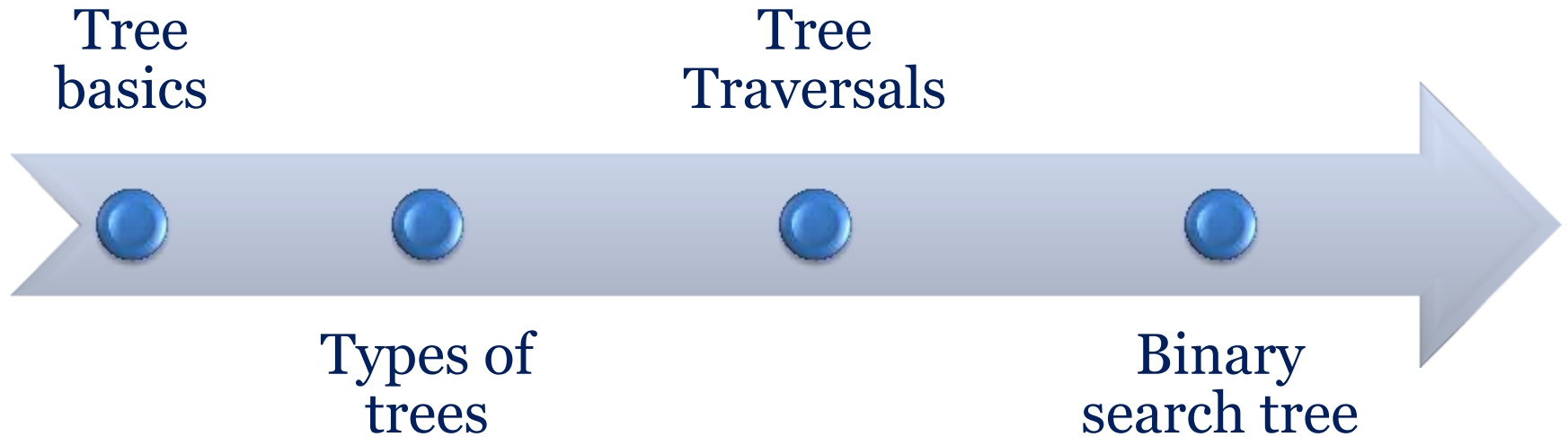


Non linear data structures-Trees

Deliverables



Tree

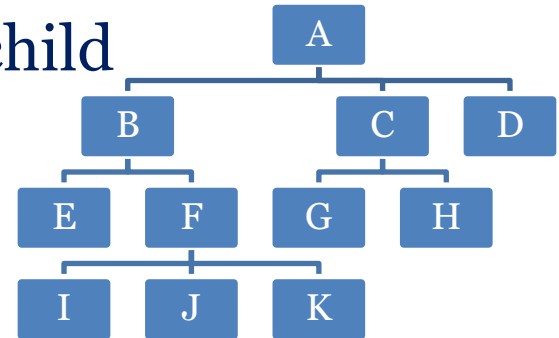
Technically tree is an abstract model of a hierarchical structure

A tree consists of nodes with a parent-child relation

Applications: Organization charts,
File systems , Programming environments

Tree terminology

- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- External node (or leaf node) or Node without children (E, I, J, K, G, H, D)
- Ancestors of a node: parent, grandparent etc
- Depth of a node: number of ancestors



Tree basics

Height of a tree: maximum depth of any node

Descendant of a node: child, grandchild etc.

Sub tree: tree consisting of a node and its descendants

Siblings: Children of the same parent

In degree: number of nodes arriving at that node

Out degree: number of nodes leaving that node

Tree operations

size()

isEmpty()

root()

parent(p)

children(p)

isInternal(p)

isExternal(p)

isRoot(p)

swapElements
(p, q)

replaceElement
(p, o)

Minimum()

Maximum()

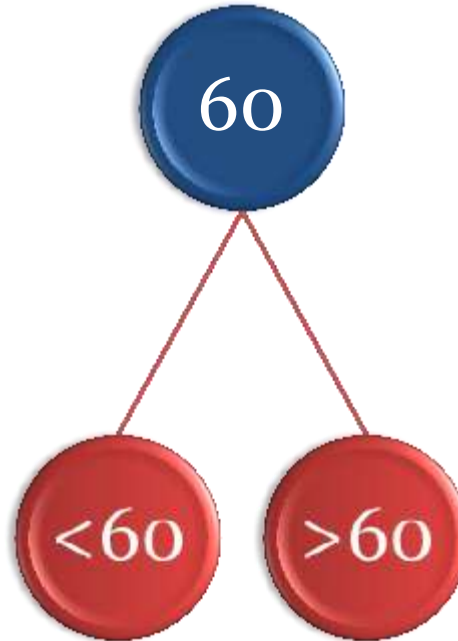
Successor(x)

Predecessor(x)

Search(x)

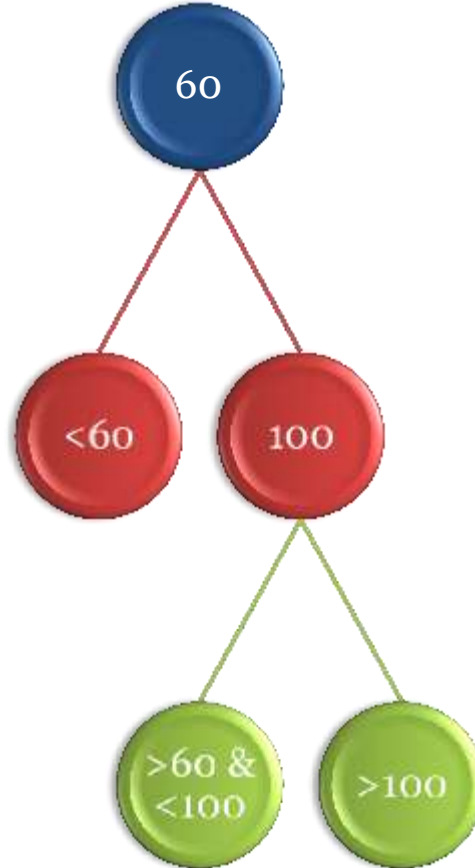
Guess a number between 1 and n

- First Guess 60



Guess a number between 1 and n

- Second Guess 100



Guess a number between 1 and n

How to play so that guesses are minimized

Binary Tree

Each internal node has at most two children

Ordered Binary Tree or Binary Search Tree

Children of node are ordered pair known as left, right child

Left sub tree of node contains nodes with keys $<$ node's key

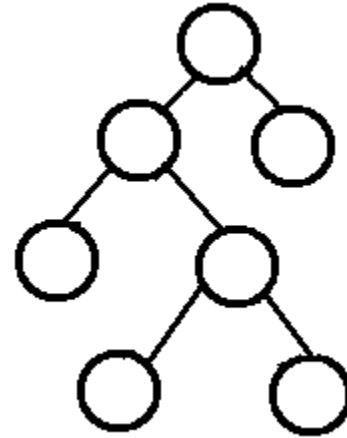
Right sub tree of node has nodes with keys \geq node's key

Both left and right subtrees must be binary search trees

Recursive definition: Either a tree of single node, or whose root has an ordered pair of children, each is binary tree

Complete Binary tree

Complete Binary tree consists of each internal node having exactly two children.

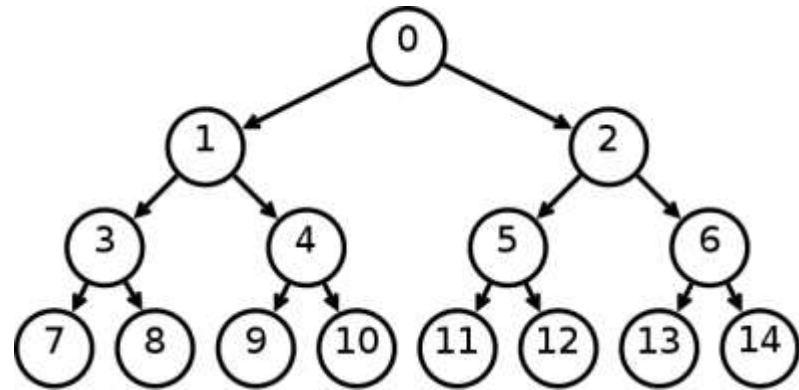
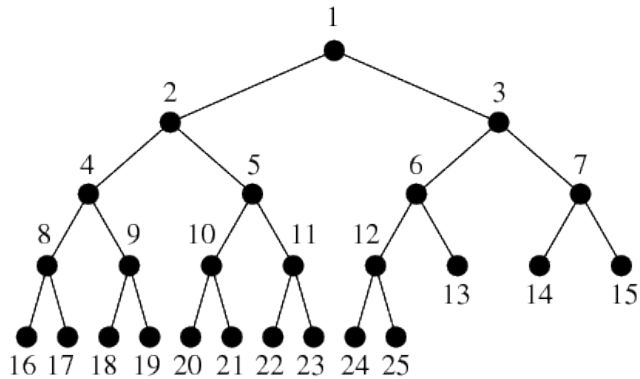


Full and Perfect Binary Tree

Full binary tree has each level of tree completely filled(except possibly the last) in which nodes will be as left as possible

Perfect binary tree will have each internal node having two children and all leaf nodes will be at the same level

Full and Perfect Binary Tree

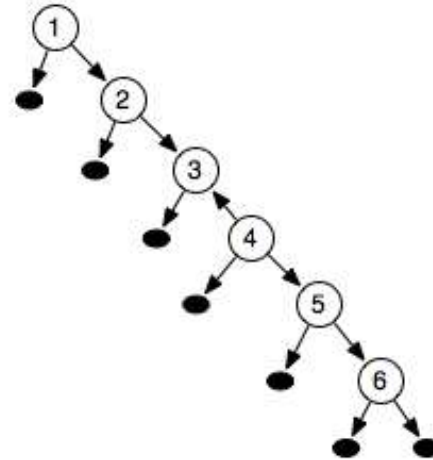
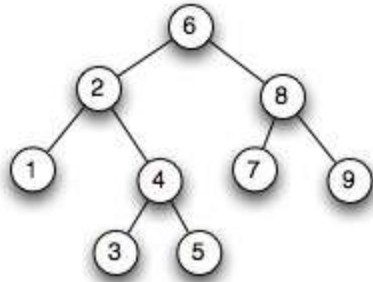


Balanced and Degenerate Tree

Balanced binary tree is a binary tree in which the height of two subtrees of every node never differ by more than 1

Degenerate tree : Where each parent has only one child

Balanced and Degenerate Tree



Few formulas

n number of nodes

e number of external nodes

i number of internal nodes

h height

$n = 2^{h+1} - 1$ for perfect binary trees

min: $n = 2^h$ and max: $n = 2^{h+1} - 1$ for complete binary trees

$e = 2^h$ in a perfect binary tree

Left most child is the smallest element of the BST and right most element is the largest element of the tree

Formulas for complete binary trees

$$e = i + 1$$

$$n = 2e - 1$$

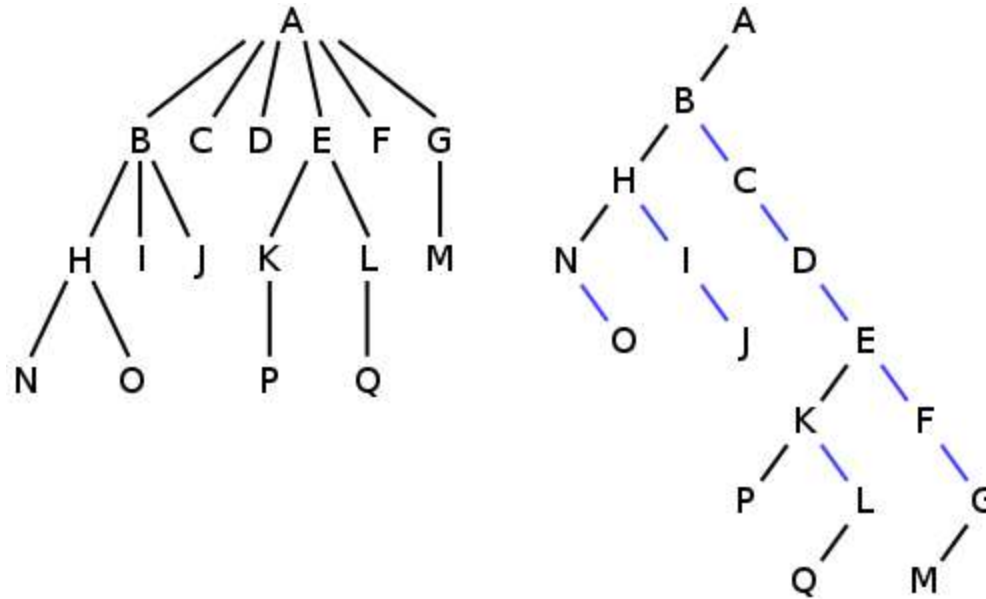
$$h \leq (n - 1)/2$$

$$e \leq 2^h$$

$$h \geq \log_2 e$$

$$h \geq \log_2 (n + 1) - 1$$

Converting a tree to binary tree



Searching in a tree (Recursion)

```
Search(t, k)
```

```
If t=null or k=t
```

```
    return t
```

```
    if k < x
```

```
        return Search(Left(t),k)
```

```
    else
```

```
        return Search(Right(t),k)
```

Searching in a tree (Iterative)

```
Search(t, k)
```

```
While t  $\neq$  null or k  $\neq$  t
```

```
    if k < t
```

```
        t  $\leftarrow$  Left(t)
```

```
    else
```

```
        t  $\leftarrow$  Right(t)
```

```
    return t
```

Complexity of search

It will depend upon the height of the tree because with every loop iteration we are progressing to next level of the loop.

Worst case: when we have a twig $O(n)$

Average case : When tree is nearly complete $O(\log n)$

Best Case: $O(1)$

Using a complex proof it is known that Expected height of a BST built from a random data is $O(\log n)$

Insert(T, k)

```
Insert(T, k)
  if Root(T) = null
    Root(T) = k
  else
    y ← Root(T)
    while y ≠ null
      prev ← y
      if k < y
        y ← left(y)
      else
        y ← right(y)
```

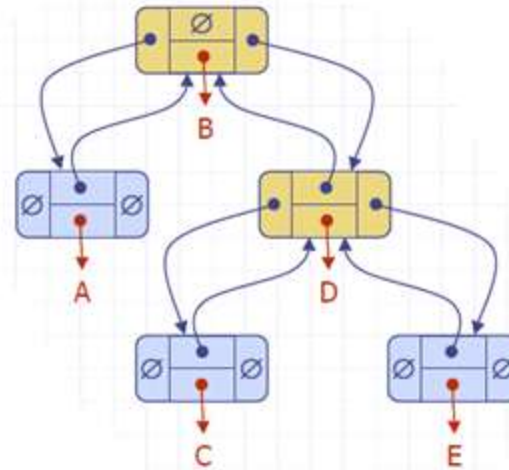
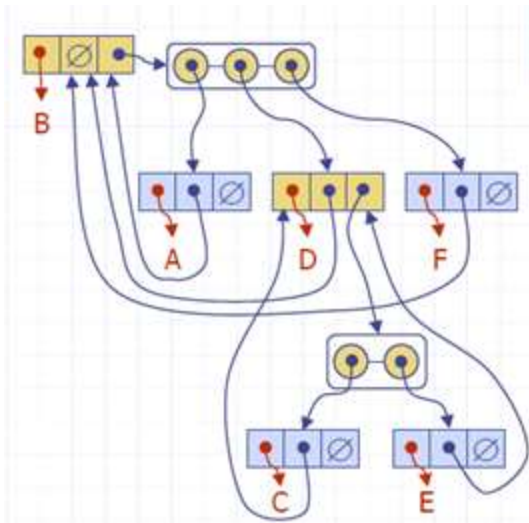
```
parent(k) ← prev
If k < prev
  left(prev) ← x
else
  right(prev) ← x
```

Special Case

What if we need to insert a duplicate element

Does the order of insertion matters : Yes, It will change the topology of the tree

Linked Representation



Tree Traversal

A traversal visits the nodes of a tree in a systematic manner

DFS (Depth first search of a tree has three types Preorder, Postorder and Inorder)

BFS (Breadth first search) of a tree is level wise

Inorder traversal

In inorder traversal, left node is visited before the root node and right node is visited after the root node

Application: It gives data in sorted order in binary search trees

InorderTraversal(x)

If $x \neq \text{null}$

InorderTraversal(Left(x))

Print x // or any other work to be done on that node

InorderTraversal(Right(x)) //Time Complexity $\Theta(n)$

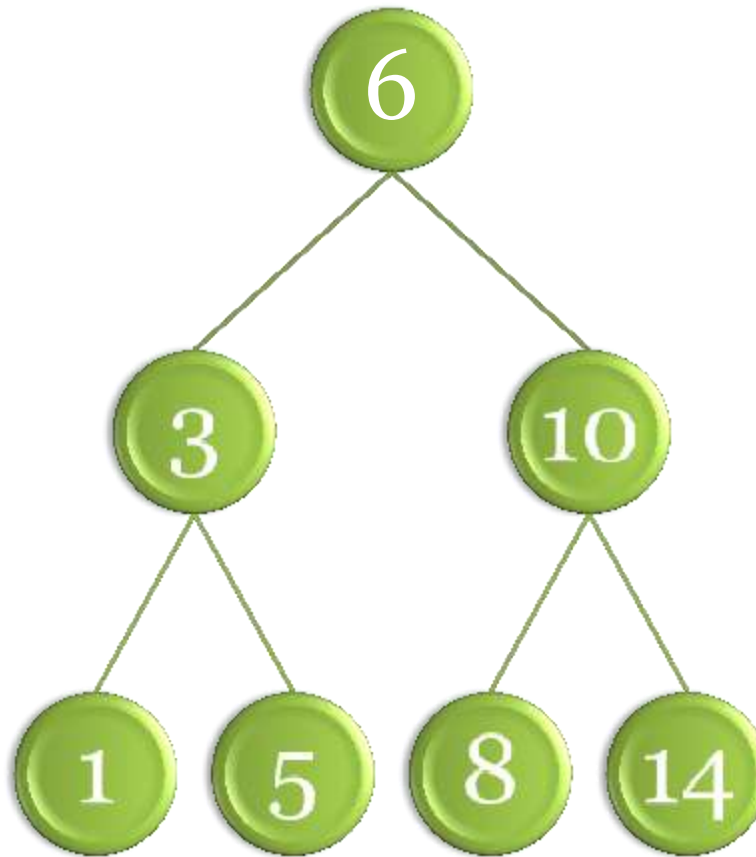
Non recursive inorder traversal

To process a node:

Follow left links until Null (push onto stack).

Pop Print and process.

Follow right link (push onto stack).



6
6 3
6 3 1
6 3
6
6 5
6
-
10
10 8
10
-
14
-

Preorder Traversal

In preorder traversal, a node is visited before its descendants

Application: print a structured document

PreorderTraversal(x)

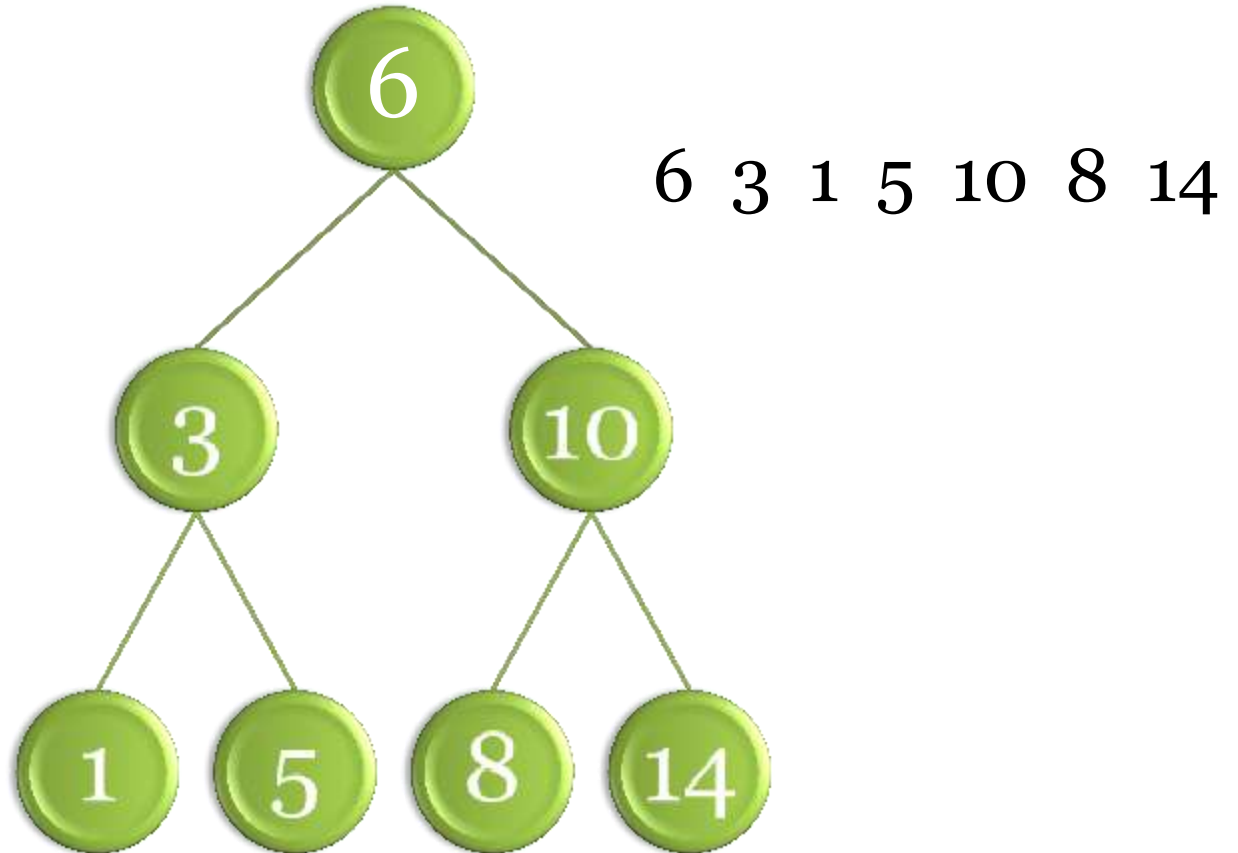
If $x \neq \text{null}$

 print x

 PreorderTraversal(Left(x))

 PreorderTraversal(Right(x))

Preorder Traversal



Postorder Traversal

In postorder traversal, a node is visited after its descendants

Application: compute space used by files in a directory and its subdirectories, Evaluating arithmetic operation

PostorderTraversal(x)

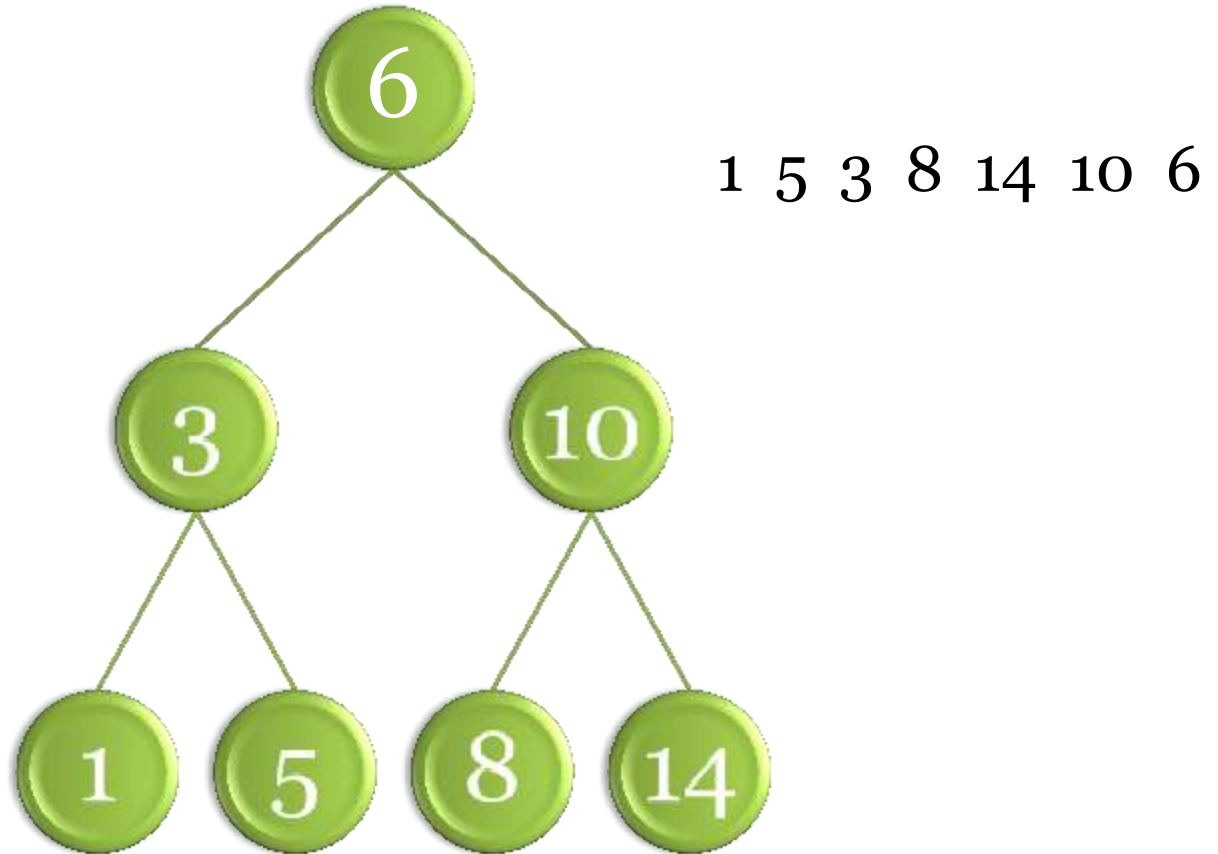
If $x \neq \text{null}$

PostorderTraversal(Left(x))

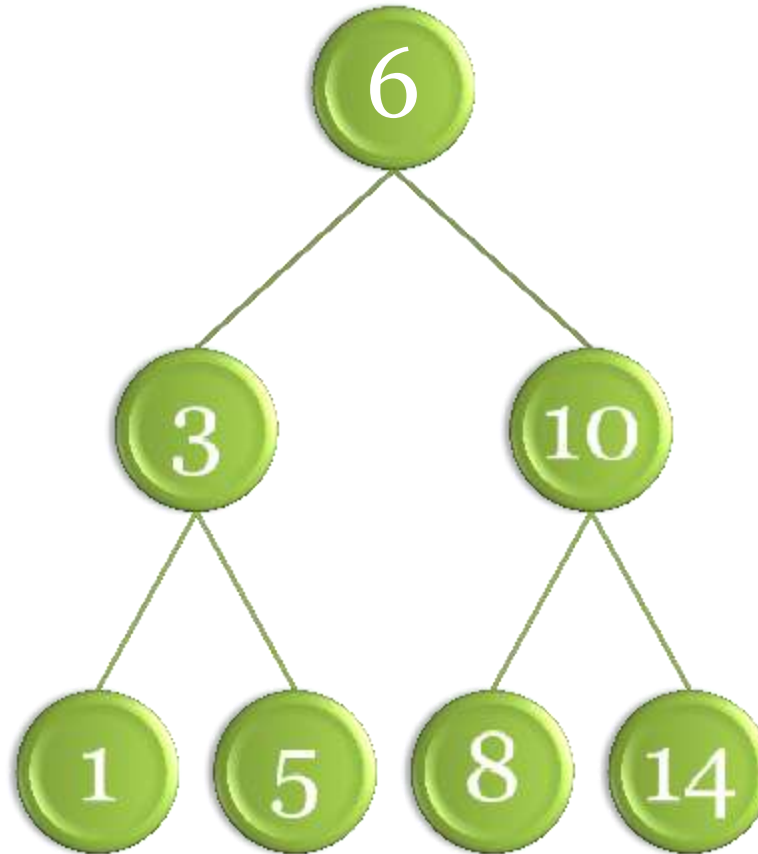
PostorderTraversal(Right(x))

print x

Postorder Traversal

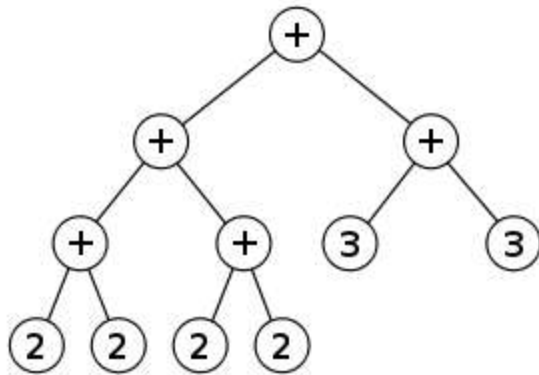


BFS Traversal



6 3 10 1 5 8 14

Arithmetic expression trees



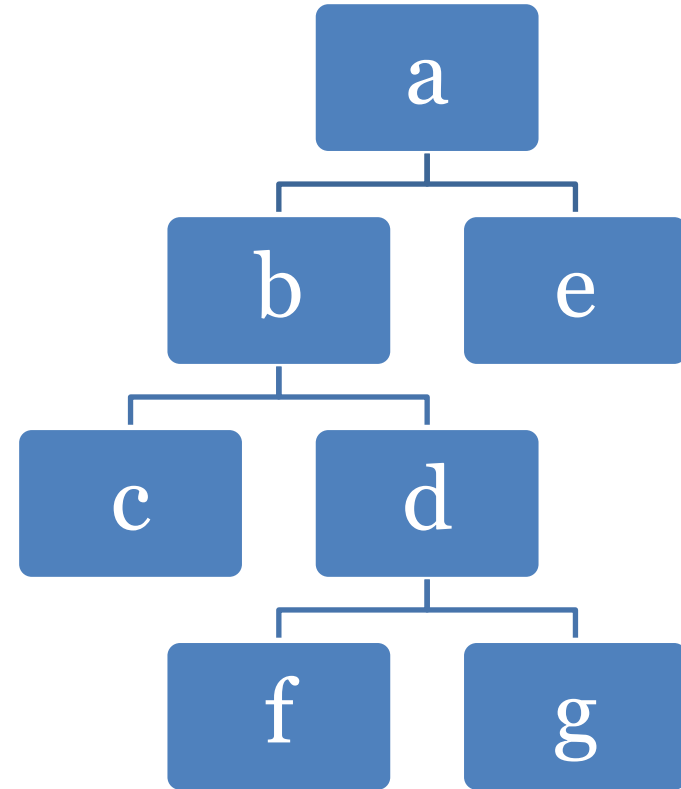
$$((2+2)+(2+2))+(3+3)$$

Infix	Postfix	Prefix
$a*b$	ab^*	$*ab$
$a+b*c$	abc^*+	$+a^*bc$
$a+b*c/d-e$	$abc^*d/+e-$	$-+a/^*bcde$

Infix, Postfix and Prefix notation

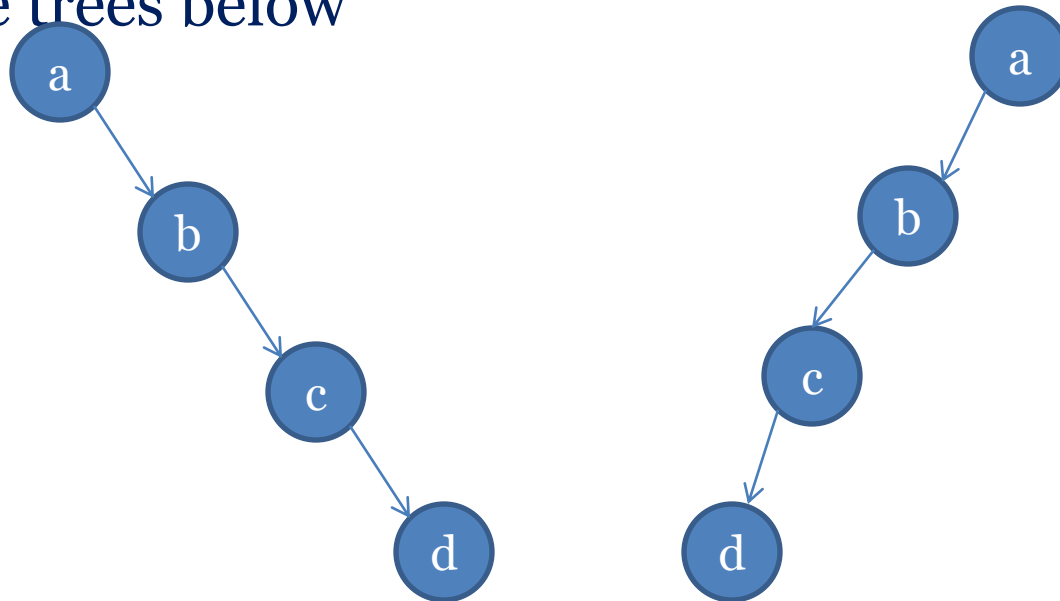
Finding postorder

- Preorder abcdfge
Inorder cbfdgae
- From preorder a is the root, now find a in the inorder traversal
- Now e is the right sub tree
- We are left with sub tree
- Preorder bcdfg
Postorder cbfdg
- Preorder dfg
Postorder fdg



Finding Inorder

We cannot find inorder because there can be two trees with the same pre and post order e.g. Postorder is d c b a for both the trees below



Finding Inorder

If each internal node of the binary tree has at least two children then the tree can be determined from pre and post order traversals.

pre

a b c d f g e

post

c f g d b e a

from this a is the root then e is the right child of a from post order and from Preorder there is nothing after e so e is the leaf

b c d f g

c f g d b

Now b is root of left sub tree and d is right children (from post order) and then from inorder c is only child of d which is left child and so on.

Finding Min and Max

TreeMin(x)

While left(x) ≠ null

x ← left(x)

Return x

TreeMax(x)

While right(x) ≠ null

x ← right(x)

Return x

Finding Successor

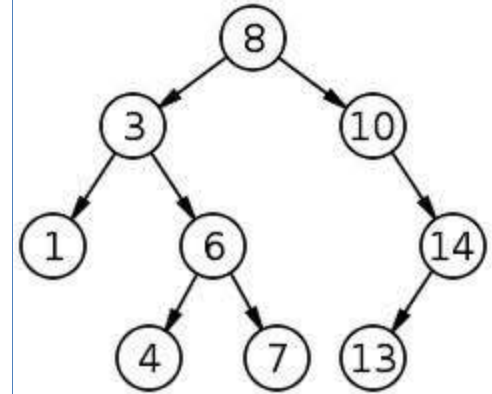
Given x , find the node with the smallest key greater than $\text{key}[x]$

Two cases depend upon right sub tree of x

1: Right sub tree of x is nonempty, then successor is leftmost node in right sub tree

2: Right sub tree of x is empty, then keep going up until we are no longer a right child. If there is no such ancestor then successor is undefined.

we are going to the next node in inorder



Finding Successor

TreeSuccessor(x)

If $\text{right}(x) \neq \text{null}$

 return TreeMin(Right(x))

Else

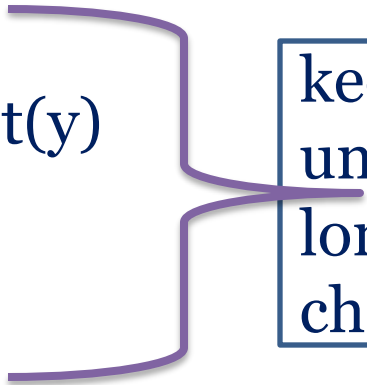
$y \leftarrow \text{parent}(x)$

 while $y \neq \text{null}$ and $x = \text{right}(y)$

$x \leftarrow y$

$y \leftarrow \text{parent}(y)$

Return y



keep going up
until we're no
longer a right
child

Insertion In a binary Search Tree

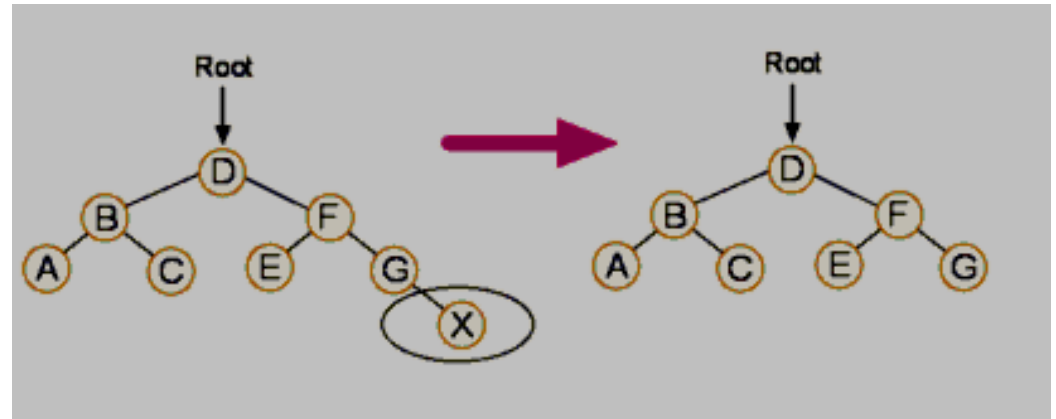
Take an element whose left and right children are null and insert it into T

Find place in T where z belongs (as if searching for z) and add z

Runtime on a tree of height h is $O(h)$

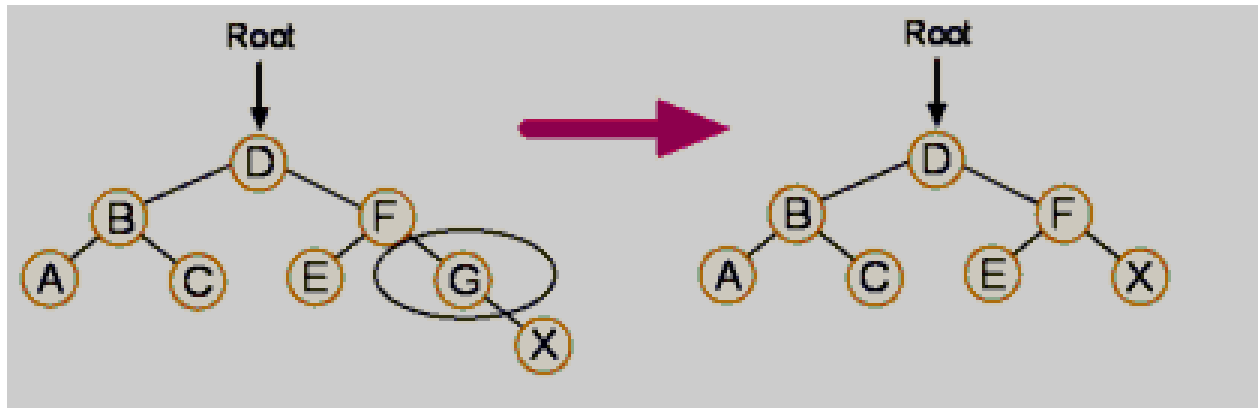
Deletion: Case-I

if x has no children :
just remove x



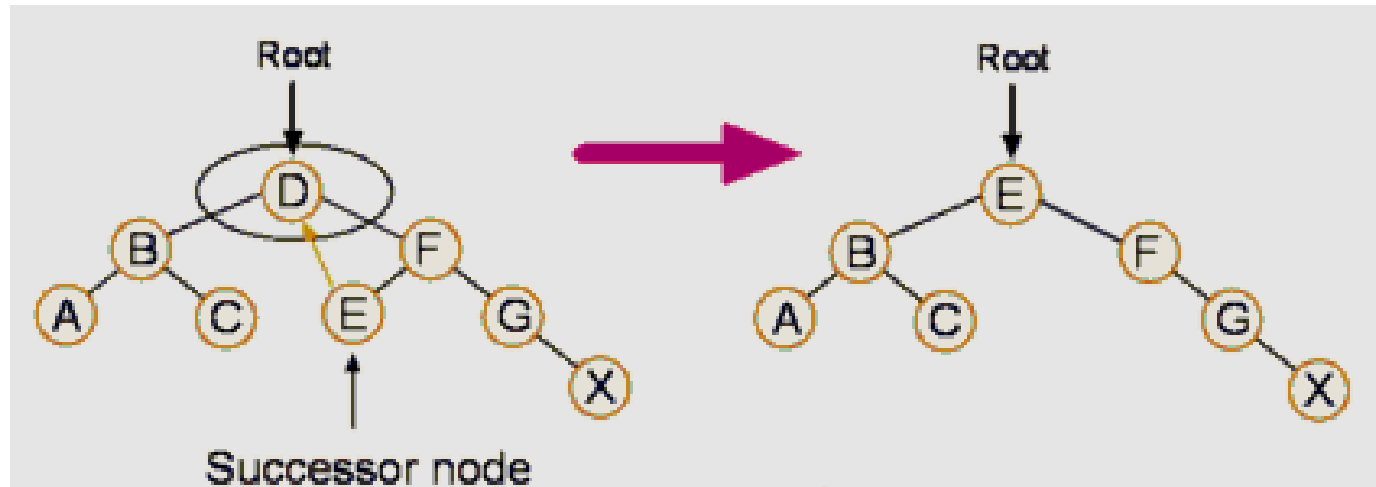
Deletion Case-II

If x has exactly one child then to delete x simply make $p[x]$ point to that child



Deletion Case-III

If x has two children, then to delete it we have to find its predecessor (going left and finding the rightmost node) or successor y and then Replace x with y (it will have at most one child) and delete y



Case-IV

To delete root and successor is undefined, then need to take care of the start pointer

Time complexity

Running time for delete, insertion and search

worst case $O(n)$

Average Case $O(\log n)$

Creating a BST (Inserting n elements one by one)

worst case $O(n^2)$

Average case $O(n \log n)$

Inorder traversal of a BST gives a sorted list and takes $O(n)$ time.

Questions, Suggestions and Comments



Question 1

Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. Which of the following sequences could not be the sequences of nodes examined.

- A) 2, 252, 401, 398, 330, 344, 397, 363
- B) 924, 220, 911, 244, 898, 258, 362, 363
- C) 925, 202, 911, 240, 912, 245, 363
- D) 2, 399, 387, 219, 266, 382, 381, 278, 363

Question 2

A binary search tree is generated by inserting in order the following integers: 50, 15, 62, 5, 20, 58, 91, 3, 8, 37, 60, 24
The number of nodes in the left sub-tree and right sub-tree of the root respectively is

- A) (4, 7)
- B) (7, 4)
- C) (8, 3)
- D) (3, 8)

Question 3

Maximum number of nodes in a binary tree of level k , $k \geq 1$ is

A) $2^k + 1$

B) $2^k - 1$

C) 2^{k-1}

D) $2^{k-1} - 1$