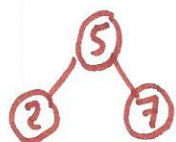
- BST is a recursive data structure.

- Left child will be the root of the left-subtree and similarly right child will be the root of right subtree

Example, BST of Integers:



Check to see if we have a BST:



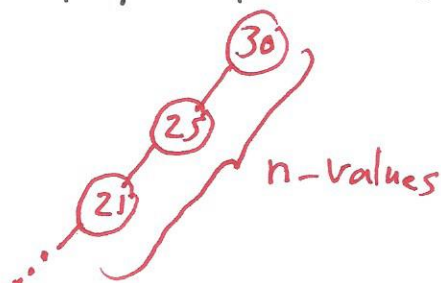
- Search in BST is similar to Binary Search.

- Tree is balanced if for all nodes, the difference between the height of left and right subtree is not greater than 1.

So if a tree is balanced we will start with a search of space of n -nodes, if we discard one sub-tree we will discard $\frac{n}{2}$ nodes.

For example if BST is not balanced:

Worst Case is $O(n)$ \longrightarrow
There is no sub-tree here.

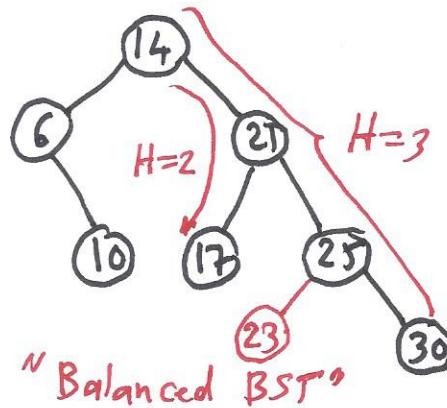
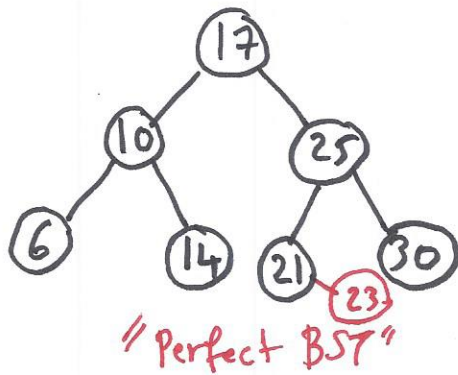
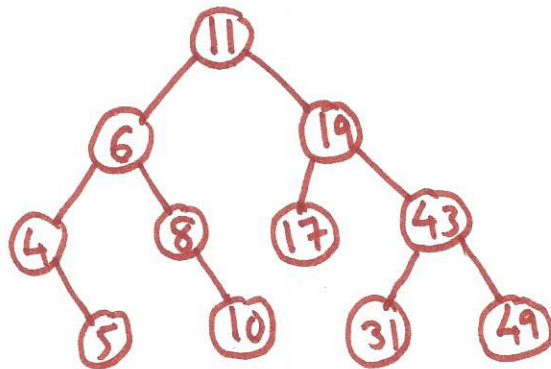


61
5p.24
Sat

Class Activity: Given a Sequence of numbers:

11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31

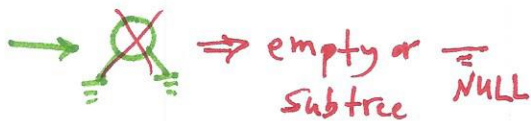
Draw a BST by inserting the above numbers from left-to-right.



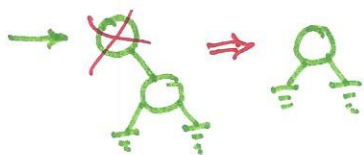
Now insert a value in BST, We need to find a proper position to insert.
e.g. inserting 23

BST (Deleting Nodes): We have 3 cases.

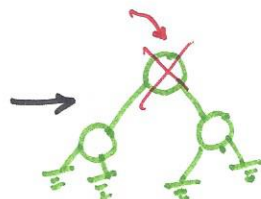
1 - 0 children (leaf node)



2 - A Node with 1 child



3 - A node with 2 children



Rule #1: For two children, find the smallest value in nodes in right Sub-tree, Copy it in the node we want to delete and then delete that node.

Rule #2, Find the max value in left-subtree, Copy the value in targeted node, delete the duplicate from left subtree.

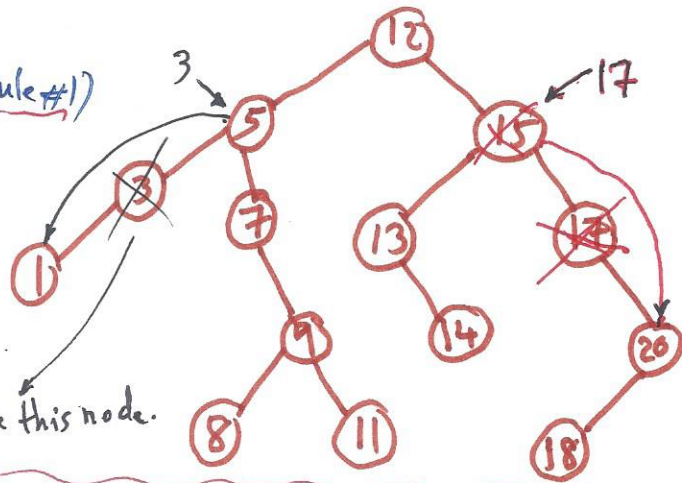
62
5/24
Sat

Example: We want to delete 15 (Rule #1)

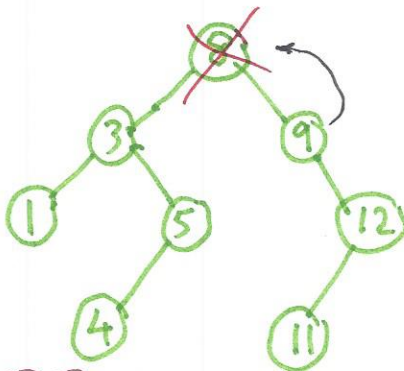
Smallest value in right subtree of 15 is 17. We copy 17 to 15 and we delete the old 17.

- To delete 5 (Rule #2)

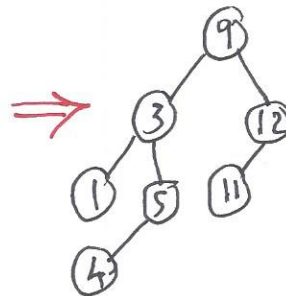
We delete this node.



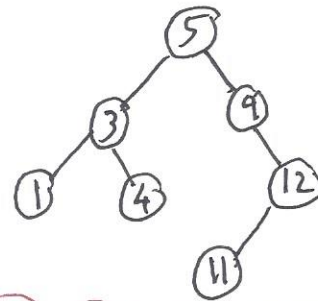
Example: To delete a node like 8.



Rule #1



Rule #2



BST: Implementation

struct TreeNode

```
{ int data;
  TreeNode *left;
  TreeNode *right;
};
```



- We use pointers to dynamically Create nodes in BST and Recursive functions.

