

NAME: K. Grawtham STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: _____

Data Structures

Linear Data Structures

1. Arrays

→ 1-D array

→ 2-D array

- upper triangular matrix
- lower triangular matrix
- Toeplitz matrix
- S-matrix
- C-matrix
- Tri-diagonal matrix
- Square band matrix

→ 3-D array

Ref books

→ DS by Sahani

Programming & DS

15 to 18

Around 7M from DS

2. Stack and Queue

→ Operations on stack

- push, pop, isEmpty, peek, change

→ Applications of stack

• stack permutation

★ • Recursion

• Converting infix to prefix

infix to postfix

• Constructing expression tree using prefix & using postfix

• Multiple stack implementation

→ Queue (least priority)

• operations on queue

• Linear, circular, double ended queue

• Implementation of queue using multiple stacks.

• Implementation of stack using queue

3. Linked List

- * → Single linked list
- Doubly linked list
- * → Circular linked list

- Traversing
- no of nodes
- merging two lists
- Reversing
- operations
- Insertion
- Deletion

Non-Linear Data Structures:

1. Tree

- Properties of trees.
- Complete binary tree, full binary tree, k-ary tree.

→ Tree traversals

- Inorder, preorder, postorder
- converse of inorder, preorder, postorder.

→ Construction of binary a unique binary tree using

- (i) Inorder & Preorder
- (ii) Inorder & postorder

→ Binary search tree

- Insertion, deletion, searching

→ Balanced binary search tree (AVL Tree)

- Max no of nodes

- Min no of nodes

- Insertion, deletion

* → Heap Tree ----- There has been a question almost every yr (9 out of 10)

- Insertion

- Deletion

- Max finding Max & Min elements

- Construction of heap tree

- Inserting keys one after other in given order.

- Inserting using heapify method (Build heap)

- Heap Sort

2. Graphs

- BFS (Breadth First Search Tree)
- DFS (Depth First Search Tree)

3. Hashing

- Advantages of hashing

- Hash function
 - good hash function
 - collision
 - Collision Resolving techniques

* (i) Linear probing (90%)

(ii) Quadratic probing

(iii) Double hashing

(iv) Separate chaining

Previous Year Question Analysis

1987 - 2020 ... 170 questions (approximately)

I 1. Arrays

5%

I 2. Stack & Queues

15%

I 3. Linked list

5%

I 4. Tree

Binary tree

65%

BST

AVL

Heap tree

5%

II 5. Graph

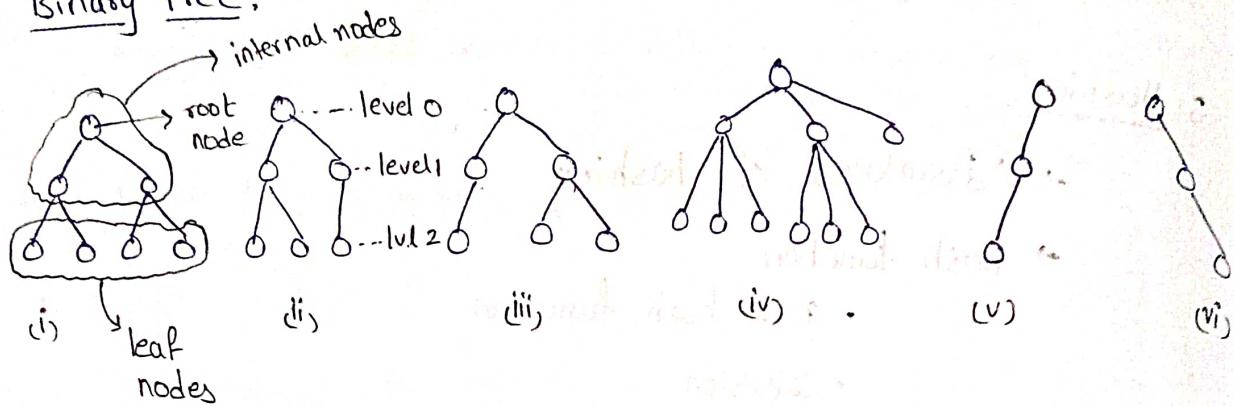
5%

III 6. Hashing

5%

Trees:

Binary tree:



→ In a tree if every node contains atmost 2 children then it is called binary tree.

∴ (iv) is not a binary tree

Full binary tree:

The binary tree in which all levels are full is called full binary tree.

Eg: Fig (i)

Every full binary is complete binary tree

Complete binary tree:

In a complete binary tree, all levels are full except possibly last level and that last level is filled from left to right.

Eg: fig(i), fig(ii)

fig(iii) is not complete binary tree cuz last level is not filled from left to right.

Note:

→ (v) is called left skewed binary tree

→ (vi) is called right skewed binary tree.

k-ary tree:

In a k-ary tree, every internal node contains exactly

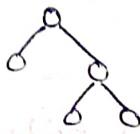
k-children.

\therefore fig(i) is 2-ary tree

fig(ii) & fig(iii) are not 2-ary trees

fig(iv) is 3-ary tree

Eg:



is 2-ary tree



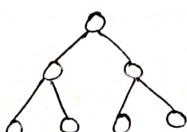
is 3-ary tree

Q) In a complete k-ary tree, if there are n-internal nodes, then how many no of leaf nodes are possible.

- a) nk b) $n(k-1)$ c) $n(k-1)+1$ d) $k(n-1)+1$

Sol:

Consider



This is 2-ary tree

$$\therefore k=2$$

Here no of internal nodes, $n=3$

Here we have no of leaf nodes = 4

a) $n \times k = 3(2) = 6$

b) $3(2-1) = 3$

c) $3(2-1)+1 = 4$

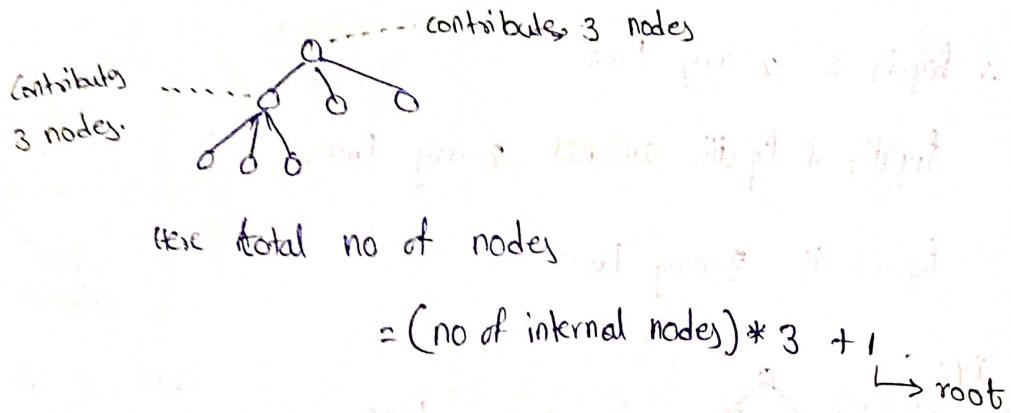
d) $2(3-1)+1 = 5$

\therefore opt(c)

Method 2: (Derivation)

total no of nodes = total no of internal nodes + total no of leaf nodes

Consider a 3-ary tree



Here total no of nodes

$$= (\text{no of internal nodes}) * 3 + 1$$

↓ root

Similarly in a k-ary tree if no of internal nodes = n

$$\text{then total no of nodes} = n * k + 1$$

↓ root
contribution
of each internal node.

$$\text{Now no of leaf nodes} = \text{total no of nodes} - \text{internal nodes}$$

$$= nk + 1 - n$$

$$\text{no of leaf nodes} = n(k-1) + 1$$

∴ opt (c)

- Q2 In a complete n-ary tree, if there are I no of internal nodes and L no of leaf nodes. For I=10 and L=41, find the value of n

Sol:

$$\text{total no of nodes} = \text{internal} + \text{leaf}$$

∴

$$I * n + 1 = \cancel{k+1} I + L \Rightarrow 10n + 1 = 10 + 41$$

$$\Rightarrow n = 5$$

(Q3) In a binary tree, there are 10 internal nodes of degree 2 and 5 internal nodes of degree 1. Find no of leaf nodes.

Sol:

total nodes = ~~internal + leaf~~
internal + leaf

$$(10 \times 2 + 5 \times 1) + 1 = (10 + 5) + l$$

$$l = 26 - 15$$

$$l = 11$$

∴ 11 leaf nodes

(Q4) In a binary tree if there are 20 leaf nodes, then how many no of nodes are having degree 2?

Sol:

If ~~the no of nodes with degree 1~~

(equal to 11) ~~is~~ ~~not~~ ~~possible~~

~~tree~~ ~~(1 + other, which is not)~~ ~~possible~~

~~incorrect answer to degree 1 is 0~~

~~the question is not given in book, pls fix~~

Note:

Since it is a binary tree, the result will be independent of no of internal nodes with degree 1.

Let x be no of internal with degree 1

i be no of internal nodes with degree 2.

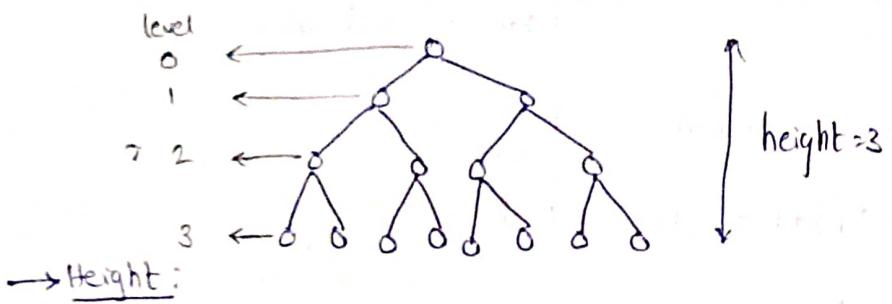
$$(i \times 2 + x \times 1) + 1 = (x + i) + 20$$

$$2i + x + 1 = x + i + 20$$

$$\Rightarrow i = 19$$

08/09/20

Properties of full binary tree:



→ Height:

- * Height of leaf node is always zero

Height of any internal node = longest path from that node to a leaf node

Height of the tree = height of the root node

Note: Full binary tree

(→ Sometimes height may also be defined as no of levels too.)
(In exam, it will be defined)

- * Height of full binary tree = $\log_2(\text{no of leaves})$

$$= \log_2(\text{no of internal nodes} + 1)$$

→ At any. level i, we have 2^i nodes.

→ total no of nodes in a full binary tree is $2^{h+1} - 1$

$h \rightarrow$ height of the FBT

→ total no of nodes in a full k-ary tree is

$$= k^0 + k^1 + k^2 + \dots + k^h$$

$$= \frac{1}{k-1} (k^{h+1} - 1)$$

$$= \frac{k^{h+1} - 1}{k-1}$$

$h \rightarrow$ height of the tree.

→ Total no of nodes of height $k = \left\lceil \frac{n}{2^{k+1}} \right\rceil$

∴ Total no of nodes in tree of height h \Rightarrow total no of nodes

→ Sum of heights of all nodes $= \sum_{k=0}^h \left\lceil \frac{n}{2^{k+1}} \right\rceil \times k$

where

h ... height of the tree.

n ... total no of nodes in the tree.

$$\sum_{k=0}^h \left\lceil \frac{n}{2^{k+1}} \right\rceil \times k = \frac{n}{2} \sum_{k=0}^h \frac{k}{2^k}$$

Consider

$$\sum_{k=0}^{\infty} x^k = 1 + x + x^2 + \dots \text{ for } x < 1$$

$$= \frac{1}{1-x}$$

diff w.r.t x

$$\sum_{k=0}^{\infty} k \cdot x^{k-1} = \frac{1}{(1-x)^2}$$

$$\Rightarrow \sum_{k=0}^{\infty} k \cdot x^k = \frac{x}{(1-x)^2}$$

$$\Rightarrow \sum_{k=0}^{\infty} k \cdot x^k = \frac{x}{(1-x)^2}$$

now

$$\sum_{k=0}^h \frac{k}{2^k} \leq \sum_{k=0}^{\infty} \frac{k}{2^k}$$

Here $x = 1/2$

$$\therefore \sum_{k=0}^h \frac{k}{2^k} \leq \frac{1/2}{(1-1/2)^2}$$

$$\Rightarrow \sum_{k=0}^h \frac{k}{2^k} \leq 2$$

now $\frac{n}{2} \sum_{k=0}^h \frac{k}{2^k} \leq \frac{n}{2} \times 2$

$$\therefore \frac{n}{2} \sum_{k=0}^h \frac{k}{2^k} \leq n$$

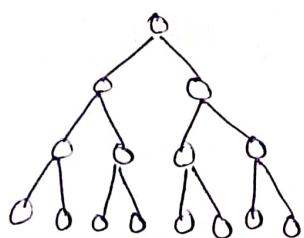
i.e., $\frac{n}{2} \sum_{k=0}^h \frac{k}{2^k} = O(n)$

For exact value for sum of heights in an FBT height 'h'

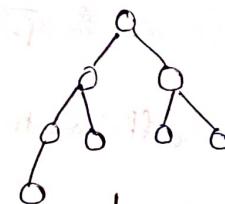
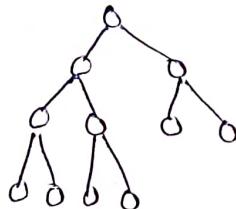
$$= 2^{h+1} - h - 2 = n - h - 1$$

\therefore Sum of heights of all nodes = $O(n)$

Properties of Complete Binary tree:



CBT of height 3 with max no of nodes.



CBT of height 3 with min of nodes

$$\rightarrow \text{Max no of nodes of CBT of height } h = 2^{h+1} - 1$$

$$\text{Min no of nodes of CBT of height } h = \frac{2^h - 1}{2}$$

\therefore If 'n' is no. of nodes in CBT of height 'h' then

$$2^h \leq n \leq 2^{h+1} - 1$$

\rightarrow Max height of CBT is bounded by $O(\log n)$

Proof:

n ... total no of nodes of CBT

W.K.T

$$2^h \leq n \leq 2^{h+1} - 1$$

$$2^h \leq n \quad | \quad n \leq 2^{h+1} - 1$$

$$h \leq \log_2 n \quad \text{--- (1)}$$

$$n+1 \leq 2^{h+1} \quad | \quad \log_2(n+1) \leq h+1$$

$$\log_2(n+1) - 1 \leq h \quad \text{--- (2)}$$

from (1) & (2)

$$\log_2(n+1) - 1 \leq h \leq \log_2 \frac{n}{2}$$

$$\therefore h = O(\log_2 n) = O(\log n)$$

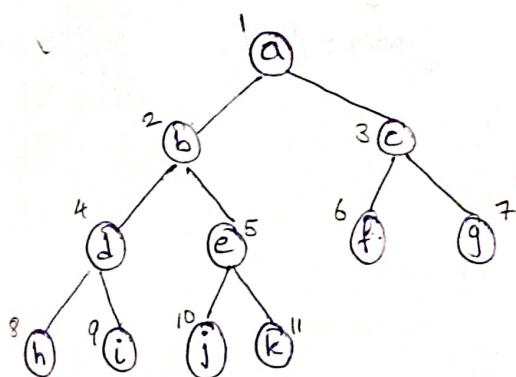
$$\because \log_2 n = \frac{\log n}{\log 2}$$

$$O(\log_2 n)$$

$$= O\left(\frac{\log n}{\log 2}\right)$$

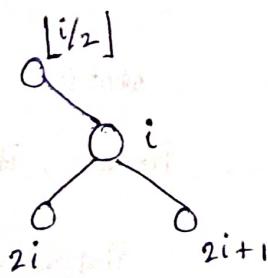
$$= O(\log n)$$

Array representation of binary tree



This representation can be used for any tree

general indices of nodes



→ Root node index is 1.

→ For any node whose index 'i', its left child at index $2i$ and its right child at index $2i+1$, and its parent is at $\lfloor \frac{i}{2} \rfloor$

→ Advantage:

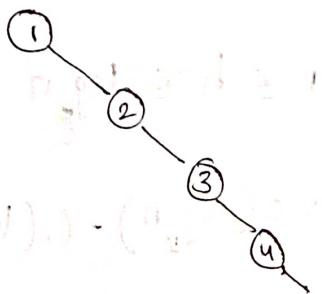
Consumes less space, as we don't need to store pointers (left & right pointers) for arrays.

09/09/20

(Q5) what is the minimum size of the array
any type of binary tree having 'n' nodes?

Sol:

If the binary tree is right skewed then the leaf node has
the maximum index value.

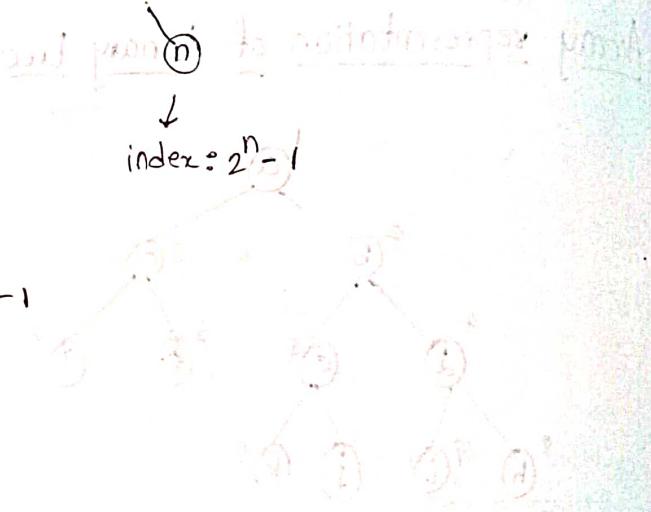


Here height = $n-1$

∴ Max no of nodes possible

$$\begin{aligned} &= 2^{n-1+1}-1 \\ &= 2^n-1 \end{aligned}$$

∴ size of array = 2^n-1



Tree Traversals

Type of questions:

Given

i) Binary tree diagram

what is

Preorder, Postorder, Inorder

ii) Preorder + Inorder
of binary tree

Postorder

iii) Postorder + Inorder
of a binary tree

Preorder

iv) Binary Search tree

Inorder, Preorder, Postorder

v) Preorder of BST

Postorder

vi) Postorder of BST

Preorder

vii) Inorder + Preorder

resultant binary tree

Inorder

1) Tr

2) Pr

3) Tr

Eg:



Inorder

Preorder

Postorder

Eg:

(viii) Inorder + Postorder

resultant binary tree

(ix) Inorder, Preorder, Postorder

Time complexity

(x) Converse of Inorder, Preorder, Postorder of a binary tree.

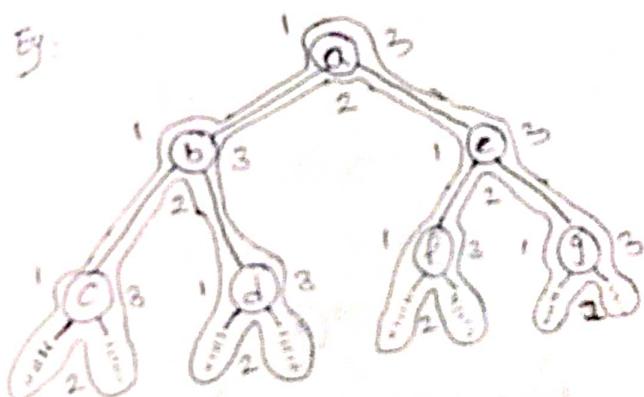
Inorder:

i) Traverse the left subtree of root node in Inorder

ii) Print the root node.

iii) Traverse the right subtree of rootnode in inorder

Eg:



we touch every three times



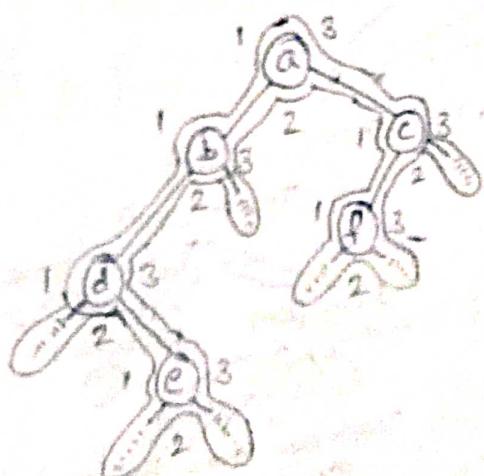
Traversal	when to print
inorder	bottom
preorder	left
postorder	right

Inorder: c b d a f e g (all 2's)

Preorder: a b c d e f g (all 1's)

Postorder: c d b f g e a (all 3's)

Eg:



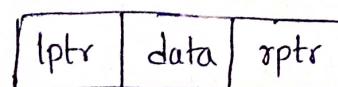
Inorder: d e b a f c

Preorder: a b d e f c

Postorder: e d b f c a

Recursive approach for tree traversals

so node structure



struct node

{

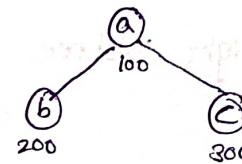
 struct node * lptr;

 struct node * rptr;

 int data;

}

Struct node * T;



→ Inorder (struct node * T)

{

if ($T \neq \text{NULL}$)

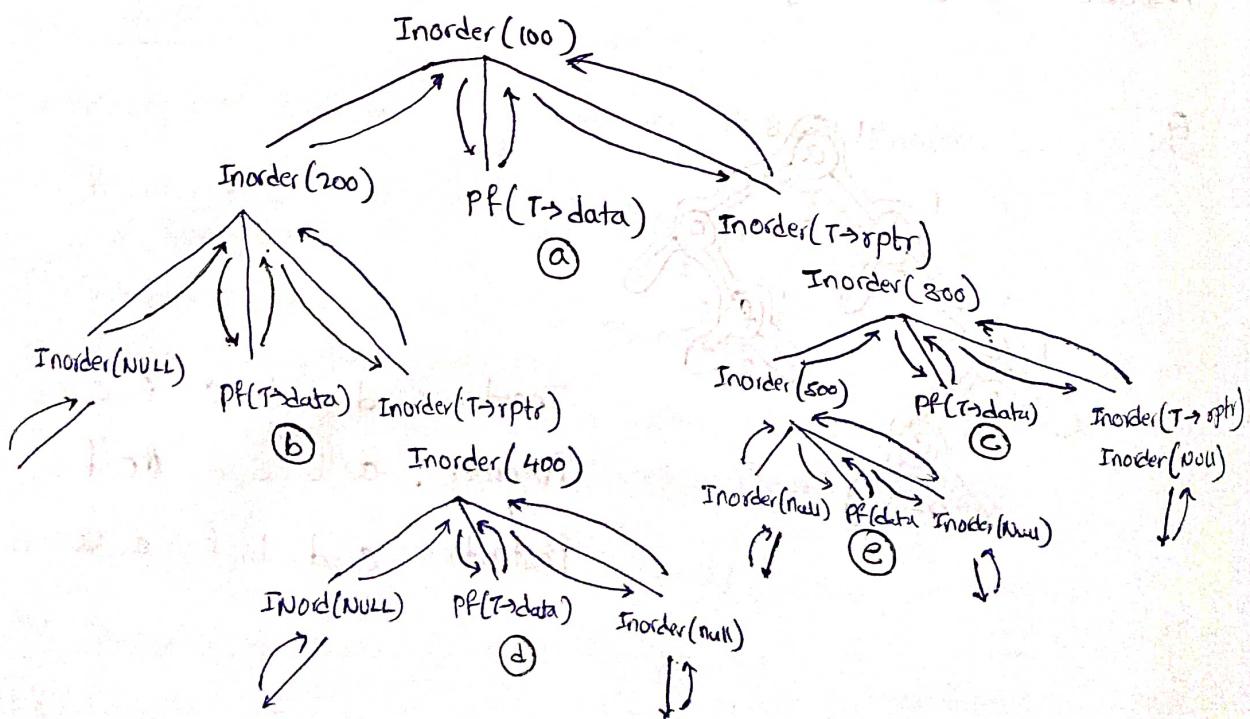
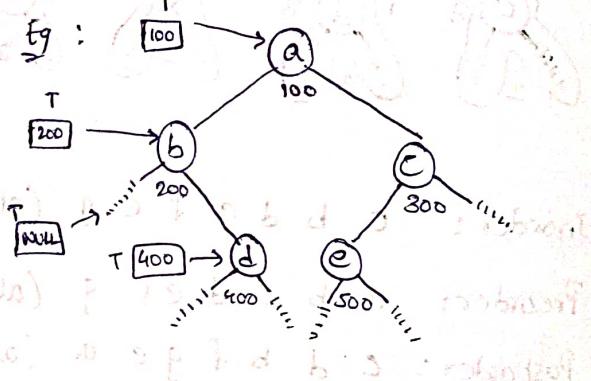
 return;

 Inorder ($T \rightarrow \text{lptr}$);

 printf (" . - d ", $T \rightarrow \text{data}$);

 Inorder ($T \rightarrow \text{rptr}$);

}



\therefore Inorder traversal: b a c i.e. print from left to right.

\rightarrow Preorder (struct node *T)

{ if ($T=NULL$)

return;

printf("%d", T->data);

preorder(T->lptr);

preorder(T->rptr);

}

\rightarrow Postorder (struct node *T)

{ if ($T=NULL$)

return;

Postorder(T->lptr);

Postorder(T->rptr);

printf("%d", T->data);

}

Note:

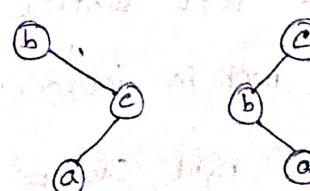
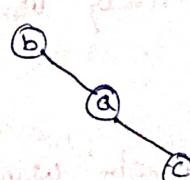
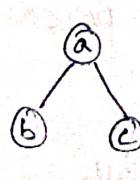
\rightarrow For a given binary tree, no of possible inorder traversals = 1

\rightarrow For a given binary inorder traversal, with n elements is

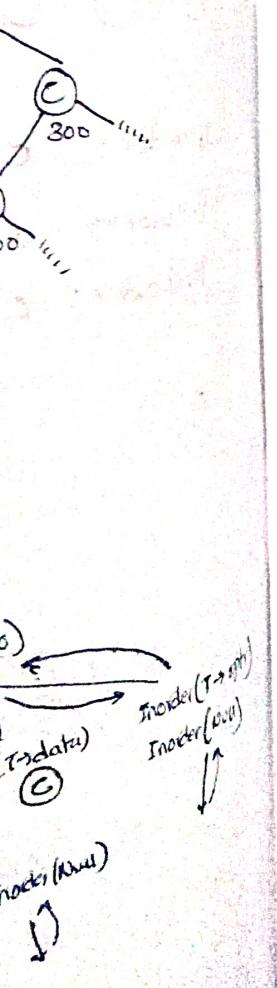
$$\text{no of binary trees possible} = \frac{1}{n+1} 2nC_n$$

Eg: Proof:

assume the inorder traversal = bac



For all above trees inorder traversal is b a c



For $n=3$, no of binary trees with given inorder traversal

$$= \frac{1}{3+1} 2(3) C_3$$

$$= \frac{1}{4} 6C_3 = 5$$

→ Similarly, for the given preorder sequence of n elements,

$$\text{no of binary trees possible} = \frac{1}{n+1} 2^n C_n$$

→ Similarly for given postorder sequence of n elements

$$\text{no of binary trees possible} = \frac{1}{n+1} 2^n C_n$$

Note:

→ for given preorder & inorder there exists unique binary tree.

→ " " postorder & inorder " "

→ However we cannot build a unique tree (i.e., more than one exists) for given preorder & postorder.

Note:

→ Time complexity of inorder, preorder & ~~postorder~~ postorder (iteratively) is bounded by sum of heights of all nodes.

$$\therefore \text{time complexity (iteratively)} = O(n)$$

13/09/20

Construction of unique binary tree using preorder & Inorder

Step 1: pickup the first key of preorder and create a node of it.

Step 2: Create left subtree recursively using the values preceding the root in inorder.

Step 3: Create right subtree recursively using the values following the root in inorder.

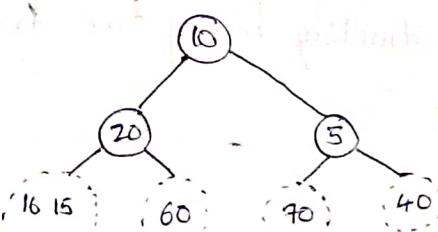
traversal

Eg.: Inorder: 16 15 20 60 10 70 5 40

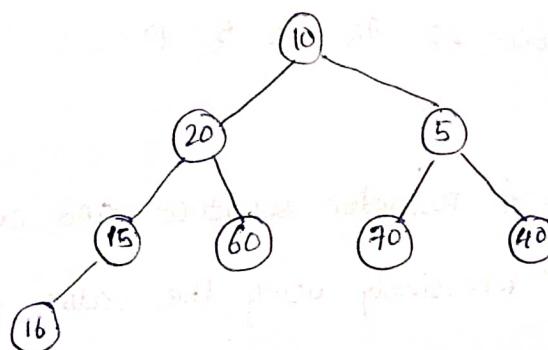
Preorder: 10 20 15 16 60 5 70 40



Inorder: [16 15 20 60] Preorder: [20 15 16 60]



In: [16 15] In: [60] In: [70] In: [40]
 Pre: [15 16] Pre: [60] Pre: [70] Pre: [40]



Analysis:

For analysis we need to consider worst case

Assume we have 'n' nodes

→ Here this worst case occurs

→ Creating node for 1st element of preOrder takes constant time.

→ Now we need to search for that node in the inorder seq.

→ In worst case we need n comparisons.

→ If we continue this, in the next search no of comparison required would be $n-1$ (worst case)

\therefore total no of comparisions

$$= n + (n-1) + (n-2) + \dots + 1$$

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

$$= \frac{n^2+n}{2}$$

$$= O(n^2)$$

\therefore time complexity for constructing binary tree from postorder and inorder = $O(n^2)$

Construction of binary tree from Postorder & Inorder.

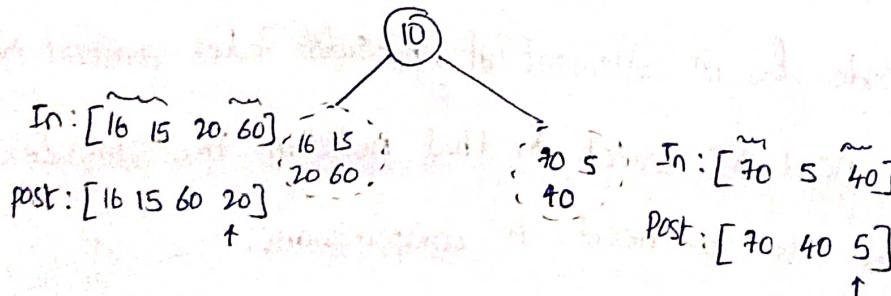
Inorder: 16 15 20 60 10 70 5 40

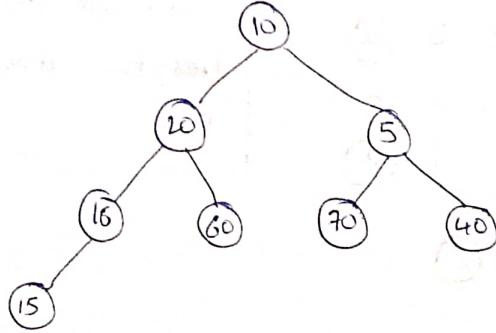
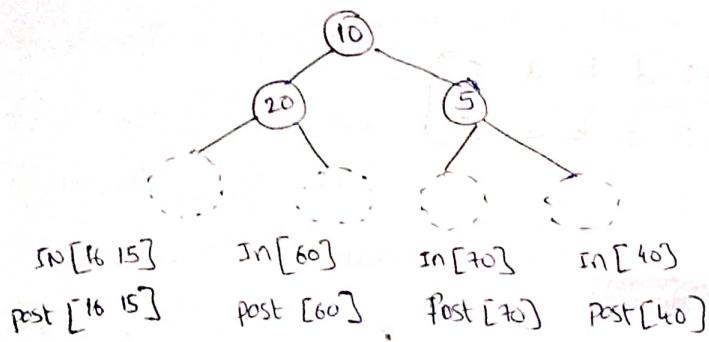
postorder: 16 15 60 20 70 40 5 10

Step 1: Pickup the last key of postorder sequence and create a node.

Step 2: Create left subtree recursively using the values preceding the root in inorder

Step 3: Create right subtree recursively using the values following the root in the inorder.





Analysis:

Same as previous analysis

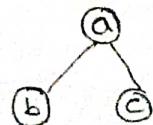
Here also worst case is for left skewed tree.

\therefore time complexity: $O(n^2)$

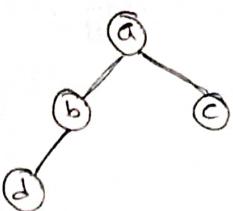
Q6) Let LASTIN, LASTPRE, LASTPOST represent the last key of inorder, preorder, postorder traversals of a complete binary tree. Then which of the following is ~~true~~ true?

- a) LASTIN = LASTPRE
- b) LASTPRE = LASTPOST
- c) LASTPOST = LASTIN
- d) none of these

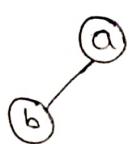
Sol:



In: b a c
Pre: a b c
Post: b c a



In: d b a
Pre: a b d



In: b @
Pre: a b
Post: b @

∴ opt(d)

Last elements of preorder and inorder will differ if the tree doesn't contain RST

- (Q7) The following three are known to be preorder, inorder, postorder traversals of a binary tree. But it is not known which one is which traversal of the binary tree.

I. MBCAFHPYK

II. KAMEBYPFH

III. MABCKYFPFH

a) I - inorder III - postorder

b) I - Inorder III - preorder

c) II - inorder III - preorder

d) III - inorder II - preorder

Sol:

Root is 1st key of preorder & last key of postorder.

∴ 1st key of preorder = last key of postorder.

∴ I - post order

II - pre order.

⇒ III - inorder

∴ opt(d)

Q8) Which of the following combination creates a unique binary tree?

I) Inorder & preorder

II) Inorder & postorder

III) preorder & postorder

~~IV)~~

a) I, III

b) II, III

c) I, II

d) I, II, III

∴ opt C

Q9) How many minimum and maximum no of nodes of binary tree are possible if the height of the binary tree is 6.

Sol:

→ (need not to be left or right skewed)

Min nodes \Rightarrow one node per level = 7

Max nodes \Rightarrow full binary tree of height 6

$$\Rightarrow 2^{6+1} - 1 = 127$$

Q10) Let T be a binary tree with 15 nodes. Then the minimum and maximum possible height of binary is

Sol:

If n is no of nodes and h is height the

$$h+1 \leq n \leq 2^{h+1} - 1$$

$$n \geq h+1$$

$$15 \geq h+1 \Rightarrow h \leq 14$$

$$n \leq 2^{h+1} - 1$$

$$15 \leq 2^{h+1} - 1 \Rightarrow 2^{h+1} \geq 16$$

$$h+1 \geq 4$$

$$h \geq 3$$

$$\therefore 3 \leq h \leq 14$$

∴ min height = 3

max height = 14

→ min height is obtained
for complete binary tree
→ Max height is possible
if we have exactly one
node per level

(11) Consider the following nested representation of a binary tree

(x, y, z) : indicates that y & z are left subtree and right subtree respectively of node x . y, z may be `NULL`, further nested. Which of the following is valid representation of binary tree.

- a) $(1 \ 2 \ (4 \ 5 \ 6 \ 7))$
- b) $(1 \ ((2 \ 3) \ 5 \ 6) \ 7))$
- c) $(1 \ (2 \ 3 \ 4) \ (5 \ 6 \ 7))$
- d) $(1 \ (2 \ 3 \ \text{NULL}) \ (4 \ 5))$

Sol:

- a) $(1 \ 2 \ (4 \ 5 \ 6 \ 7))$

left
subtree

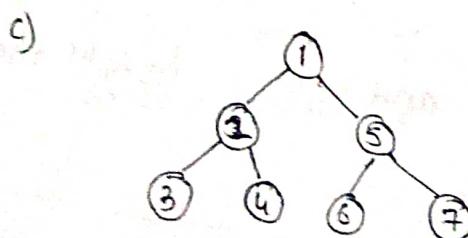
↳ cannot have 4 elements

∴ It should be of type (x, y, z)
∴ invalid

- b) $(1 \ ((2, 3) \ 5 \ 6) \ 7))$

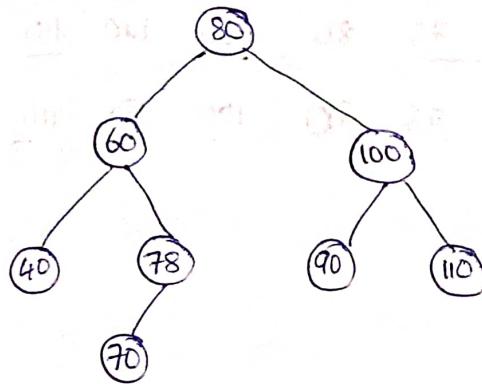
left subtree

right



- c) right subtree has only 2 elements
∴ invalid.

Binary Search Tree:



values of $LST(80) \leq 80 \leq$ values of $RST(80)$

values of $LST(60) \leq 60 \leq$ values of $RST(60)$

values of $LST(100) \leq 100 \leq$ values of $RST(100)$

Defn: For every node 'k' of binary tree if it satisfies the property where values of $LST(k) \leq k \leq$ values of $RST(k)$ then such binary tree is called A binary search tree.

Consider

Inorder: 40 60 70 78 80 90 100 110

Preorder: 80 60 40 78 70 100 90 110

Postorder: 40 70 78 60 90 110 100 80

Note:

The inorder sequence of BST is always in ascending order.
This condition is bidirectional.

Model 1: If the preorder sequence of BST is

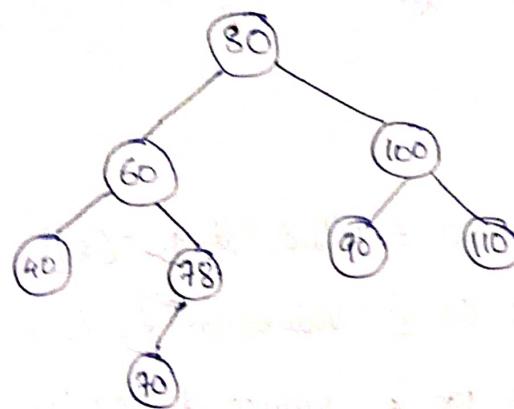
80 60 40 78 70 100 90 110

then what is post order sequence.

Sol :

Inorder is in increasing order

Inorder: 40 60 70 78 80 90 100 110
Preorder: 80 60 40 78 70 100 90 110
↑



Now postorder sequence is

40 70 78 60 90 110 100 80

Analysis (Construction of BST from Inorder & preorder)

→ Here we choose the first element of preorder and create a node.

→ Now we search for this node using in the inorder sequence using $\log n$.

→ ~~After~~

∴ total no of comparisons

$$= \log n + \log(n-1) + \log(n-2) + \dots + \log 1$$

$$= \log(n(n-1)(n-2) \dots (1))$$

$$= \log(n!) \leq \log(n^n) \leq n \log n \quad (\because n! \leq n^n)$$

$$= O(n \log n)$$

$$(\because n! \leq n^n)$$

(Q12) You are given the post order traversal 'P' of a BST on n elements: $[1, 2, 3, \dots, n]$. You have to determine the unique binary search tree that has P as post order traversal. Then what is the time complexity of most efficient algorithm for doing this.

- a) $O(n^2)$ b) $O(n \log n)$ c) $O(\log n)$ d) $O(n)$

Sol:

opt(b)

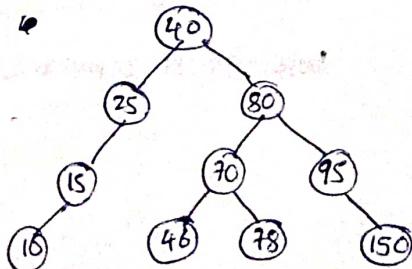
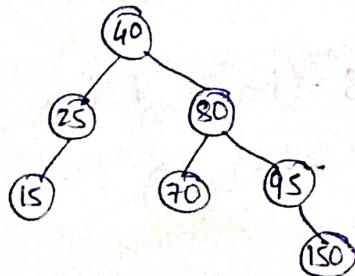
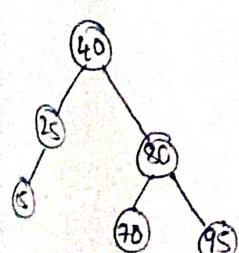
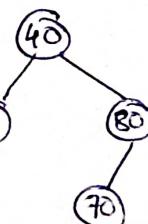
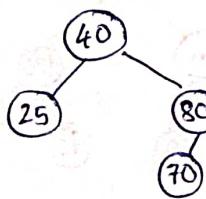
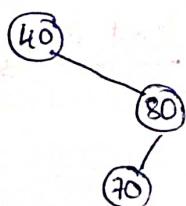
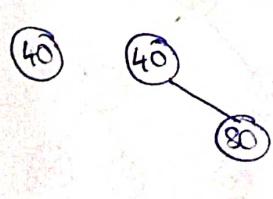
time to sort = $O(n \log n)$ \therefore Heap sort

time to build tree = $O(n \log n)$

$\therefore O(n \log n)$

Model 2: Construct BST by inserting the following keys in the given order and find height of the binary search tree. Find no of nodes in the left subtree of root node. List out all leaf nodes of the BST.

keys: 40 80 70 25 15 95 150 78 46 10



height of the BST : 3

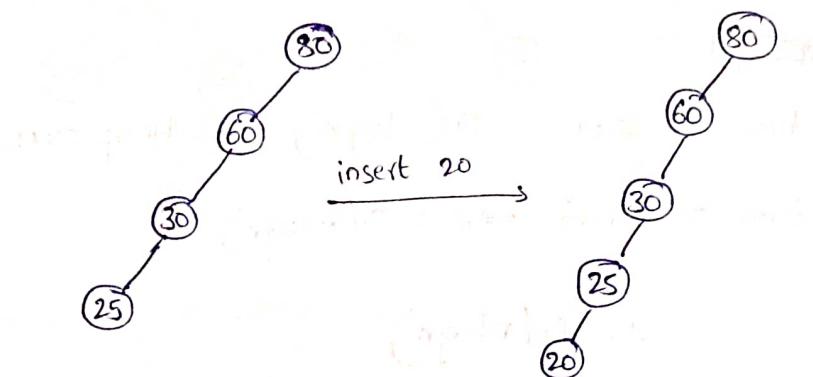
no of nodes in left subtree of the root : 3

leaf nodes : 10 46 78 150

Analysis (Inserting node into BST)

Case ii):

Insert 20 into following BST



Here no of comparisions = 4

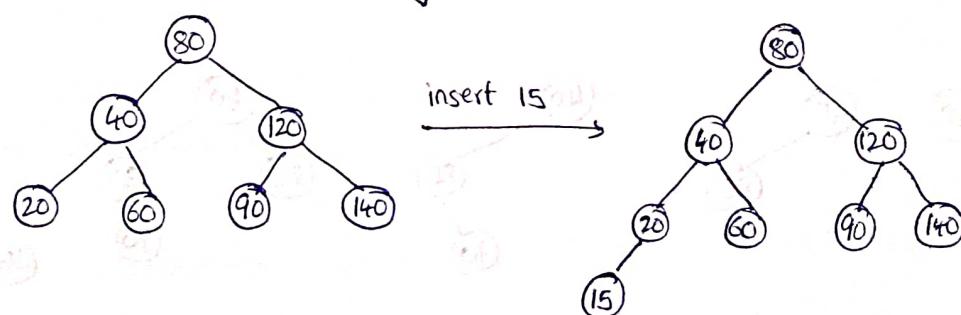
~~no of comparisons = height + 1~~

\therefore time complexity = $O(n)$

↳ no of nodes = height + 1

Case iii):

Insert 15 into the following BST



Here no of comparisions = 3 = (height of tree + 1)

$= O(\log_2 n)$

Note :

time complexity for ~~insert~~ insertion into BST = $\begin{cases} O(n) & \text{left or right skew} \\ O(\log_2 n) & \text{(or 1 node per level)} \end{cases}$ full binary searchtree

Q13 Suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted into an initially empty BST. what is inorder traversal of resultant tree?

- a) 7 5 1 0 3 2 4 6 8 9
- b) 0 2 4 3 1 6 5 9 8 7
- c) 0 1 2 3 4 5 6 7 8 9
- d) 9 8 6 4 2 3 0 1 5 7

Sol :

Inorder traversal is increasing order

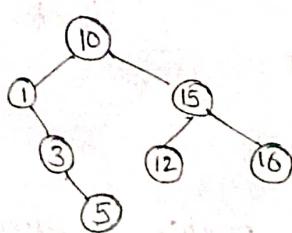
\therefore opt c

Q14 The following numbers are ~~inserted~~ inserted into an empty

BST in the given order: 10 1 3 5 15 12 16. what is

height of the BST (the height is max distance of leaf node from root)

Sol :

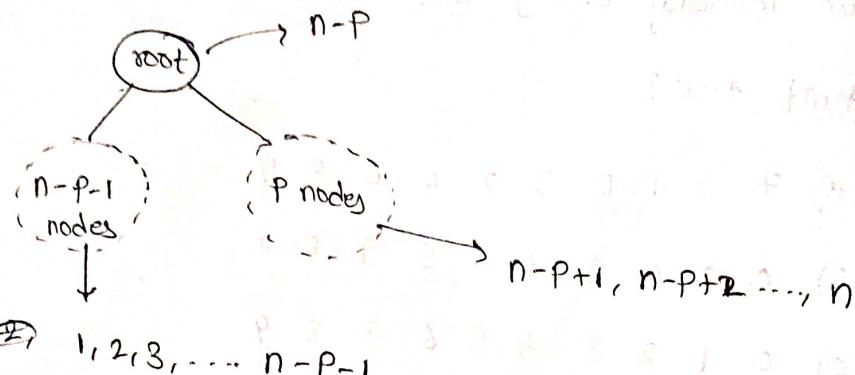


\therefore height = 3

(Q15) The numbers $1, 2, \dots, n$ are inserted in a BST in some order. In the resulting tree, the right subtree of the root contains P nodes. The first number to be inserted in the tree must be _____

- a) P b) $P+1$ c) $n-P$ d) $n-P+1$

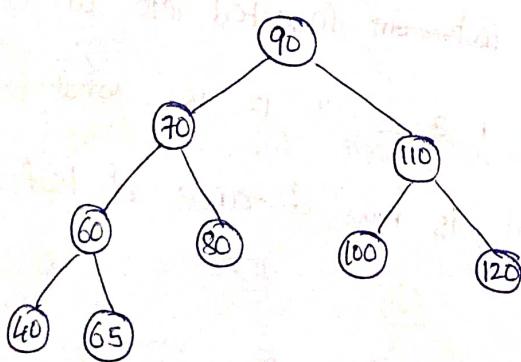
Sol:



\therefore root = $n-P$ is 1st node to be inserted.

Deletion operation on BST:

Case i): Deleting leaf node



Delete 40 from above BST

→ Now search for 40

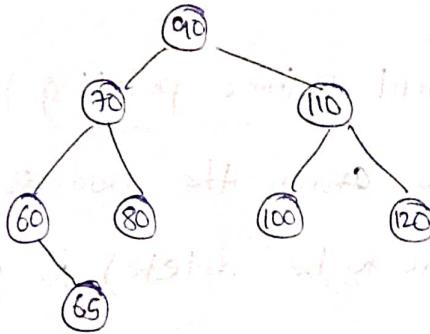
→ Now make poi. assign
to 40, to NULL

Step 1: Search if the given key is present or not

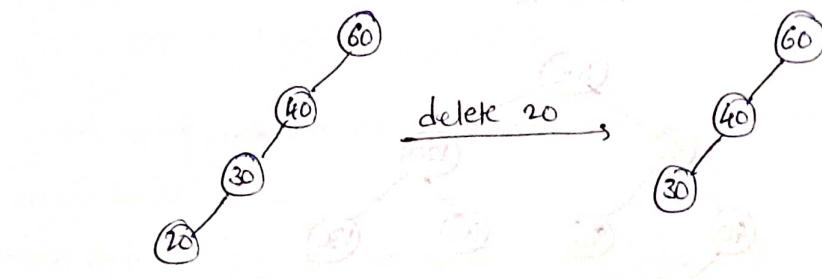
Step 2: Identify the parent pointer pointing to the node to be deleted and assign it to NULL.

pointer of parent of 40 pointing

resulting tree is



Eg 2: Delete 20 from below BST



$$\therefore \text{no of comparisons} = h + 1 \quad \begin{matrix} \nearrow \text{height} \\ \text{= } n \end{matrix}$$

$$= n \text{ (no of nodes)}$$

\hookrightarrow In the worst case.

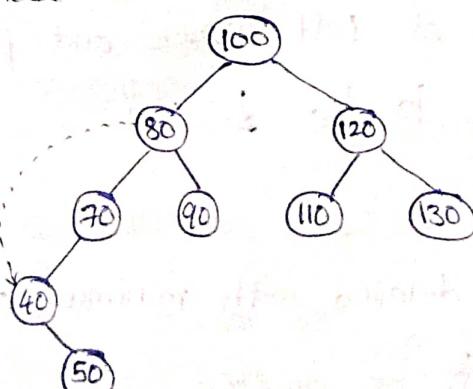
$$= O(n)$$

\therefore Time complexity for deleting node from BST $\begin{cases} O(\log n) & \text{--- complete BST} \\ O(n) & \text{--- one node per level} \end{cases}$

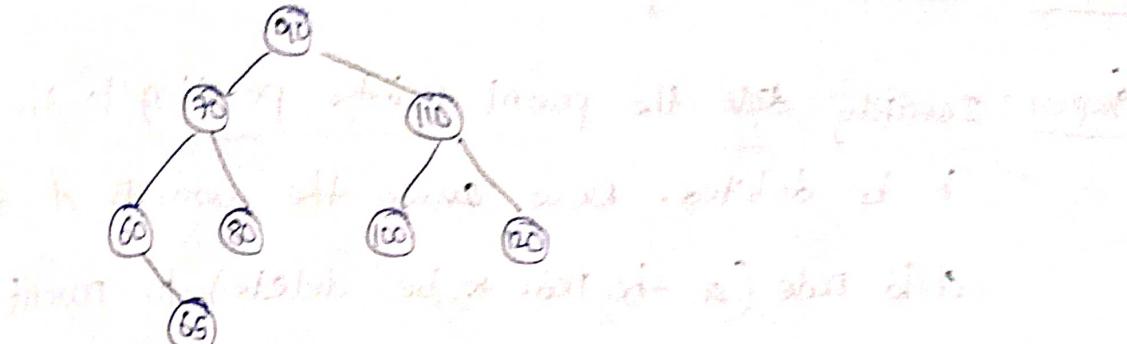
$$\therefore \text{worst case time complexity} = O(n)$$

Case (ii): Deleting a node which has only one subtree.

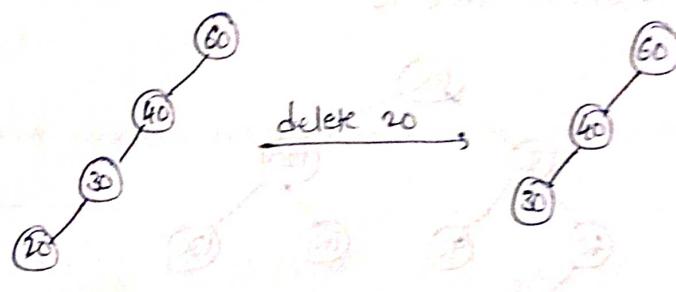
Eg: Delete node 70



resulting tree is



Eg 2: Delete 20 from below BST



$$\therefore \text{no of comparisons} = h+1$$

$$= n \text{ (no of nodes)}$$

\hookrightarrow In the worst case.

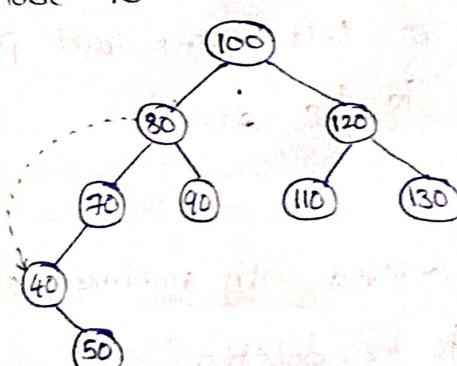
$$= O(n)$$

\therefore time complexity for deleting node from BST $= \begin{cases} O(\log n) & \text{--- complete BST} \\ O(n) & \text{--- one node per level} \end{cases}$

\therefore worst case time complexity $= O(n)$

Case (ii): Deleting a node which has only one subtree,

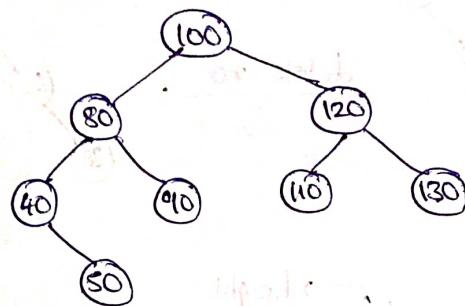
Eg: Delete node 70



Step (i): Search the key

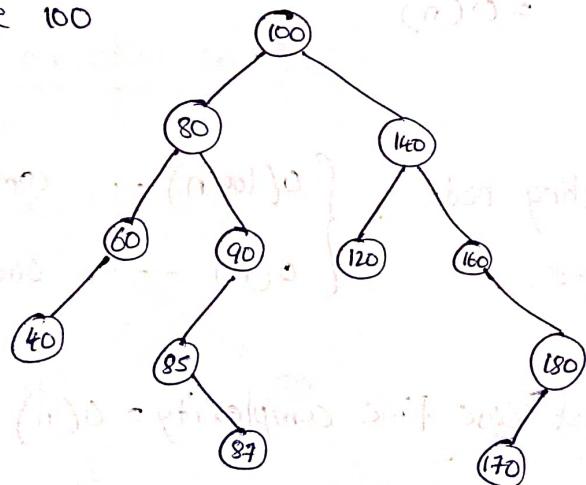
Step (ii): Identify where the parent pointer pointing to the node to be deleted. Now assign the address of the child node (of the node to be delete) to parent node's pointer.

∴ Resultant tree is



Case (iii): Deleting a node with both left & right subtrees.

Delete node 100



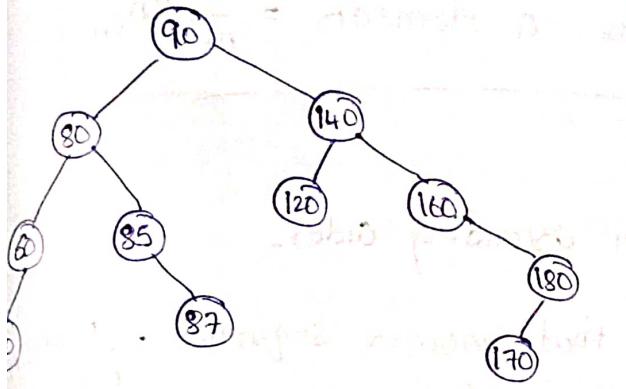
Method 1:

Find the maximum of Left subtree and place in the place of the node to be deleted.

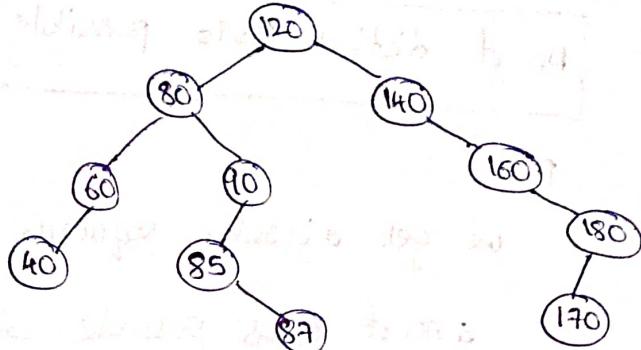
Method 2:

Replace the node to be deleted with minimum of right subtree of the node to be deleted.

Method 1 :



Method 2 :



33

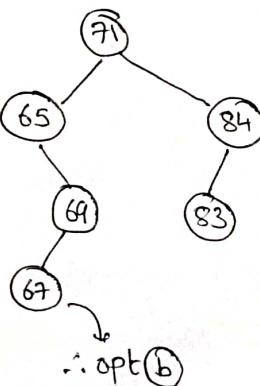
→ In both case (ii) & case (iii) also, worst case time complexity is $O(n)$

∴ time complexity for deletion from BST = $O(n)$

Q16) While inserting the elements 71, 65, 84, 69, 67, 83 in an empty BST in the sequence shown, the element in lowest level is

- a) 65 b) 67 c) 69 d) 83

Sol:



Q17) How many no of binary search trees are possible by arranging the following keys in tree given below.

100 200 300

Note:

No of distinct BSTs possible for n elements = $\frac{1}{n+1} 2^n C_n$

Proof:

We get σ in order sequence in ascending order.

No of trees possible with that inorder sequence = $\frac{1}{n+1} 2^n C_n$

No of binary trees possible with ' n ' unlabeled nodes = $\frac{1}{n+1} 2^n C_n$

No of binary trees possible with ' n ' distinct nodes = $\frac{n!}{n+1} 2^n C_n$

- Q7 Suppose that we have numbers b/w 1 and 100 in a BST, and want to search for the number ~~or~~ 55. Which of the following sequences cannot be the sequence of nodes examined?

- {10, 75, 64, 43, 60, 57, 55}
- {90, 12, 68, 34, 45, 55}
- {9, 85, 47, 68, 43, 57, 55}
- {79, 14, 72, 56, 16, 53, 55}

Sol:

Ans:

A sequence is valid iff at every element of the sequence, either all the elements right to it are smaller than the element or all are larger (from property of BST)

a) 10 75 64 43 60 57 55

all are larger than 10

75 64 43 60 57 55

all are smaller than 75

64 43 60 57 55

all are smaller than 64

43 60 57 55

all are larger

60 57 55

all are smaller

57 55

all are smaller

∴ valid sequence

c) 9 85 47 68 43 57 55

68 is larger than 47

43 is smaller than 47

∴ invalid sequence.

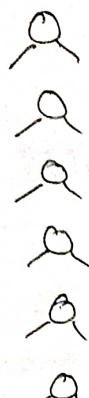
- Q18 Number of ways in which the numbers 1, 2, 3, 4, 5, 6, 7 can be inserted in an empty binary tree such that the height of resulting tree is 6 is _____ (height of tree with single node is 0)

Sol:

no of nodes = 7

height = 6

∴ nodes per level = 1



At each level we can choose one of the ~~one~~ pointers and point it to the node in next level.

∴ no of possible BSTs = $2^6 = 64$

15/09/20

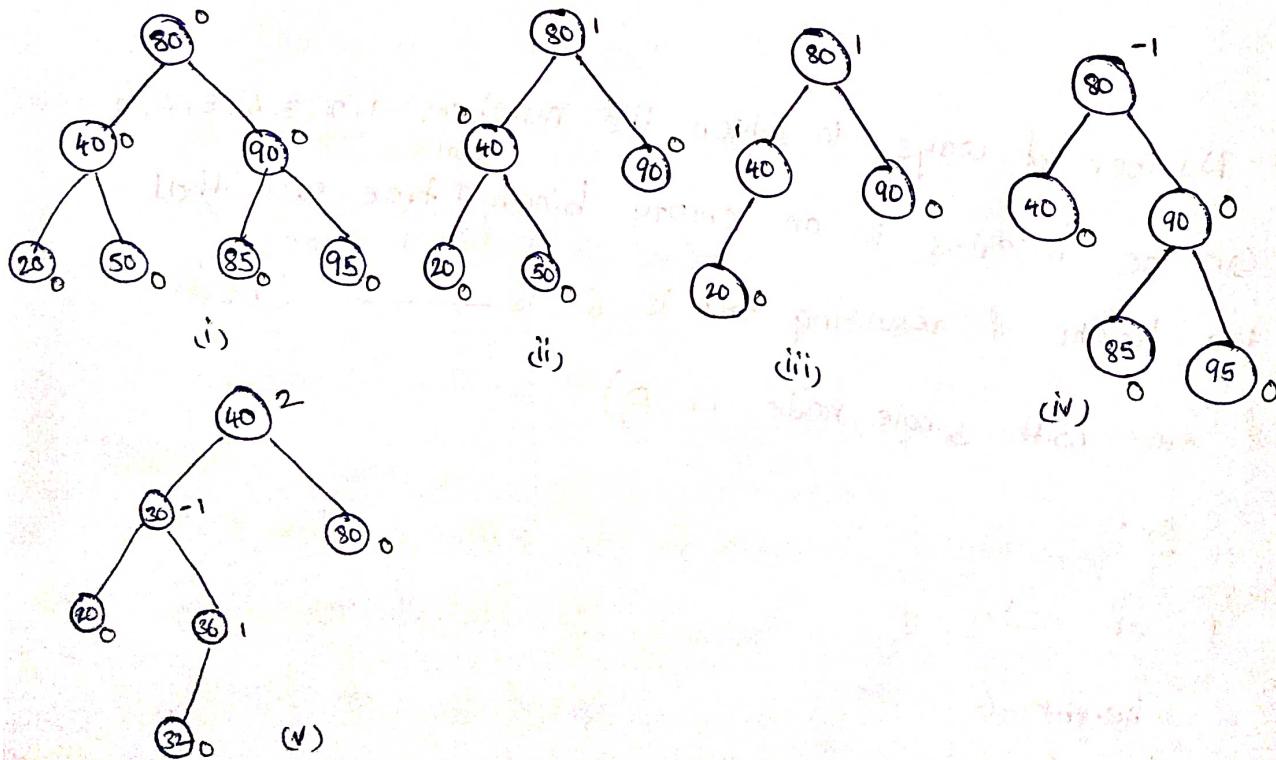
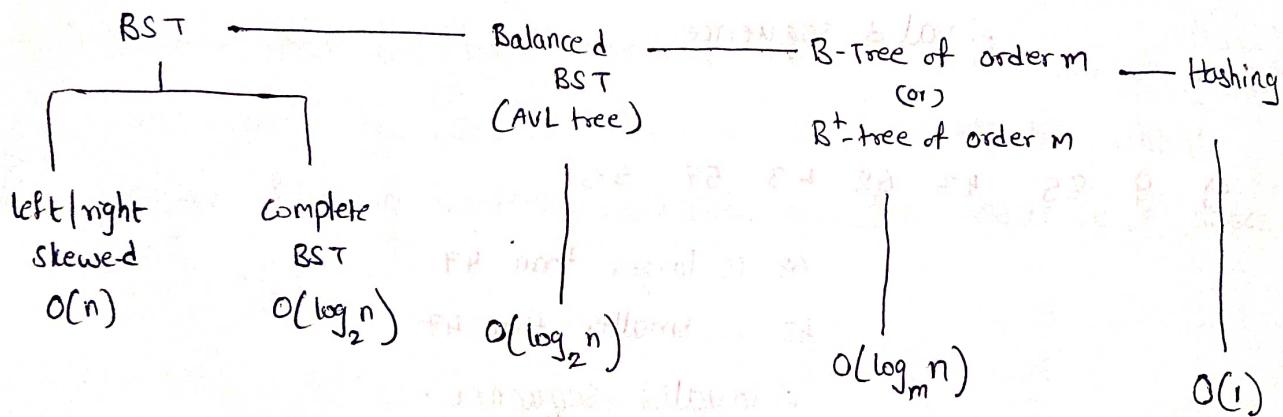
AVL - Tree : (Balanced BST)

[Adelson - Velski - Landis]

→ If BST is left or right skewed then search time is $O(n)$

→ If BST is complete then search time is $O(\log_2 n)$

Search time



Balancing

Balancing factor: Height of left subtree - Height of right subtree

Defn of AVL tree: A BST is called AVL tree if balancing factor is ≤ 1 or ≥ -1 of every node is -1 (or) 0 (or) 1

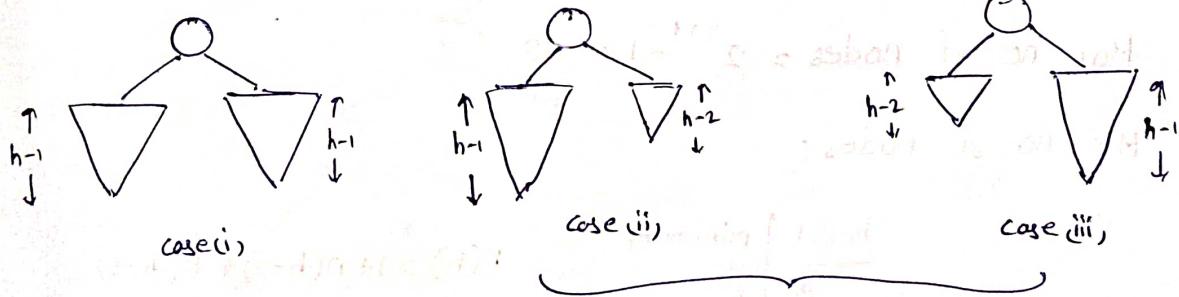
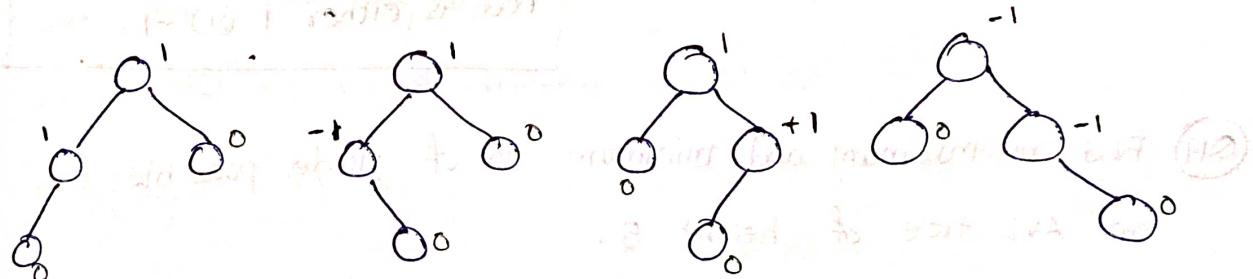
Eg: (i) (ii) (iii) (iv) are AVL trees
(v) is not an AVL tree

Note:

Max no of nodes in an AVL tree of height ' h ' = $2^{h+1} - 1$ (i.e., FBT)

Min no of nodes possible in an AVL tree of height ' h '

Eg: for $h=2$, AVL tree with min no of nodes



Min no of nodes are possible in case (ii) & case (iii)

Let $n(h)$ denote minimum no of nodes possible in an AVL tree of height ' h '.

from case (ii) & case (iii),

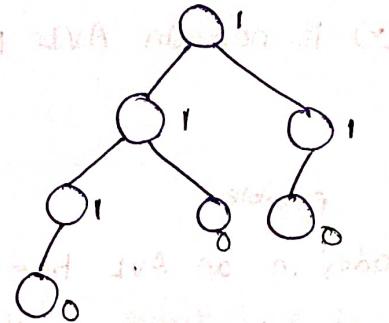
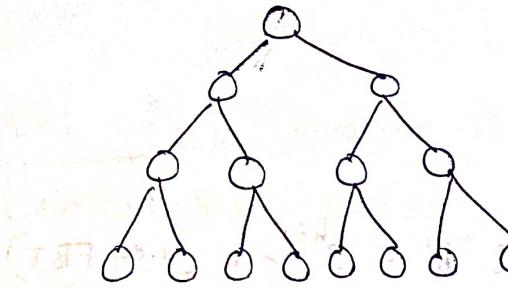
$$n(h) = \underbrace{1}_{\text{root}} + n(h-1) + n(h-2)$$

$$n(h) = \begin{cases} 1 + n(h-1) + n(h-2) & \text{if } h \geq 2 \\ 1 & \text{if } h=1 \\ 2 & \text{if } h=0 \end{cases}$$

For height $h=3$

Max no of nodes = 15

Min no of nodes = 7



In the case of min no of nodes

balancing factor of every internal node is either 1 (or) -1.

- (Q19) Find maximum and minimum no of nodes possible for an AVL tree of height 5.

Sol:

$$\text{Max no of nodes} = 2^{5+1} - 1 = 63$$

Min no of nodes:

height	minimum
0	1
1	2
2	4
3	7
4	12
5	20

$$n(h) = 1 + n(h-1) + n(h-2)$$

$$\begin{aligned} n(5) &= 1 + n(4) + n(3) \\ &= 1 + 2 + 1 \\ &= 4 \end{aligned}$$

(Q20) what is max height of any AVL tree with 7 nodes (height of leaf nodes is 0)

Sol:

min no of nodes possible for an AVL tree

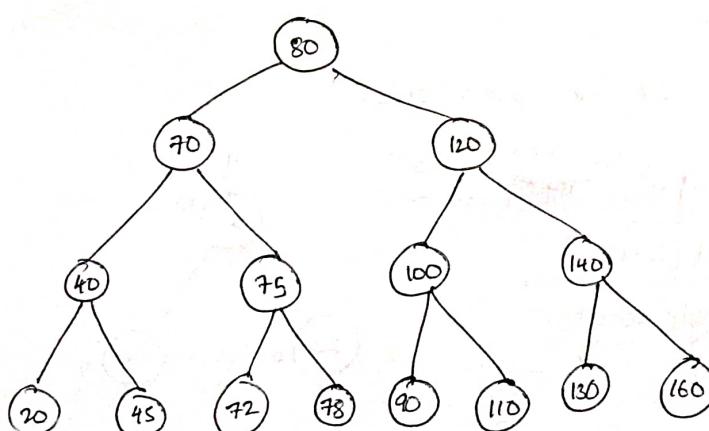
height	min no of nodes
0	1
1	2
2	4
3	7
4	12

∴ Max possible height = 3

(Q21) In a balanced BST with n elements, what is the worst case time complexity of reporting all elements in range [a,b]? Assume that the no of elements is k?

- a) $\Theta(\log n)$ b) $\Theta(\log n + k)$ c) $\Theta(k \log n)$ d) $\Theta(n \log k)$

Sol:



Max height of BST with n elements = $\Theta(\log n)$

let a=20 b=78

so the interval is $[a, b] = [20, 78]$

Step 1: to report the elements, we first need to search 20

case (iii) :
Insert

This searching takes $O(\log_2 n)$

Silly we search $b=78$ in $O(\log_2 n)$

so to construct $[20, 78]$ time = $O(\log_2 n)$

Now we need to report all the elements in $[20, 78]$

No of elements to be reported, $k = 7$

now we apply inorder traversal on elements $[20, 78]$

$[a, b]$ has k elements

\therefore Inorder traversal on k elements takes $O(k)$

\therefore total time complexity

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

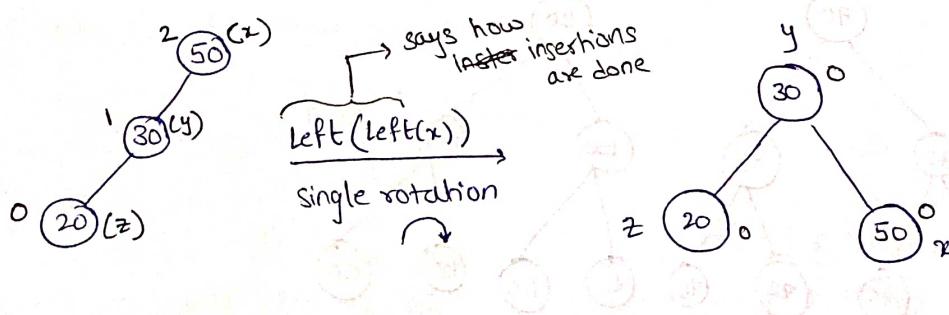
\therefore opt(b)

16/09/20

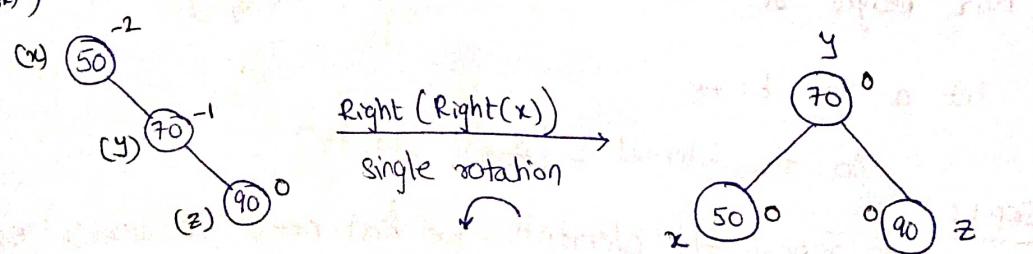
Construction of AVL tree by inserting keys in the given order:

Case(i) (LL(+))

Insert 50, 30, 20 in given order.



Case(ii): Insert 50, 70, 90 in given order
(RRW)



case(iv) :
Insert

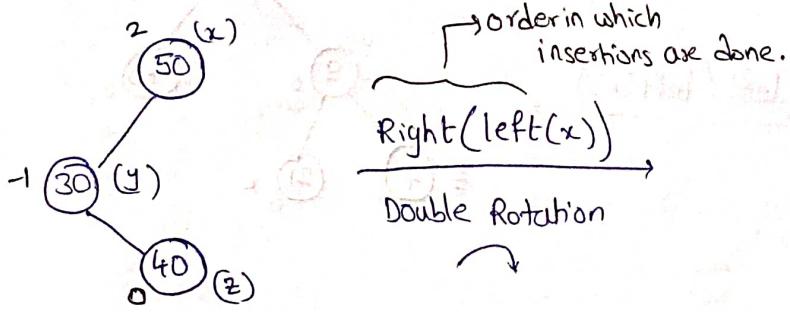
Insert

In
21

P/29

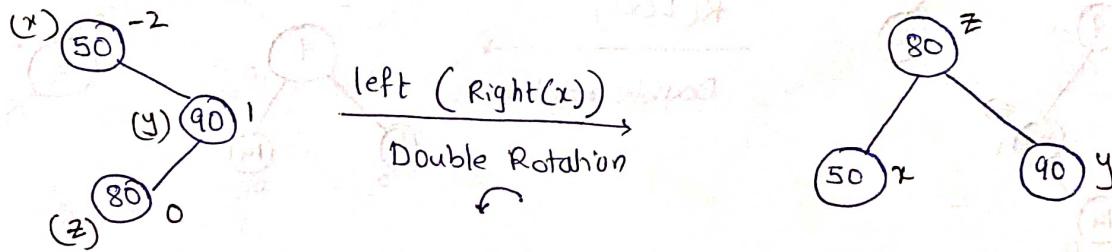
Case (iii) : $(R-L(x))$

Insert 50, 30, 40



Case (iv) $(L-R(x))$

Insert 50, 90, 80



Insertion :

P/29
21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7

Insertion :

21:



26:



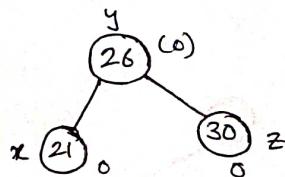
30:

21 -2 (x)

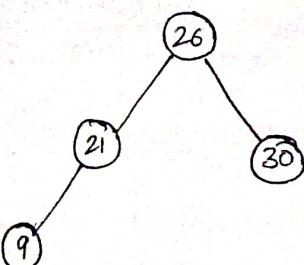
26 -1 (y)

30 0 (z)

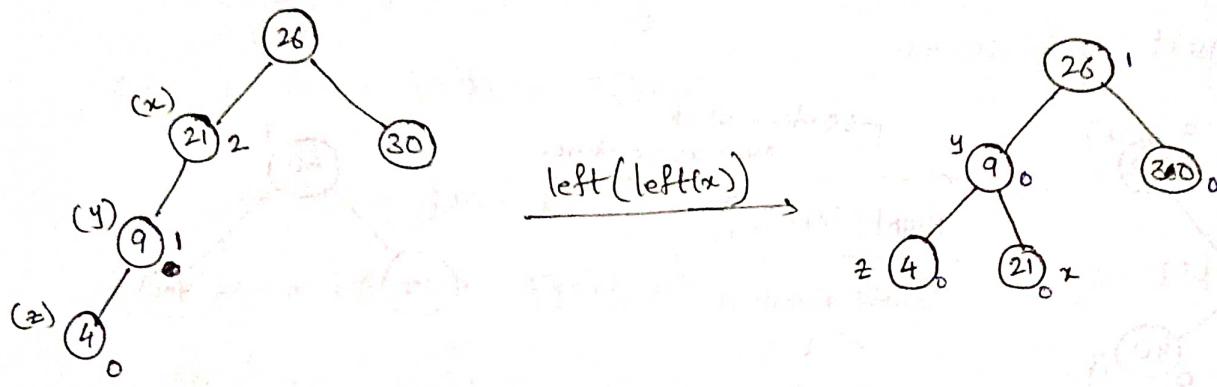
$R(RC(x)) \rightarrow$



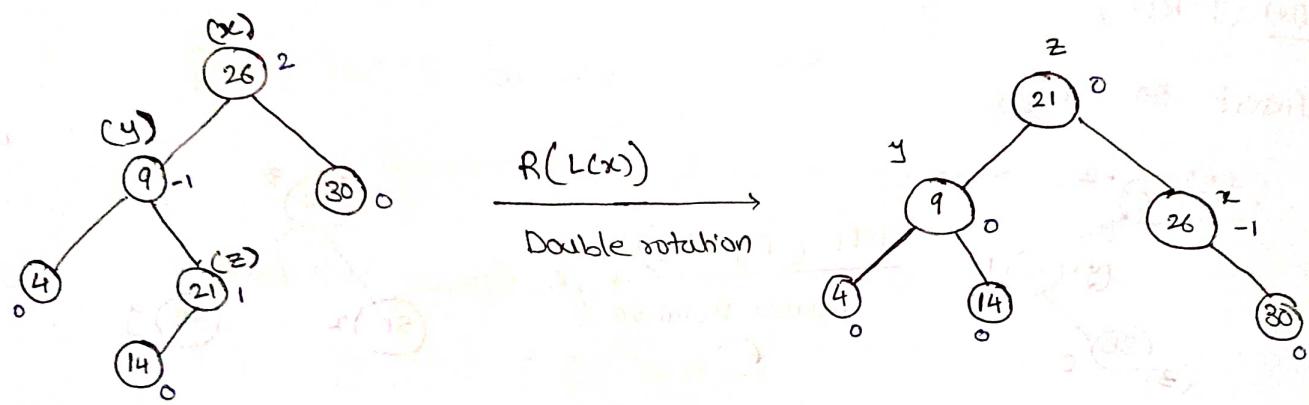
9:



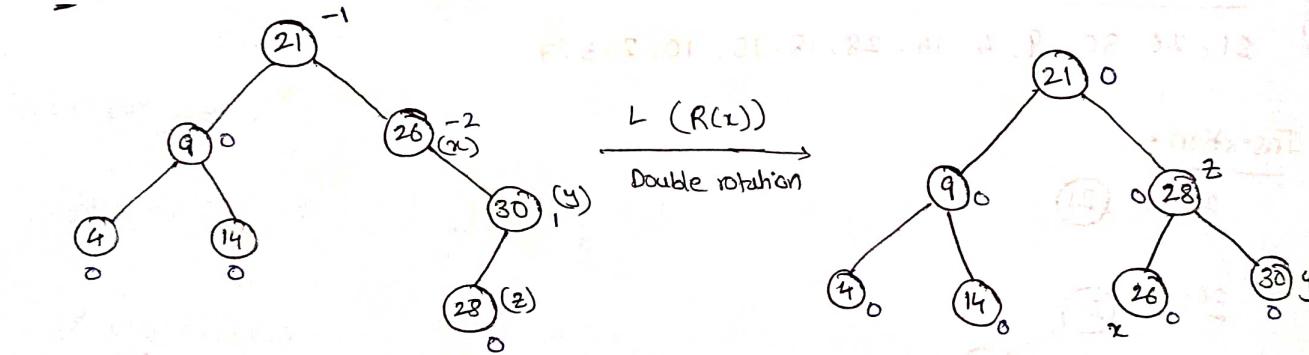
4:



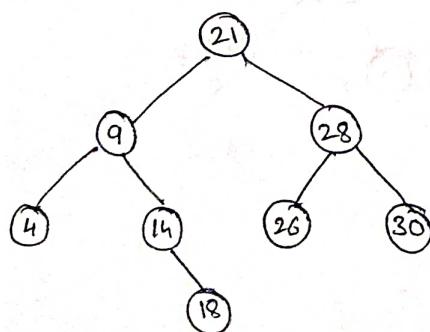
14:



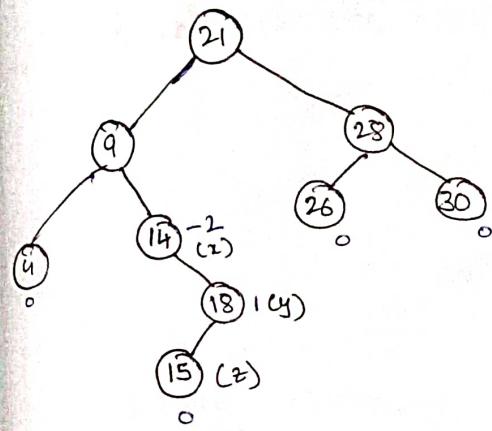
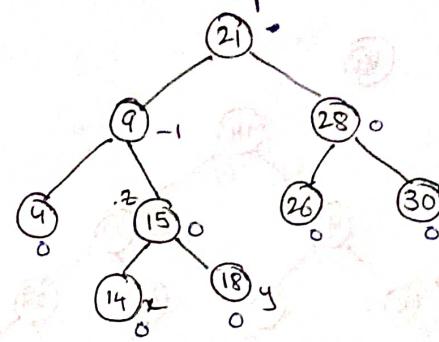
28:



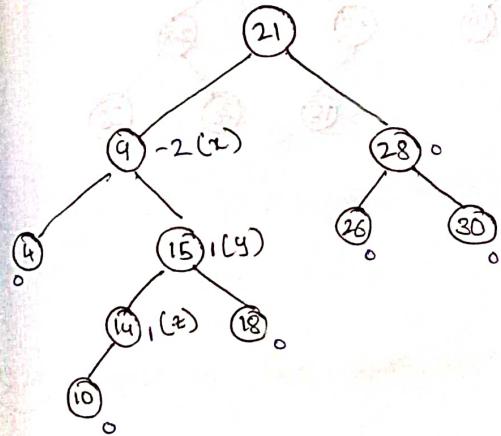
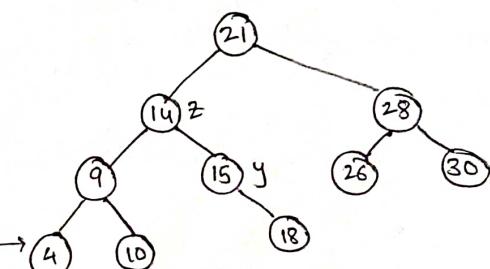
18:



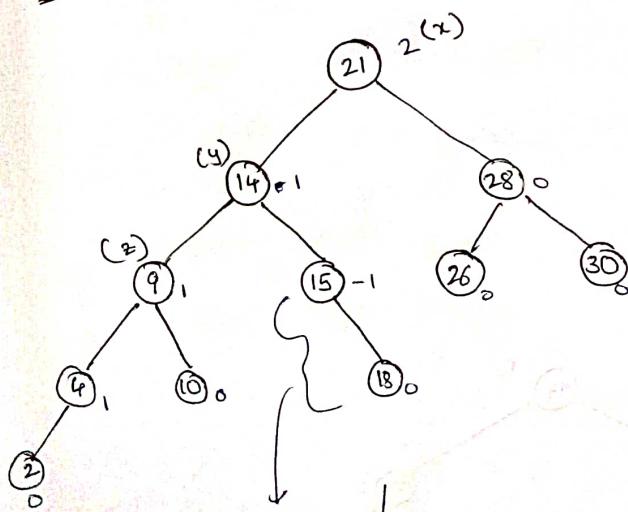
15 :

 $L(R(z))$ 

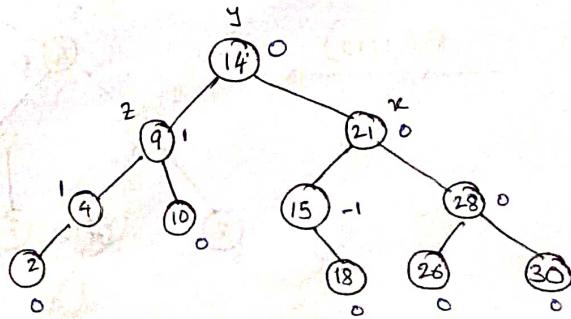
16 :

 $L(R(z))$ 

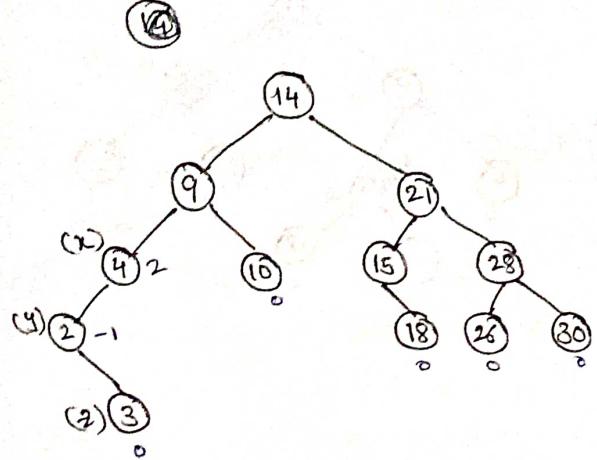
2 :

 $LL(z)$

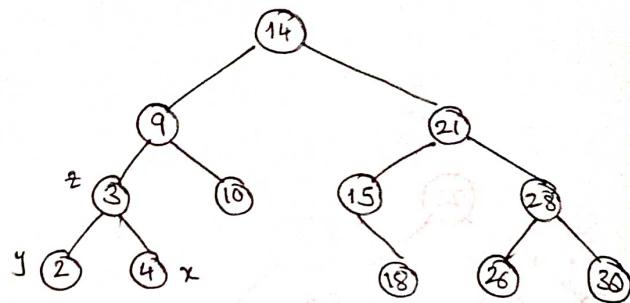
Here 15 & 18
are right of
14 and left of 21



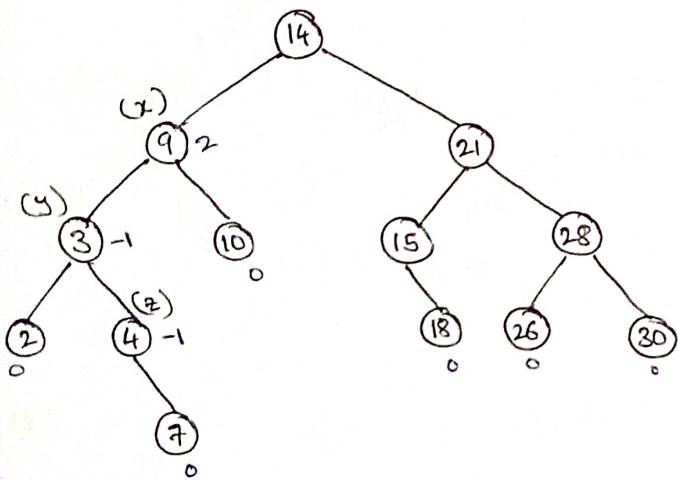
3:



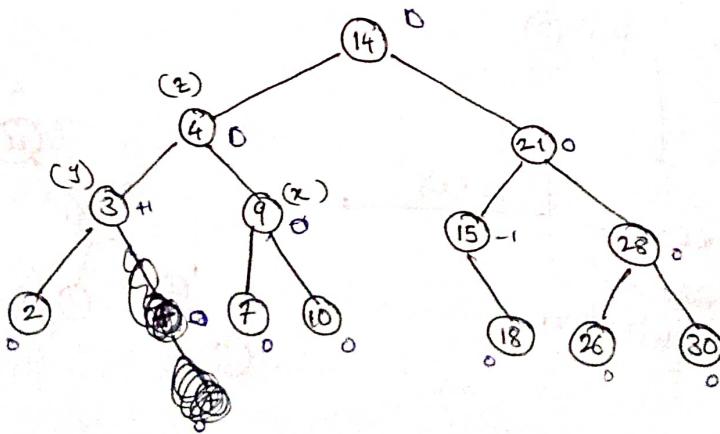
$R(L(x))$



7:



$R(L(x))$



Analysis:

→ No of comparisons performed in worst case for insertion

$$= \text{height of AVL tree} + 1$$

$$= \log_2 n + 1$$

$$\approx O(\log n)$$

→ total time complexity for constructing AVL with n^2 elements is

$$\log_1 + \log_2 + \log_3 + \dots + \log n$$

$$= \log(1 \cdot 2 \cdot 3 \cdots n)$$

$$= \log n!$$

$$= O(\log n^n)$$

$$= O(n \log n)$$

Q22 what is the ~~worst~~ worst case time complexity of inserting n^2 elements into an AVL-tree with n elements initially?

- a) $O(n^2)$ b) $O(n^2 \log n)$ c) $O(n^4)$ d) $O(n^3)$

Sol:

inserting one element --- $O(\log n)$

inserting n^2 elements --- $O(n^2 \log n)$

since we are talking about worst case we consider

Method 2:

$$\log n + \log(n+1) + \log(n+2) + \dots + \log(n+n^2)$$

$$= \log(n \cdot (n+1) \cdot (n+2) \cdots (n+n^2)) = \log\left(\frac{1 \cdot 2 \cdot 3 \cdots (n+n^2)}{1 \cdot 2 \cdot 3 \cdots (n-1)}\right)$$

$$= \log(n+n^2)! - \log(n-1)!$$

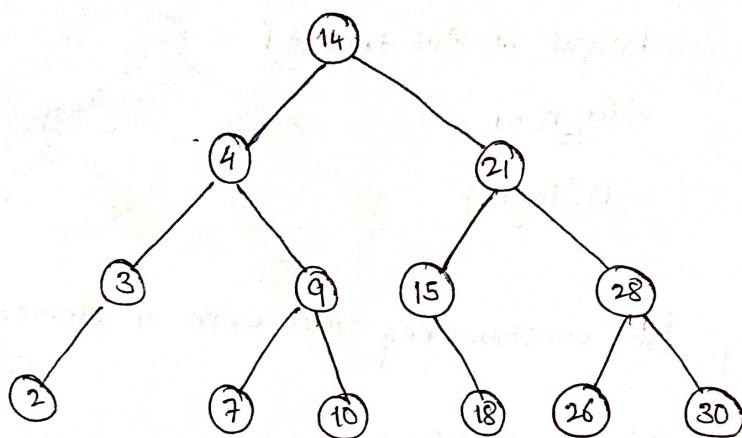
$$= O(\log(n+n^2)!) = O\left(\log(n+n^2)^{(n+n^2)}\right) = O(n^2 \log n)$$

17/09/20

P/30

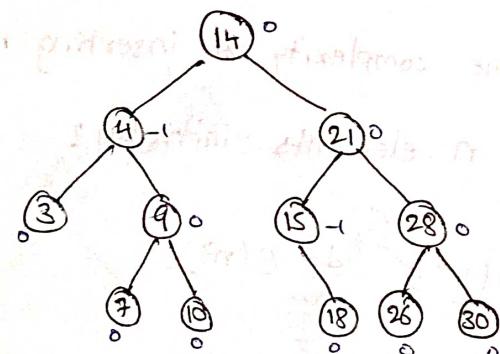
Deletion on AVL Tree

Delete 2, 3, 10, 18, 4, 9, 14, 7, 15



Here we perform deletion in the same way as we did for BST.
But after every deletion we check whether AVL tree property is satisfied or not.

2: Deleting leaf node:



Step 1:

→ Search the node $O(\log n)$

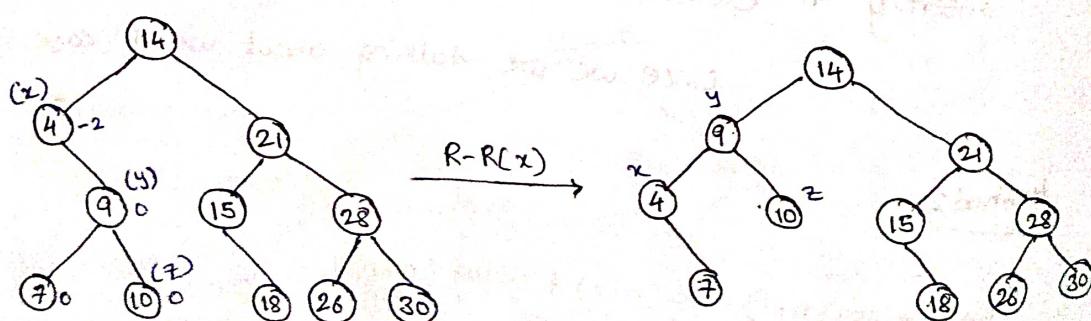
→ Identify the deleted node's parent pointer and assign it to NULL.

→ Now if the AVL property is satisfied then stop the process.

Otherwise perform necessary rotations to convert it into AVL tree.

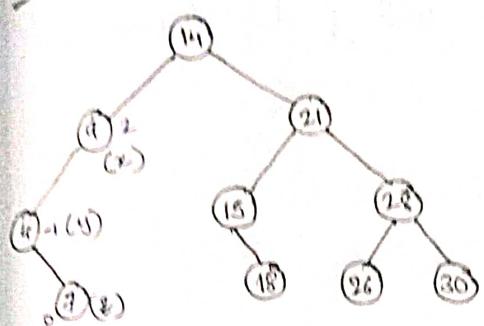
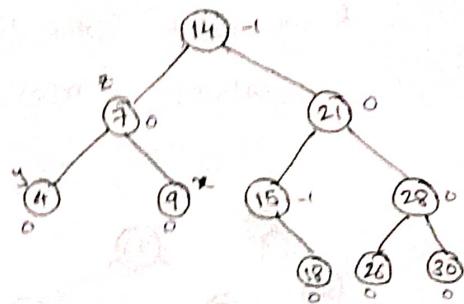
~~then~~

3:

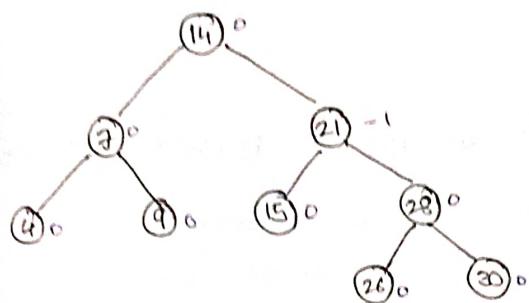


Here we can choose either
7 or 10 as z.

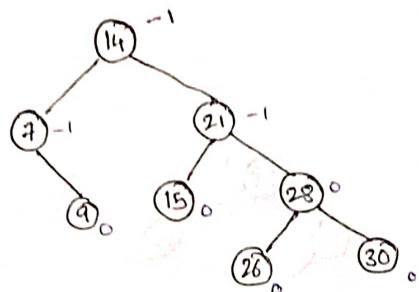
10:

 $R(L(1)) \rightarrow$ 

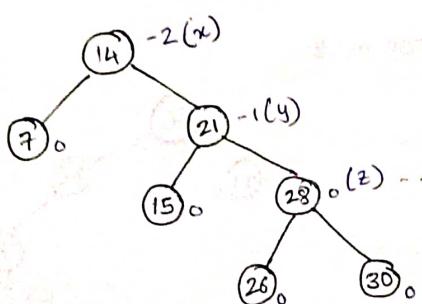
15:



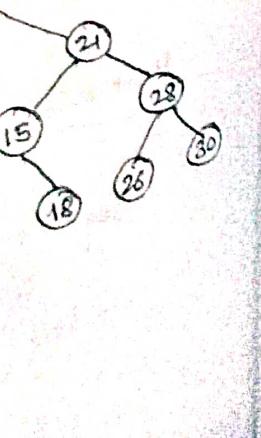
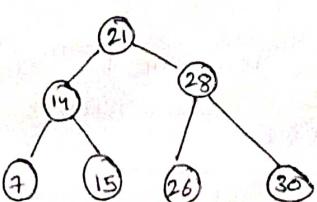
16:



17:



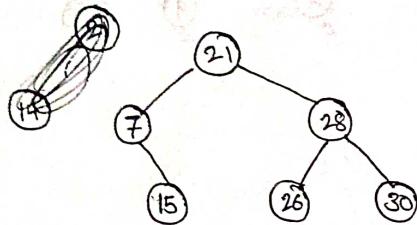
..... This is chosen as 2 because we have longest path in this direction

 $R-R(\star) \rightarrow$ 

Deleting node with 2 children

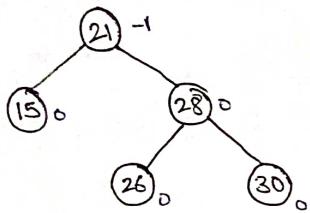
14:

A we delete same like we did for BST and perform appropriate rotations if needed.

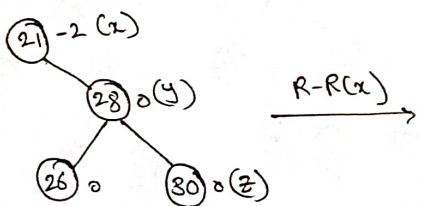


15: (Deleting node with 1 child)

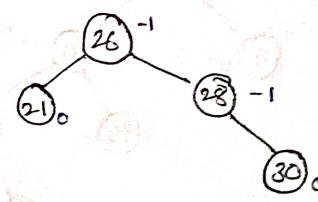
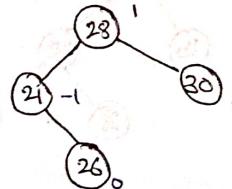
B Same as we did for BST and we perform necessary rotations.



15:



Here we can choose either 26 or 30 as z. If 26 is chosen as z



Analysis:

Deleting a node is done by search it first and delete and perform necessary rotations.

∴ time complexity for deletion from AVL tree = $O(\log n)$

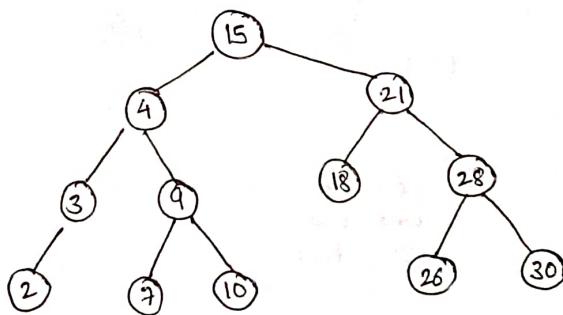
Deleting n nodes = $O(n \log n + \log(n-1) + \dots + \log 2 + \log 1) = O(n \log n)$

height	min no of nodes
0	1
1	2
2	4
3	7
4	12
5	20
6	33
7	54
8	88

P/27 Deleting 14

we need replace with successor 15

and 15 is replaced with its successor i.e. 18



∴ root node : 15

P/26 In a BST for every node, nodes in LST are lesser and nodes in RST are greater

a) 5 3 1 2 4 7 8 6

↑
5 is root

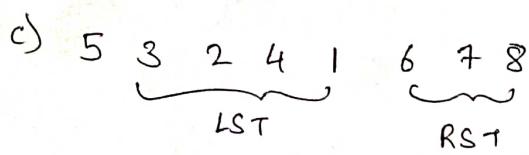
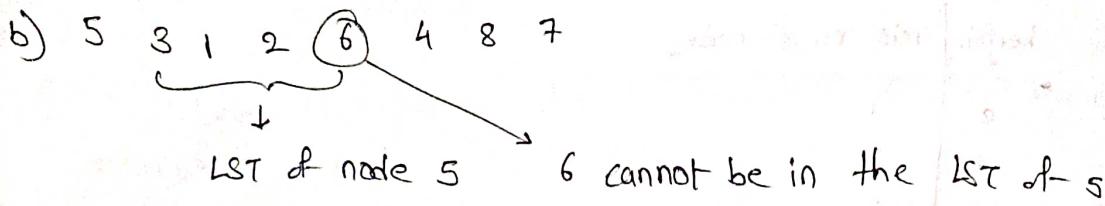
∴ next 4 elements must be among 1 to 4
and next 3 elements should be 6 to 7

3 1 2 4 ✓

7 8 6 ✓

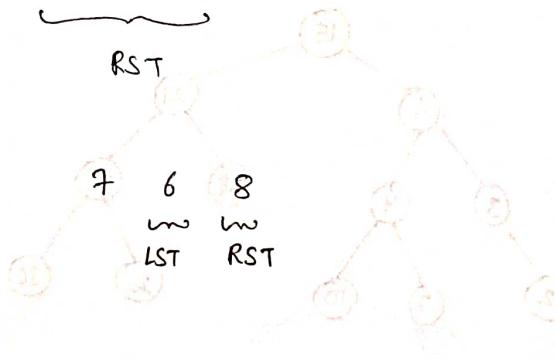
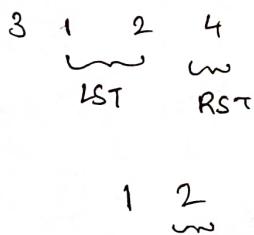
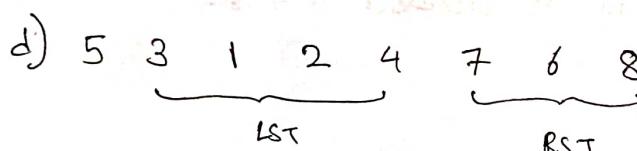
left subtree has 8

∴ α



3 2 4 1
 \uparrow \curvearrowright \curvearrowright
LST of 3 RST of 3

$\therefore \text{d}$



$\therefore \text{opt } \textcircled{2}$

P/25

RC: Right child

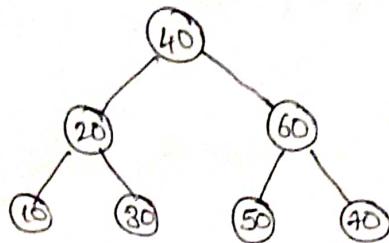
if(! t \rightarrow RC)

\hookrightarrow if we have right child then condition fails
else we return data

So this program keeps traversing towards right and finally prints when there is no right child.

$\therefore 30$

P/21



∴ 4

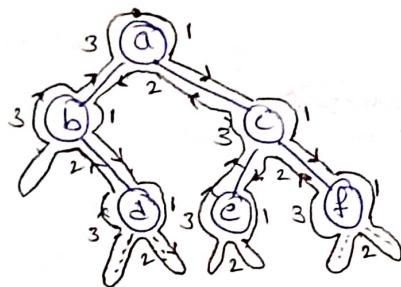
19/09/20

P/01

left, right, root

∴ postorder

Converse of postorder traversals:



Converse of postorder: f e c d b a (3's)

Converse of preorder: a c f e b d (1's)

Converse of inorder: f c e a d b (2's)

→ C-preorder(struct node* root)

```

{
    if (!root) return;
    printf("%c", root->data);
    C-preorder(root->right_child);
    C-preorder(root->left_child);
}
  
```

→ C-inorder (struct node* root)

```

{
    if (!root) return;
    C-inorder(root->right_child);
    cout printf("%c", root->data);
    C-inorder(root->left_child);
}
  
```

\rightarrow c-postorder (struct node* root)

{ if (!root)

 return;

 c-postorder (root -> right-child);

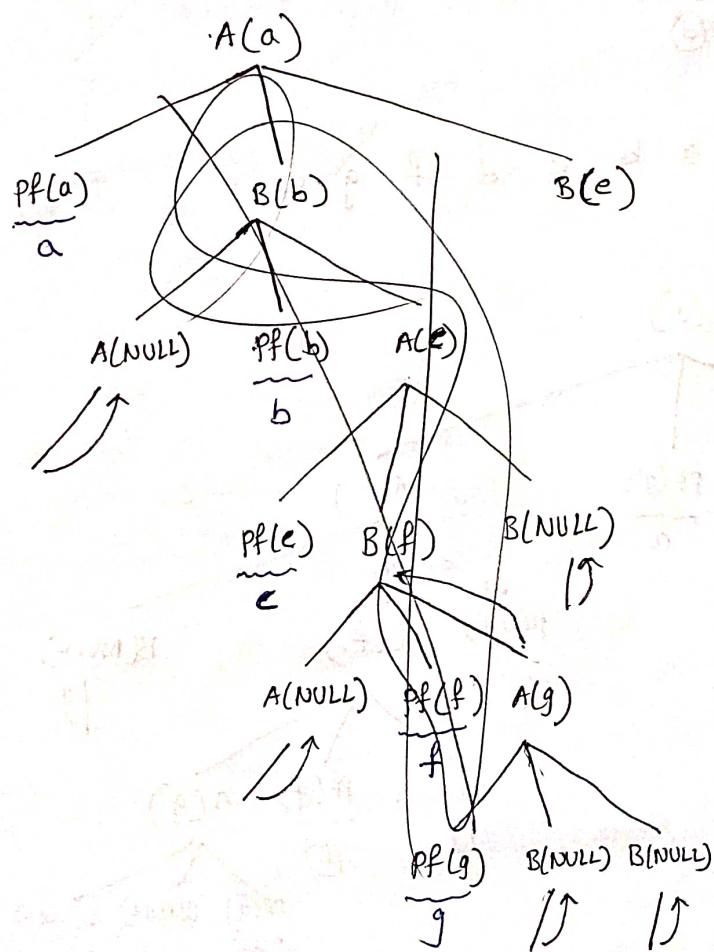
 c-postorder (root -> left-child);

 printf ("root->data");

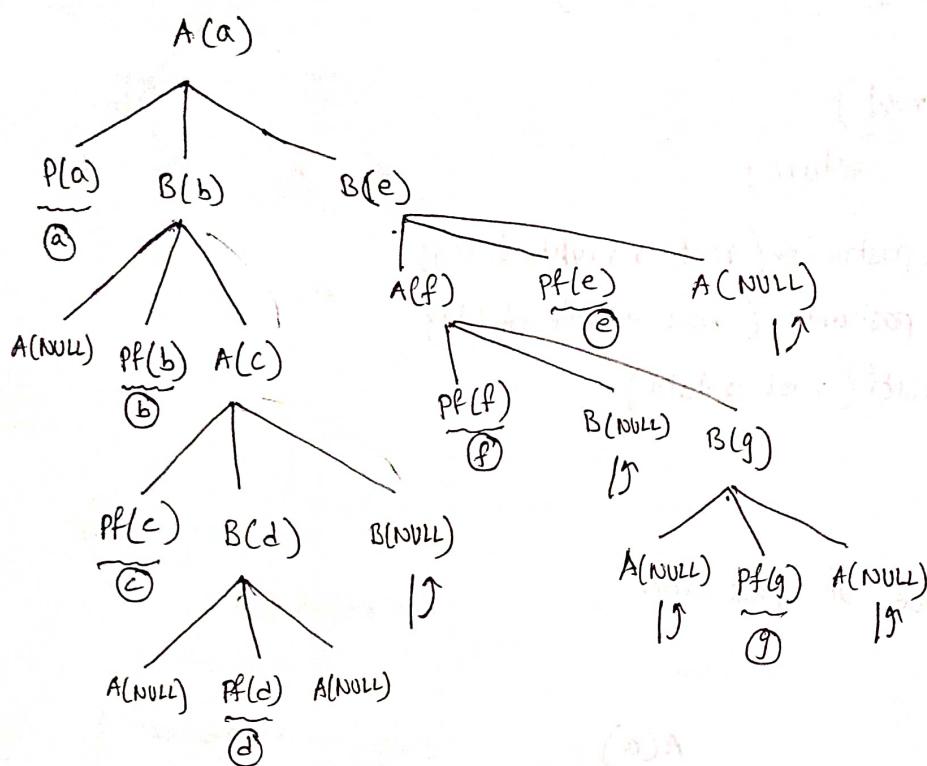
}

(P/2) Converse of post order

(P/3)

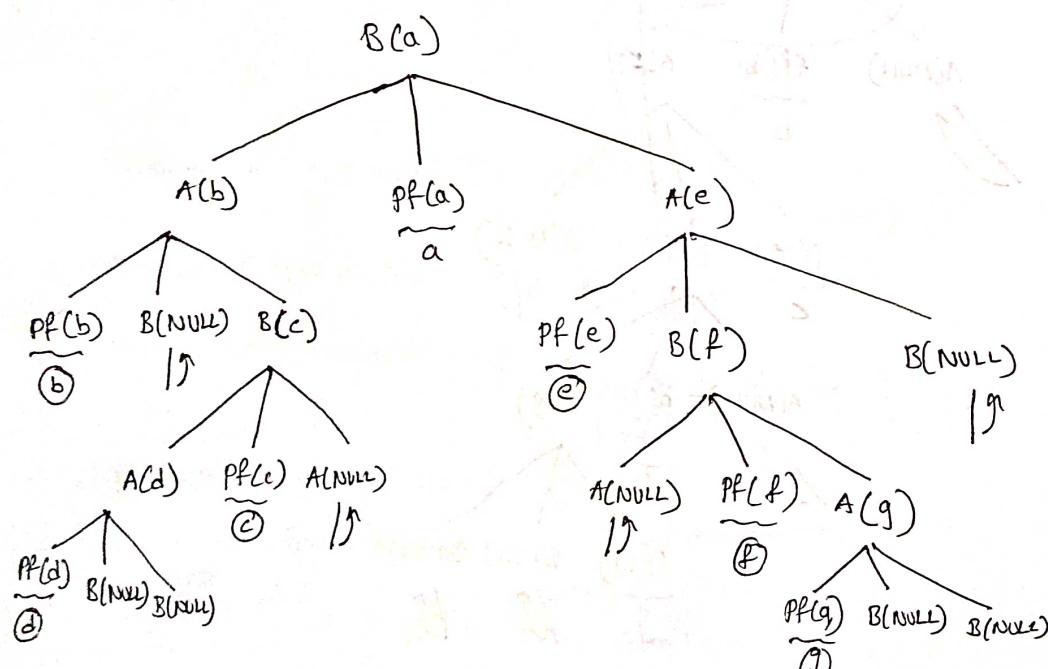


P/3



$\therefore A(a) : a \ b \ c \ d \ f \ g \ e$

P/6



$\therefore B(a) : b \ d \ c \ a \ e \ f \ g$

Note :

→ For a
we can

→ Only postor

B(t)
A(c)

P/5

$\frac{1}{n+1}$

Ino

Preo

(P/4) $B(t) = A(e) \cup A(b)$

$A(e) = e \ B(f)$

$e \ A(g) \ f$

$e \ g \ f$

$A(b) = b \ B(c)$

$b \ c \ A(d)$

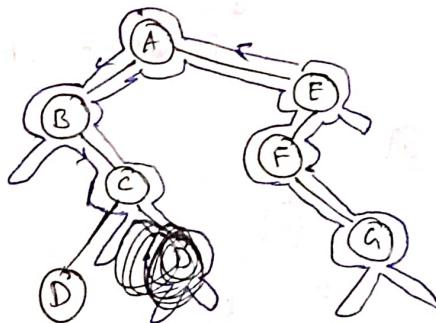
$b \ c \ d$

$\therefore B(t) = e \ g \ f \ a \ b \ c \ d$

(P/5) $\frac{1}{n+1} 2nC_n = \frac{1}{4} 6C_3 = \frac{1}{4} \frac{6 \times 5 \times 4}{3 \times 2 \times 1} = 5$

(P/6) Inorder: A B D C A F G E

preorder: A B C D E F G
 $\uparrow \quad \uparrow \quad \cdot \quad \cdot \quad \uparrow \quad \underline{\quad}$



postorder: D C B G F E A

Note:

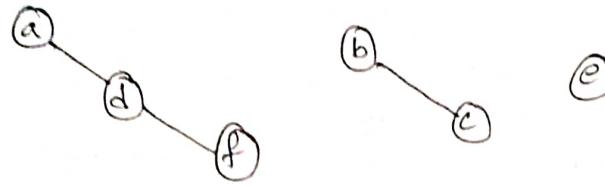
→ For a given preorder + right child pointer & terminal nodes we can construct a unique binary tree.

→ Similarly postorder + left child pointer + terminal nodes \Rightarrow unique binary tree

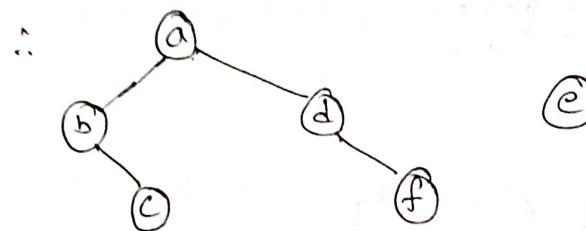
* *
P/7

TAH = 1 \Rightarrow leaf nodes

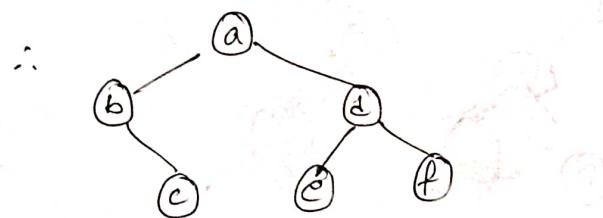
we have following parts of tree



If $b \rightarrow c$ is present in right subtree then preorder sequence cannot start with $a \rightarrow a, b$.



\therefore preorder: a b c d e f
 e is left child of d



\therefore no of nodes in RST = 3 (d, e, f)

P/8

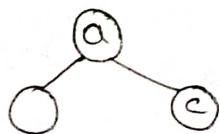
post order: d e b f g c a

degree: 0 0 2 0 0 2 2

a is root & deg of a is 2



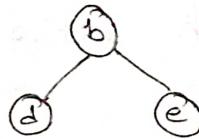
a has right child & preceding node of a in postorder is c. \therefore Right child of a is c.



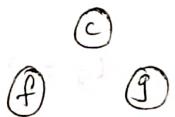
the starting 3 nodes of post order are d, e, b

d must be left most node of tree.

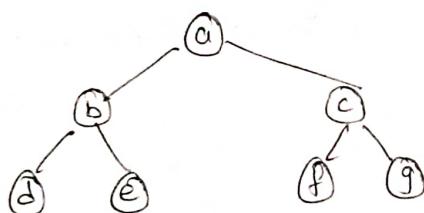
\therefore e is leaf & b has degree 2, the possible structure is



Similarly for f, g, c we have



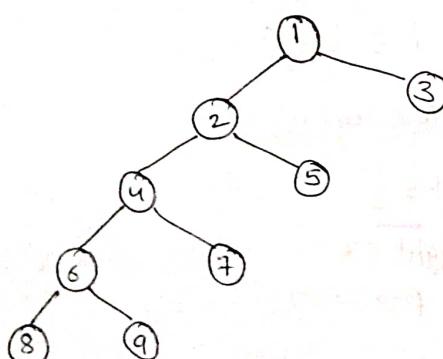
\therefore the binary tree is



\therefore no of nodes in the LST = 3

(P/a) In. : 8 6 9 4 7 2 5 1 3
post : 8 9 6 7 4 5 2 3 1

\uparrow_{root}



\therefore height = 4

P/10

a) $\text{total nodes} = \text{internal} + \text{leaves}$

$$2^*i_1 + 1 = i_1 + 20 \quad (i_1 \rightarrow \text{internal nodes})$$

$$\cancel{2i_1} = \cancel{i_1} + 19$$

~~∴ need to solve lesser than 19~~

Let i_1 be no of nodes with 1 child

i_2 be no of nodes with 2 children

$$i_1 + 2^*i_2 + 1 = i_1 + i_2 + 20$$

$$i_2 = 19$$

∴ no of nodes having 2 children = 19

b) $i_1 + 2i_2 + 1 = i_1 + 200$

$$i_2 = 199$$

P/11

Min height \Rightarrow complete binary tree

If n is no of nodes in CBT

$$2^h \leq n \leq 2^{h+1} - 1$$

$$2^h \leq n \quad n+1 \leq 2^{h+1}$$

$$h \leq \log_2 n \Rightarrow h+1 \geq \log_2(n+1)$$

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

$$h \geq \log_2(16) - 1$$

$$h \geq 3$$

Max height --- tree ~~one node per level~~ min height = 3

$$\Rightarrow \text{max height} = 14$$

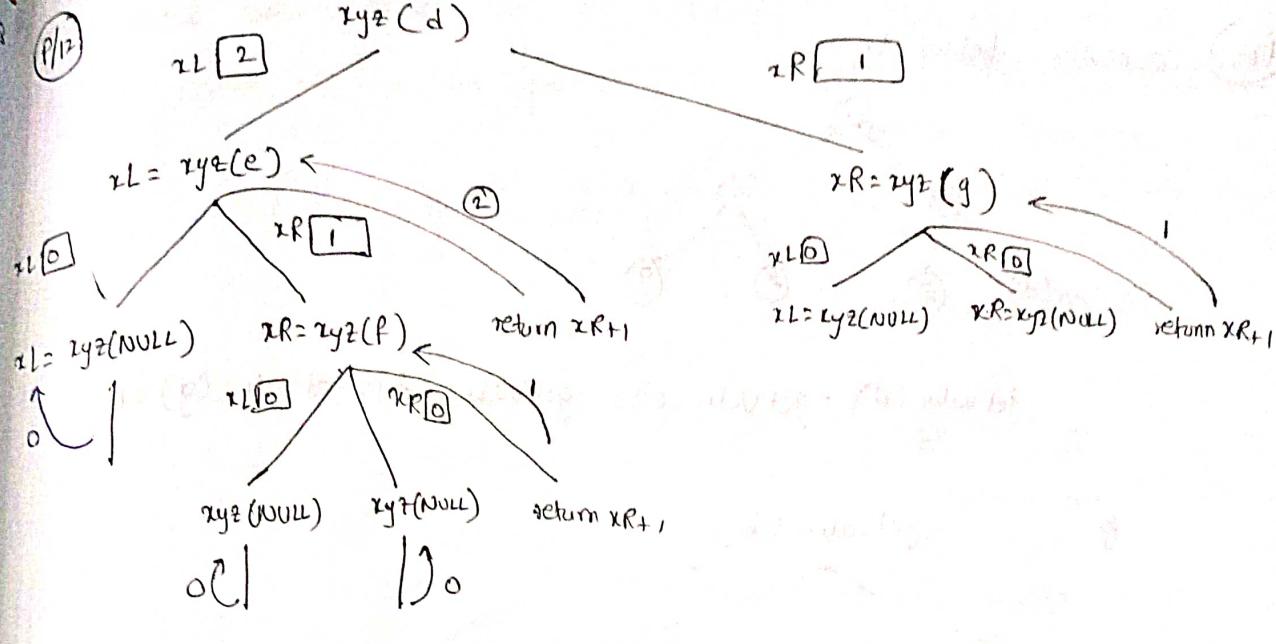
P/12

P/10

$$x_1 = x_2$$

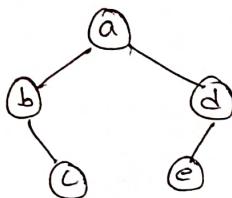
↑

P/13



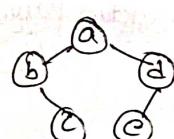
$$\therefore xL = 2 \quad xR = 1$$

P/13 From the code, swapping will be done in nodes in the order of postorder.

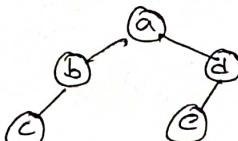


Postorder: c b e d a

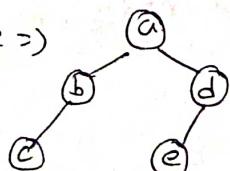
Swap on c =>



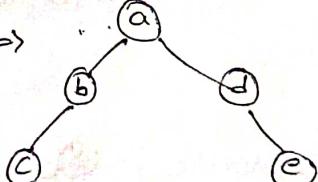
swap on b =>



Swap on e =>



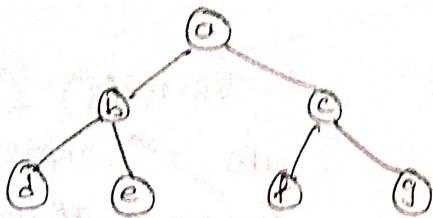
Swap on d =>



This is the tree before last swap.

P/14

Consider below tree



$$\text{getVal}(d) = \text{getVal}(e) = \text{getVal}(f) = \text{getVal}(g) = 1$$

Q

$\text{getVal}(b)$

$$\text{value} = \text{value} + \text{getVal}(d) + \text{getVal}(e)$$

1 1

return value;

[2]

$$\therefore \text{value} = 2$$

$$\therefore \text{getVal}(b) = 2$$

Similarly $\text{getVal}(c) = 2$

Now

$\text{getVal}(a)$

$$\text{value} = \text{value} + \text{getVal}(b) + \text{getVal}(c)$$

2 1 1

return value;

[4]

$$\Rightarrow \text{value} = 4$$

$\therefore \text{getVal}(a) = 4 = \text{no of leaf nodes} = \text{no of no deg without right sibling}$

Consider



$$\text{getVal}(b) = 1$$

$$\text{getVal}(a) = \text{value} + \text{getVal}(\text{NULL}) + \text{getVal}(b) = 0 + 0 + 1 = 1$$

= ~~no of~~ no of leaf node

$\therefore \text{opt}(d)$

\neq no of nodes without right sibling

* *
P/15

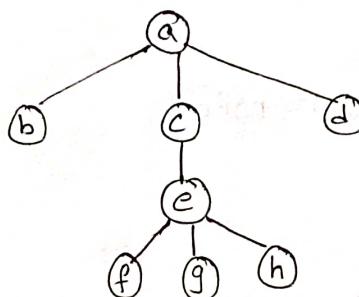
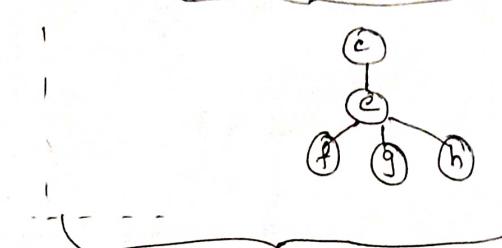
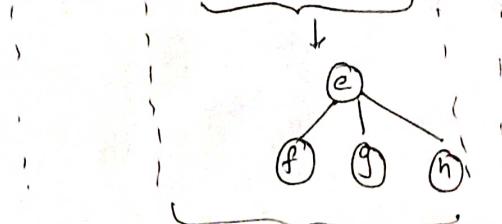
node
for

* *

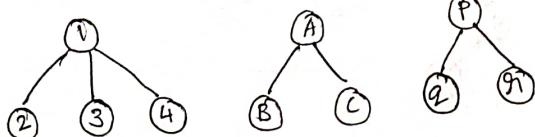
P/16
* *

* *
P/15 nodes ~~out~~ outside the parenthesis is considered parent
for nodes in the parenthesis

$a(b, c(e(f, g, h)), d)$



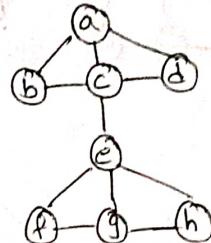
* *
P/16 * *
Converting forest into binary tree:



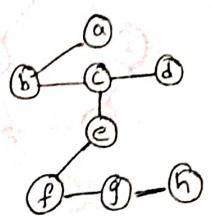
Leftmost child right sibling representation

Converting into binary tree

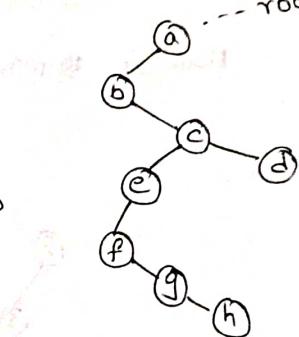
connect every child to its right sibling



keep only leftmost link



This is redrawn as

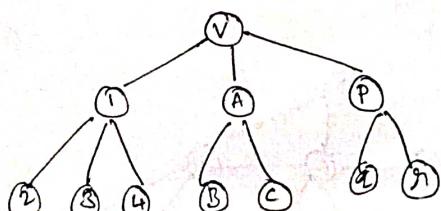


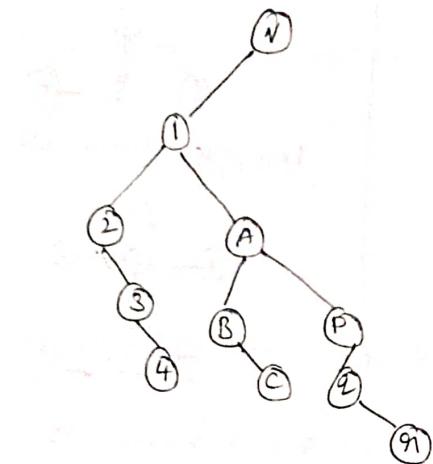
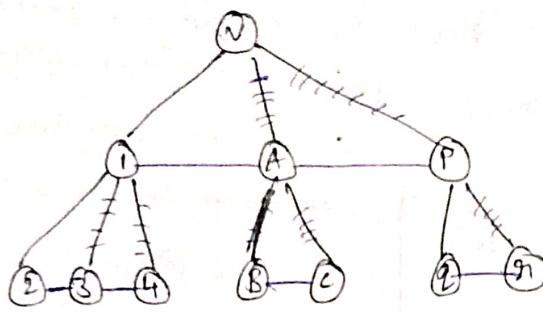
∴ Height = 3

∴ Height = 6

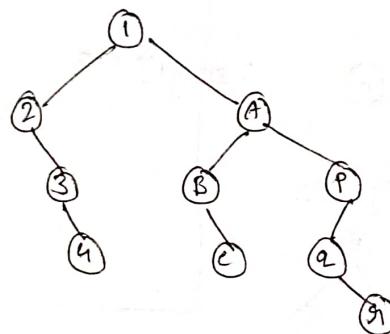
Rule for converting general tree into binary tree
→ Using leftmost child right sibling representation we convert into binary tree.

let 'v' be a virtual node



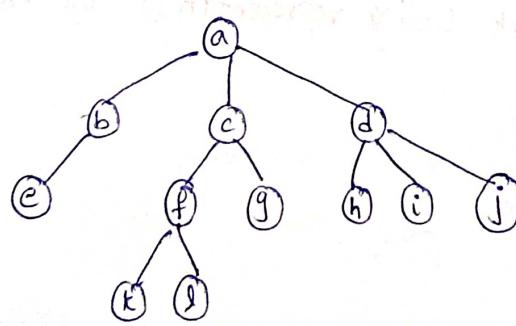


Now we remove the virtual node

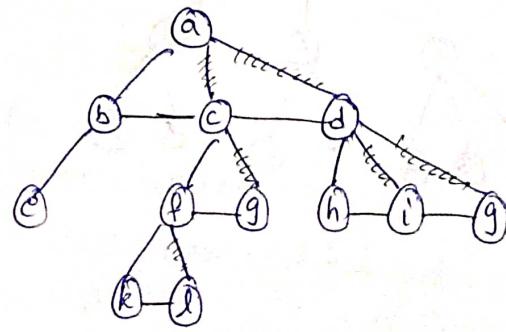


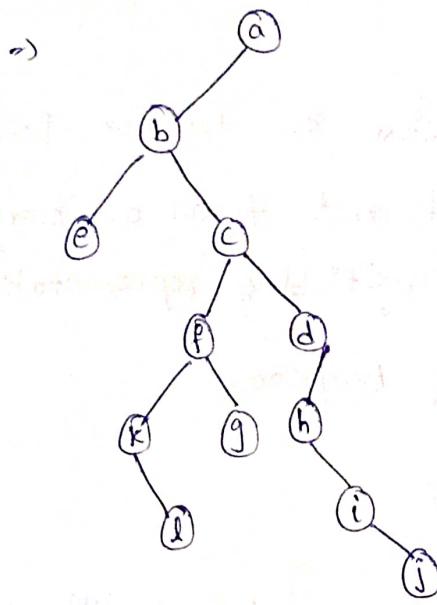
Height = 4

Eg: Convert below tree into binary tree



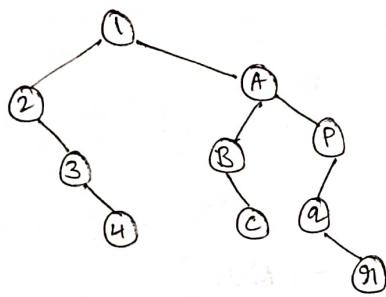
=>





P/17

Consider converted binary tree in P/16



~~D(P)~~ DO(1)
~~D(2)~~ |
~~D(A)~~ 2
~~D(B)~~ |
~~D(C)~~ 1
~~D(a)~~ |
~~D(g)~~

Here '3' is returned and 3 is no of ~~trees~~ trees in the forest.

Also consider the tree in P/15

the root of converted binary tree has no right child

∴ the value returned = 1

i.e. no of trees in the forest.

∴ opt (d)

20/09/20

*
Q23) Consider the pseudocode given below. The function `DoSomething()` takes as argument a pointer to root of an arbitrary tree represented by the leftMostChild-rightSibling representation.

Each node of the tree is of type `treenode`.

```
typedef struct treenode* treeptr;
struct treenode
{
    treeptr leftMostChild, rightSibling; } leftmost child - right sibling
}; representation.

int DoSomething (treeptr tree)
{
    int value=0;
    if (tree != NULL)
    {
        if (tree → left Most Child == NULL)
            value = 1;
        else
            value = DoSomething (tree → left Most Child);
        value = value + DoSomething (tree → rightSibling);
    }
    return (value);
}
```

when the pointer to the root of tree is passed as the argument to `DoSomething` the value returned by the function corresponds to

- a) no of internal nodes in the tree.
- b) height of the tree
- c) no of nodes without right sibling in the tree.
- d) no of leaf nodes in the tree.

Sol:

Consider tree (a) ... m. (right sibling is NULL)

Dos(a) returns 1

Here no of internal nodes = 0

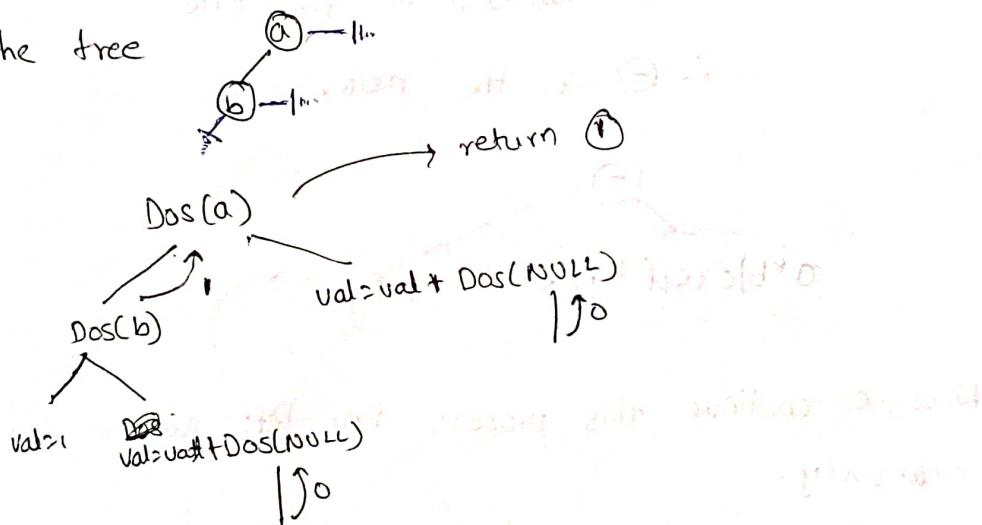
height = 0

nodes without right sibling = 1

no of leaf nodes = 1

∴ eliminate opt (a) & (b)

Consider the tree



∴ no of nodes without right sibling = 2

no of leaf nodes = 1

∴ opt (a)

P/18

$$a = -b + c * d | e + f \uparrow g \uparrow h + i * j$$

Precedence: $\uparrow > (-) > (* = /) > + > =$
 _{unary minus}

$\uparrow, (-)$ associativity is right to left
 $*$ many left to right
 $+,-$

Expression tree:

(i) In expression tree, operators are parents
 operands are children

(ii) least precedence operator is the root node.

Eg: Consider expression

$$a * b / c + e / f * g + k - x * y$$

Step 1: Scan the infix expression from left to right

+ & - have least precedence

The root node will be evaluated in the end

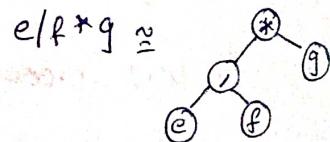
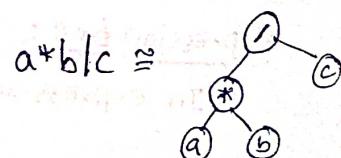
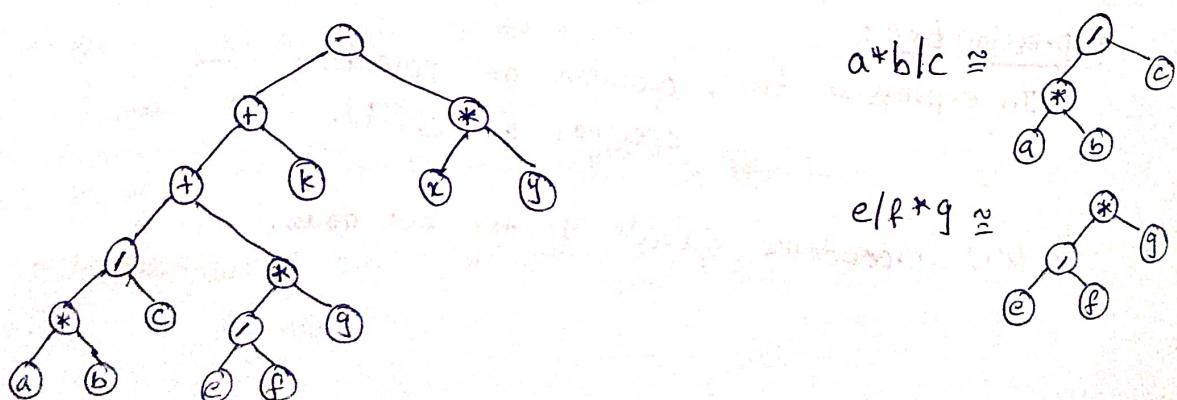
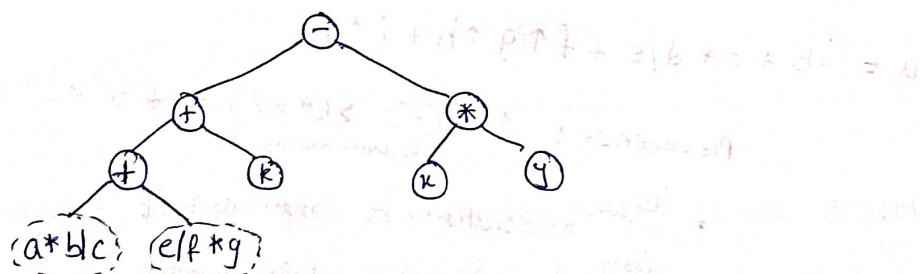
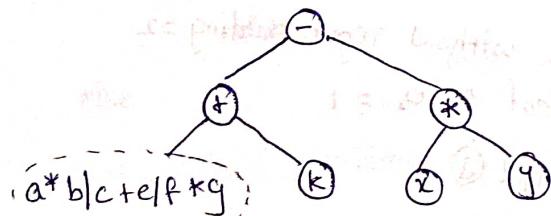
associativity of + & - is left to right.

It means rightmost of + or - must be the root, since it will be evaluated in the end.

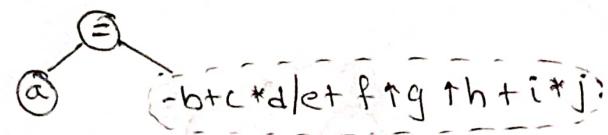
∴ - is the root.



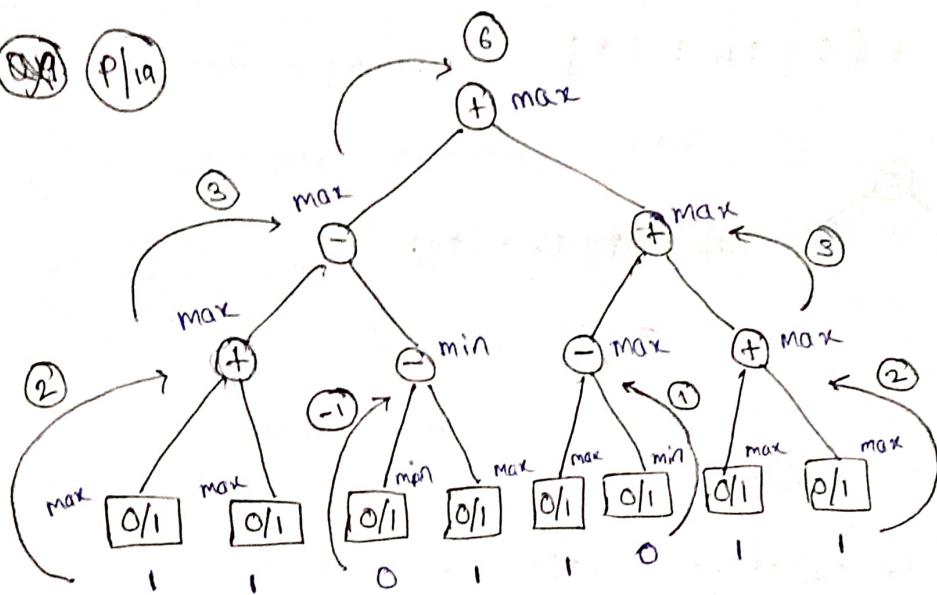
Now we continue this process for left subtree & right subtree recursively.



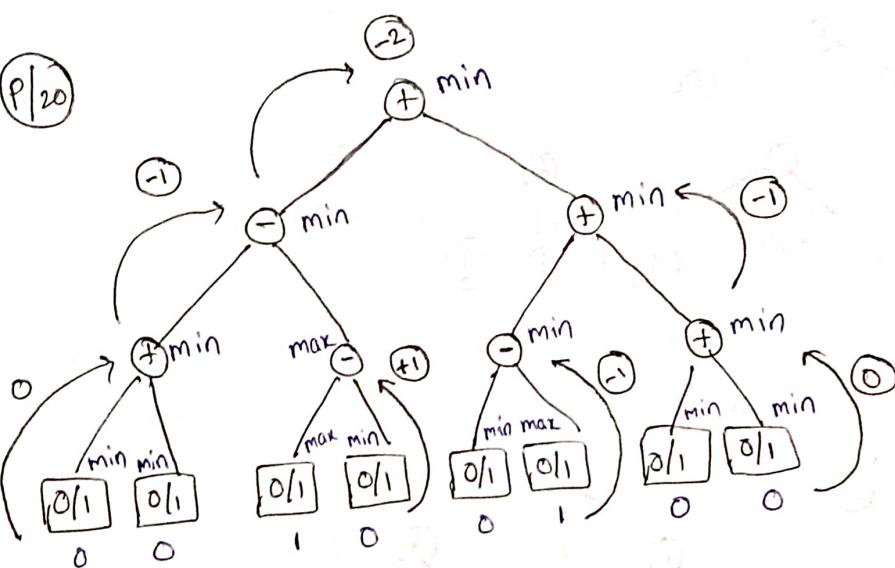
$$a = -b + c * d / e + f \uparrow g \uparrow h + i * j$$



Q19 P/19

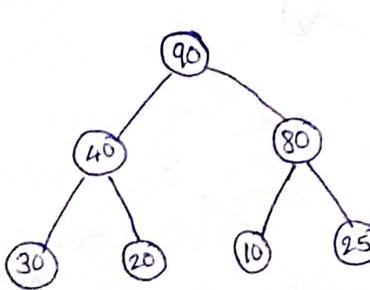


Q20 P/20

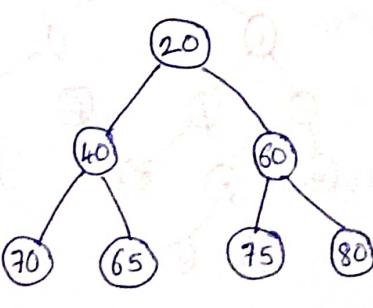


* * V. IMP
* Heap Tree:

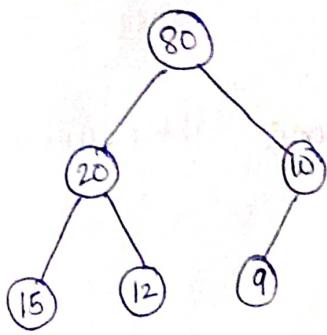
- A complete binary tree is called max heap, if parent value is greater than or equal to its children.
- A complete binary tree is called min heap if parent value is less than or equal to its children.



Max Heap

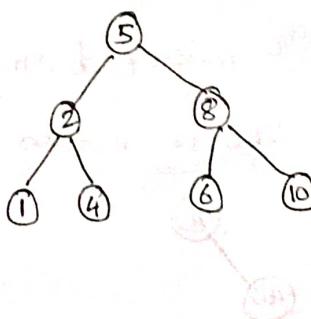
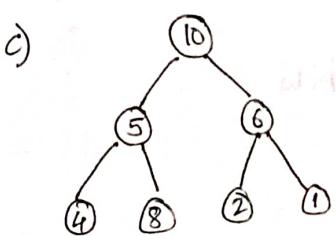
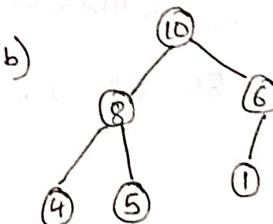
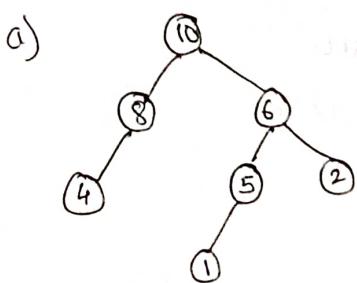


Min Heap



Max Heap

- (Q24) A max heap is a heap where the value of each parent is greater than or equal to the value of its children. Which of the following is a max heap?



opt a is not complete binary tree.
 \therefore not max heap

opt b is true.

Construction of heap tree:

Method 1:

Construct heap tree by inserting keys one after another in the given order.... $\Theta(n \log n)$

Method 2:

Convert the given array into heap (Build heap or heapify algorithm)
 $\hookrightarrow O(n)$

Ex: Construct heap tree (max heap) by inserting following keys in the given order. (insertion must be done one after other)

20, 40, 80, 60, 30, 75, 95, 110

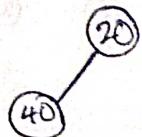
20:

(20)

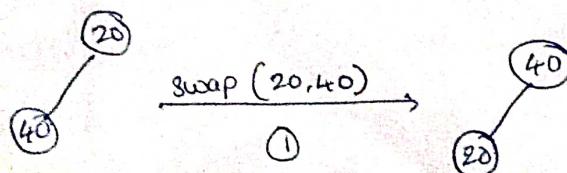
40:

Heap tree is complete binary tree

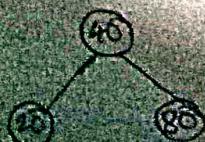
$\therefore 40$ is inserted as left child



Here $40 > 20$ and hence this doesn't satisfy heap tree property. So we need to perform an interchange.

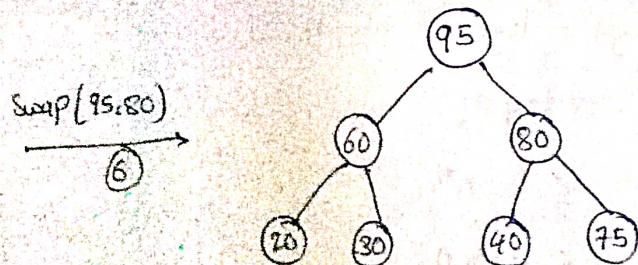
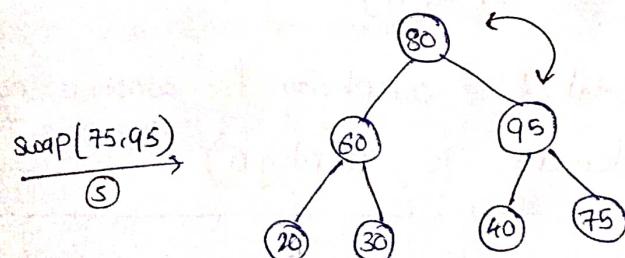
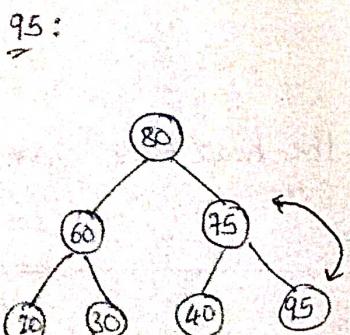
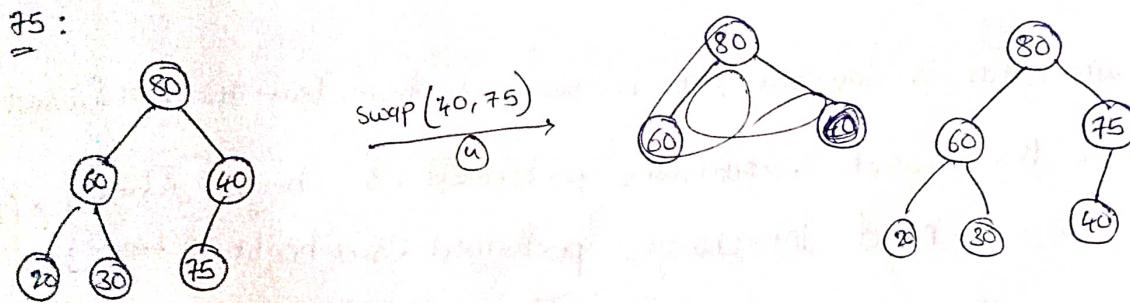
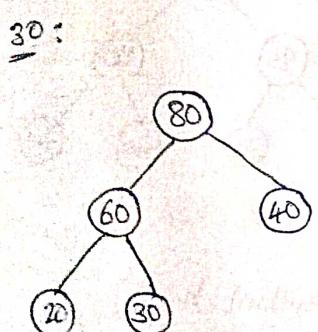
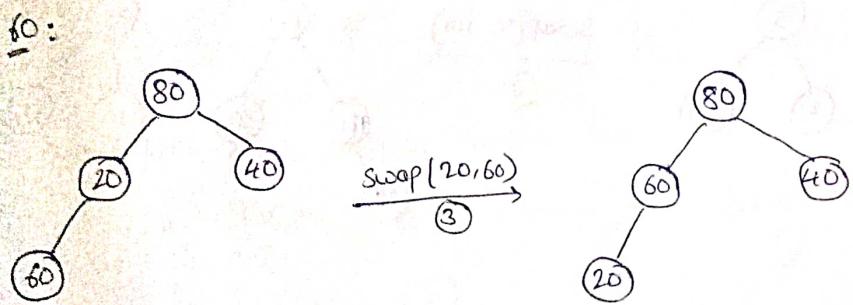
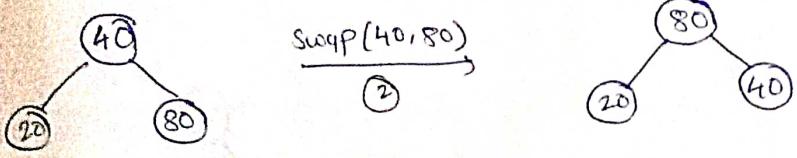


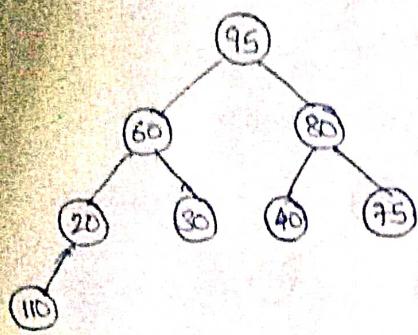
80:



$80 > 40$

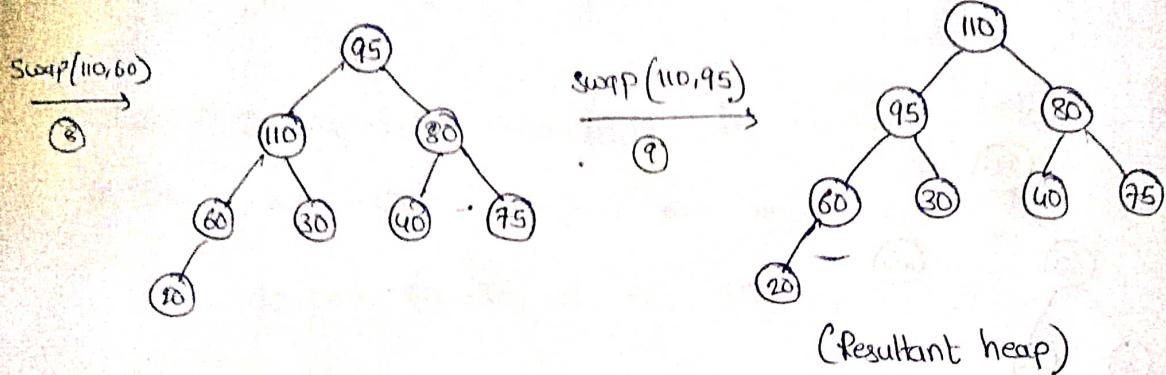
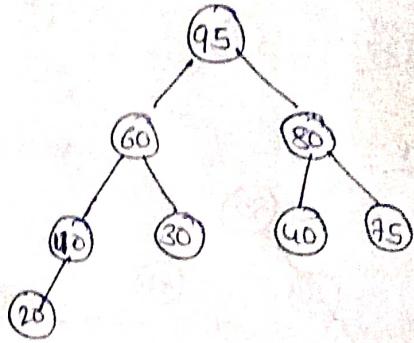
Max heap property is not satisfied





swap(10, 110)

(7)



Here no of interchanges performed = 9

→ when 110 is inserted, it is swapped from leaf to root (worst case)

Here no of comparisons performed : 3 ... height of tree
 no of interchanges performed : 3 ... height of tree } $O(\log_2 n)$

∴ The time complexity for inserting a node into a heap with n nodes is $O(\log_2 n)$

∴ The total time complexity for constructing the heap tree with n elements is $O(n \log n)$

$$= O(\log 1) + O(\log 2) + \dots$$

$$= \log 1 + \log 2 + \dots + \log n$$

$$= \log n!$$

$$\leq \log n^n$$

$$\leq n \log n$$

$$= O(n \log n)$$

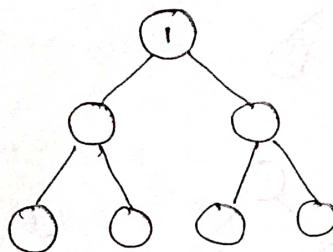
21/09/20

(Q25) How many no of min heaps are possible by using keys
1, 2, 3, 4, 5, 6, 7.

Sol:

Heap tree is complete binary tree

→ and since it is minheap the tree must be

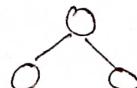


LST has 3 locations

So we can choose 3 elements from 1, 2, 3, 4, 5, 6, 7 to construct LST

→ i.e., ${}^6C_3 = 20$ ways

After selecting 3 elements for LST



the minimum of 3 has to be in the root

the other two elements into the leaf can be arranged in 2 ways.

i.e., 2 way for each combination of 3 elements.

∴ the RST can also be constructed in 2 ways.

$$\begin{aligned}\therefore \text{total no of min heaps possible} &= {}^6C_3 \times 2 \times 2 \\ &= 20 \times 2 \times 2 = 80 \text{ ways.}\end{aligned}$$

~~Finding no of min heaps possible with key 1, 2, 3, 4, 5, ... n~~

→ Finding no of min heaps possible with keys 1, 2, 3... n be $T(n)$

<u>n</u>	<u>Possible minheaps</u>	<u>$T(n)$</u>
1	①	$T(1) = 1$
2	① ②	$T(2) = 1$
3	① ② ③ ① ③ ②	$T(3) = 2 = {}^1C_1 \cdot {}^2C_1 \cdot {}^1C_1$ root
4	① ② ③ ④ ① ② ④ ③ ① ③ ② ④	$T(4) = {}^1C_1 \cdot {}^3C_2 \cdot {}^1C_1 \cdot 1! \cdot {}^1C_1 = 3$ root LST LST parent RST arranging leaf nodes & LST

Now for n elements, '1' is placed at the root.

Now assume LST has k elements, then no of elements in RST will be $n-k-1$

$$T(n) = {}^1C_1 \cdot (n-1) {}^{n-1}C_k \cdot T(k) \cdot T(n-k-1)$$

Array representation of heap tree:

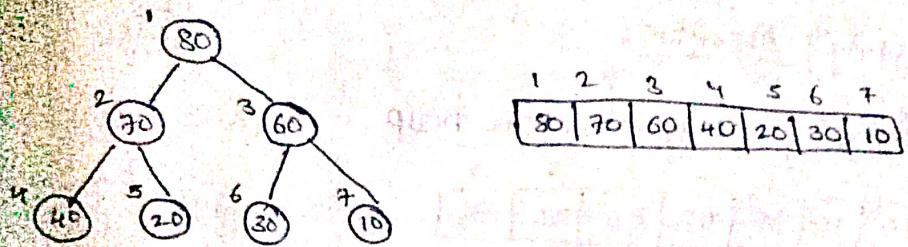
i) Array index of root node is 1. (general convention but not compulsory)

ii) For any node whose index is at 'i' then

index of leftchild --- $2i$

index of rightchild --- $2i+1$

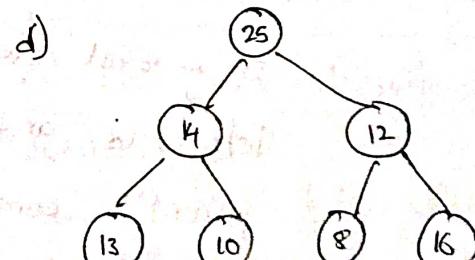
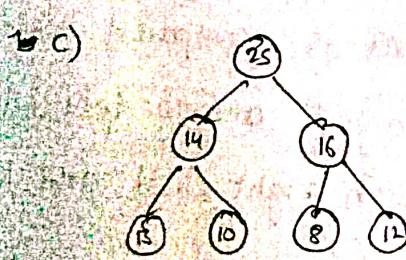
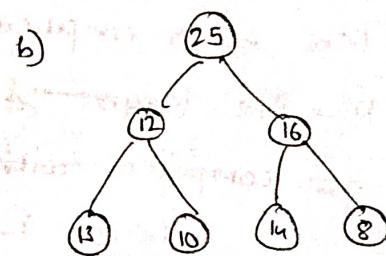
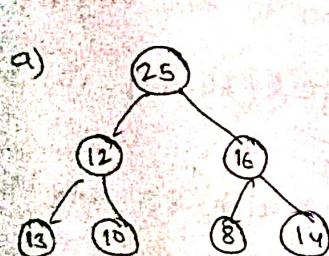
index of parent --- $\lfloor i/2 \rfloor$



Q26 Consider a binary max heap implemented using an array. Which of the following represents a binary max-heap?

- a) 25 12 16 13 10 8 14
- b) 25 12 16 13 10 8 18
- c) 25 14 16 13 10 8 12
- d) 25 14 12 13 10 8 16

Sol: -



∴ opt c

Method 2:

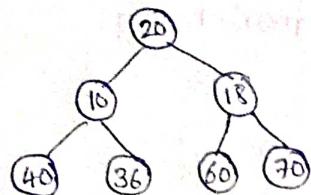
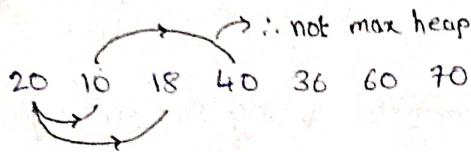


∴ opt c

Build Heap (Heapify) Algorithm:

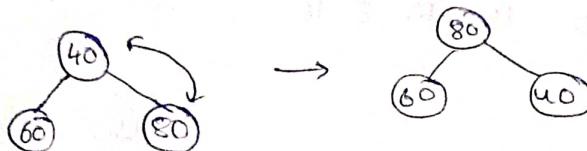
Convert the following array into max heap

1	2	3	4	5	6	7
20	10	18	40	36	60	70



1. By default all leaf nodes satisfy the heap tree property.

Eg: Consider



Here 40 is compared with 60 to find largest(40, 60).

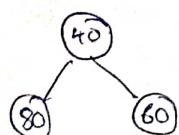
Now the ~~largest~~ largest(40, 60) is 80

compared with 80.

i.e., 60 is compared with 80 and then
80 is swapped with 40.

i.e., In general every element is compared with its left child and the largest among left child & parent is compared with right child and swap is performed.

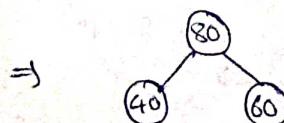
Eg: Consider

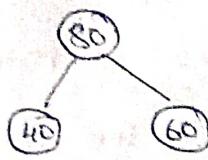


40 & 80 are compared ~~and~~ and 80 is largest

Now 80 is compared with 60 and 80 is largest.

~~so 80 moves to the root~~ so 80 is swapped with root.

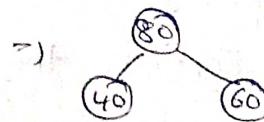




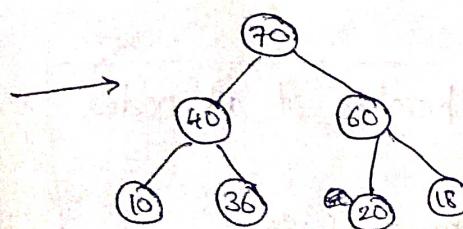
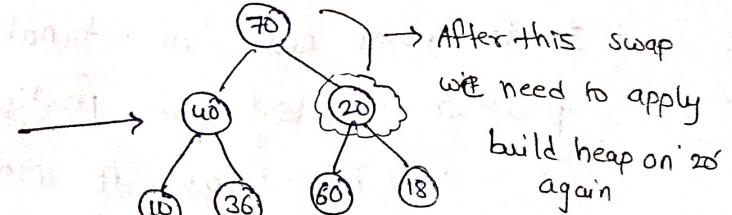
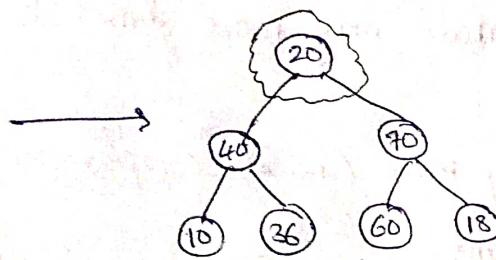
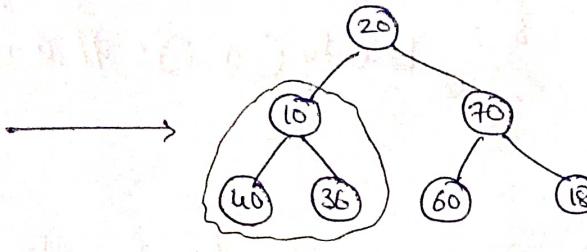
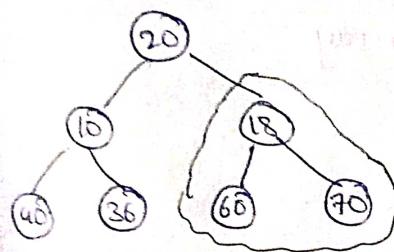
80 & 40 are compared \Rightarrow 80 is largest

80 & 60 are compared \Rightarrow 80 is largest.

\therefore no swapping is performed.



Now we apply build heap on the given problem



At every node we perform two comparisons

i) parent & left child

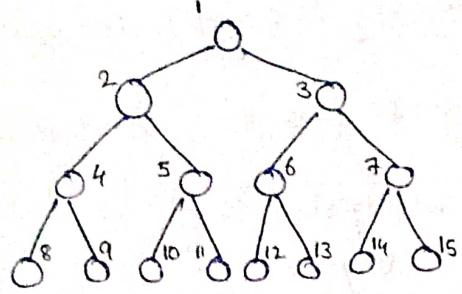
ii) largest(parent, left child) & right child

Analysis of Build heap:

(i) By default all leaf nodes satisfy heap tree property. So we don't need to apply heapify on leaf nodes.

(ii) If there are 'n' elements in array representation of complete binary tree then,

leaf node index starts from $\left\lfloor \frac{n}{2} \right\rfloor + 1$ and ends at n .
i.e., the highest index of non-leaf node is $\left\lfloor \frac{n}{2} \right\rfloor$



Here we call build heap on nodes

7, 6, 5, 4, 3, 2, 1 in this order.

$$\left\lfloor \frac{n}{2} \right\rfloor = \left\lfloor \frac{15}{2} \right\rfloor = 7$$

(iii) we have to apply heapify algorithm starting from highest index of non leaf node and continue the process till we reach the root node.

i.e., for ($i = \lfloor n/2 \rfloor$; $i \geq 1$; $i--$)

{ heapify(a, i); // a is array
 }

(iv) The total time complexity for heapify algorithm.

In the worst case, an internal node may move from its position to leaf node position.

i.e., in worst case it may move (no of comparison) upto distance of its height.

~~time~~

∴ time complexity = sum of heights of all nodes of building heap using build heap = $O(n)$ \hookrightarrow no of nodes.

(Q27) we have a binary heap on n elements and wish to insert n more elements (not necessarily one after another) into this heap. The total time required for this is

a) $O(\log n)$

b) $O(n)$

c) $O(n \log n)$

d) $O(n^2)$

Build heap

Sol:

we first insert 'n' node into array normally.

so we have $2n$ nodes.

we apply build heap on this.

$$\Rightarrow O(n)$$

i.e., $\Theta(n)$

Q28

what is the time complexity for converting complete binary tree into max-heap if only root node does not satisfy the heap tree ~~property~~ property.

- a) $\Theta(\log n)$
- b) $\Theta(1)$
- c) $\Theta(n)$
- d) $\Theta(n \log n)$

Sol:

In the worst case root node may be moved to leaf node.

This takes $\log_2 n$ comparision

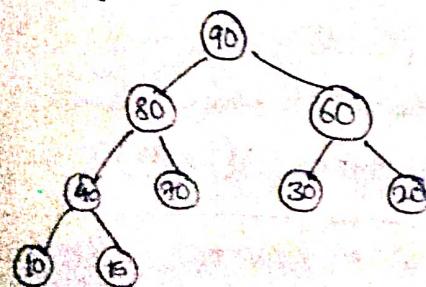
$$\therefore \Theta(\log n)$$

Since from above question,

if there is only one node in tree which doesn't satisfy the heap tree property, then converting the tree into heap tree take $\Theta(\log n)$

Deletion Operation on heaptree:

In heap tree, which is a priority queue, we perform deletion only on root node.

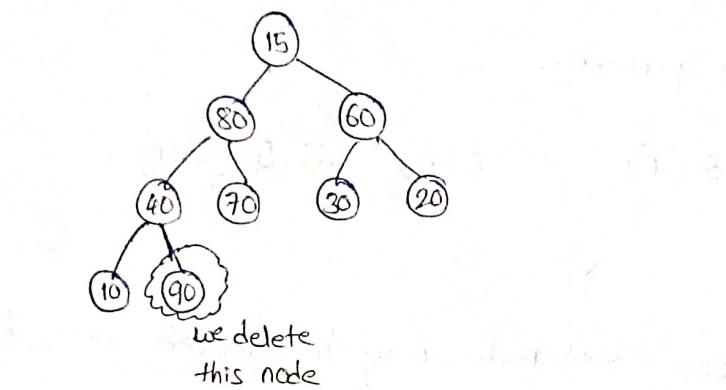


Step 1: swap ($A[1], A[n]$)

Step 2: Apply heapify algorithm on root node and convert complete binary of $(n-1)$ nodes into heap tree.

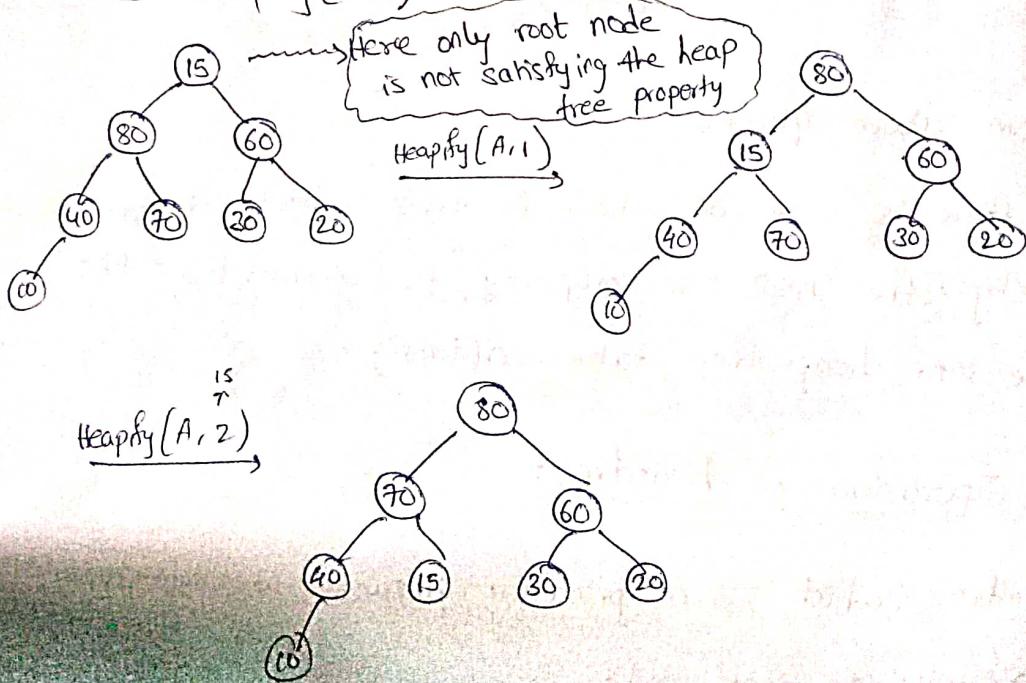
Delete 90:

Step 1: swap (90, 15)



Now no of nodes = $9-1=8$

Step 2: Heapify ($A[1]$)



Analysis:

Swapping can be done in constant time.

Applying heapify on root nodes takes $O(\log n)$

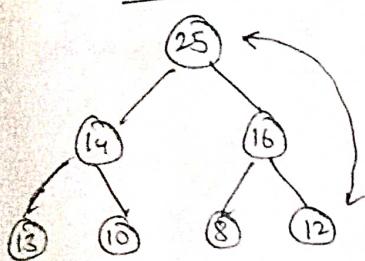
∴ time complexity for deletion on heap tree = $O(\log n)$

(Q29) Consider max heap implemented with array. given below

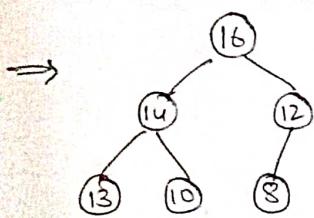
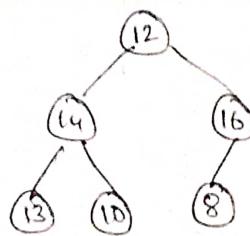
25 14 16 13 10 8 12.

what is the content of the array after two delete operations,

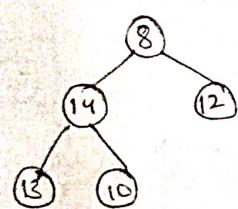
Sol: Delete 1:



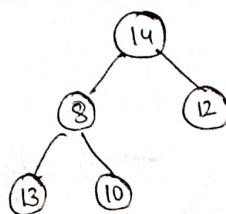
→



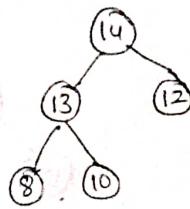
Delete :



→



→



∴ Ans: 14 13 12 8 10

(Q30) An operator `delete(i)` for a binary heap data structure is to be designed to delete the item in i -th node. Assume that the heap is implemented in an array and i refers to i th index of the array. If the heap has depth d (no of edges on the path from root to the furthest leaf) then what is the time complexity to re-fix the heap efficiently after the removal of the element?

a) $O(1)$ b) $O(d)$ but not $O(1)$

c) $O(2^d)$ but not $O(d)$ d) $O(d \cdot 2^d)$ but not $O(2^d)$

Sol:

Here d is height

we perform this operation ~~similar to th~~ in a similar way we deleted the root node.

so the ~~delete~~ swapped node may move ' d ' distance (in worst case)

$\therefore O(d)$

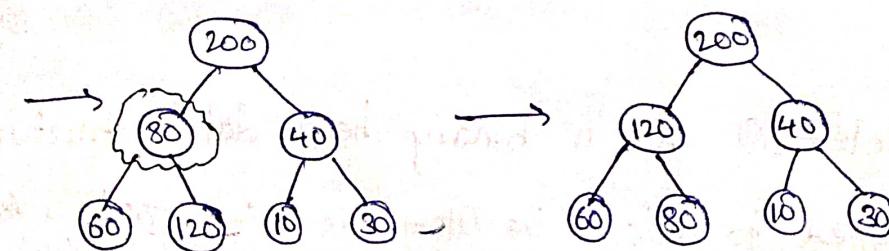
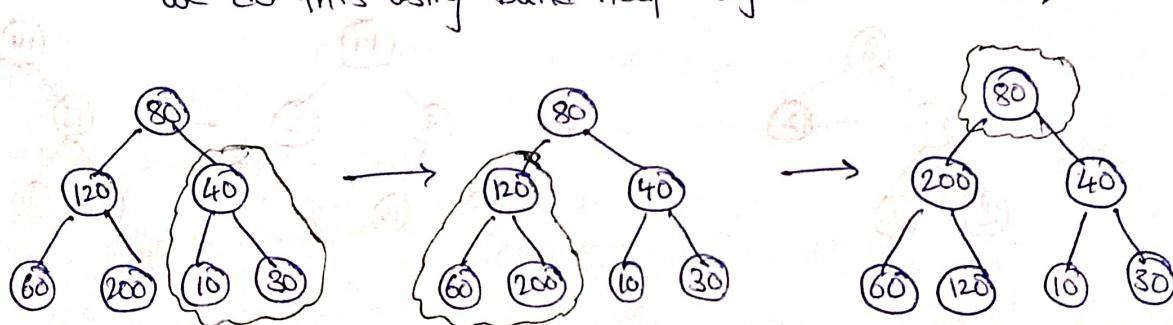
Heapsort:

Sort the following array using heapsort.

1	2	3	4	5	6	7
80	120	40	60	200	10	30

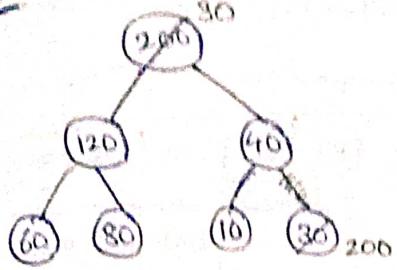
Step 1: Convert the given array into max heap.

we do this using build heap algorithm ... $O(n)$



Step 2: In each iteration ' i ', we have to delete root node by replacing it with leaf node at last index and by applying heapify(A, i) by taking size of A as $n-i$.
value of ' i ' starts from 1 and runs till $n-1$

i=1:

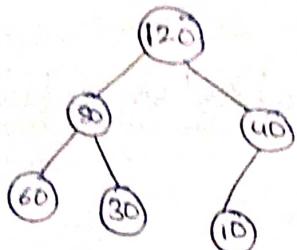


30	120	40	60	80	10	200
----	-----	----	----	----	----	-----

Heapify ($A, 1$)

$$\text{size of } A = 6 \cdot 7 - i = 7 - 1 = 6$$

Heapify ($A, 1$)

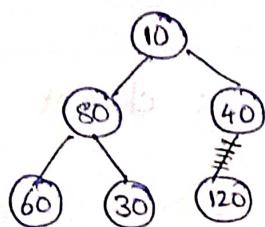


120	80	40	60	30	10	200
-----	----	----	----	----	----	-----

↳ Deleted from heap tree

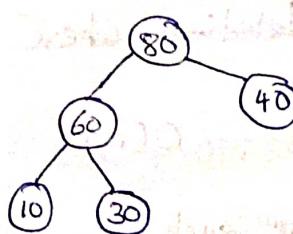
i=2:

swap ($A[1], A[7-i+1]$)



10	120	30	60	80	40	200
----	-----	----	----	----	----	-----

Heapify ($A, 1$) .. size of array = $n - i = 7 - 2 = 5$

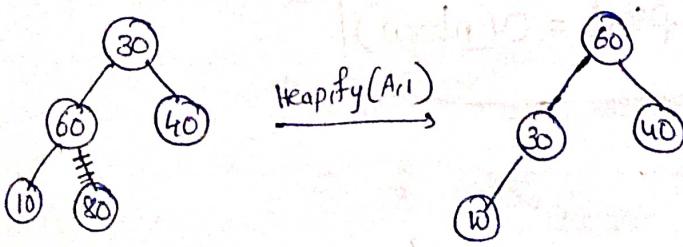


80	60	40	10	30	120	200
----	----	----	----	----	-----	-----

Delete from heap tree

i=3:

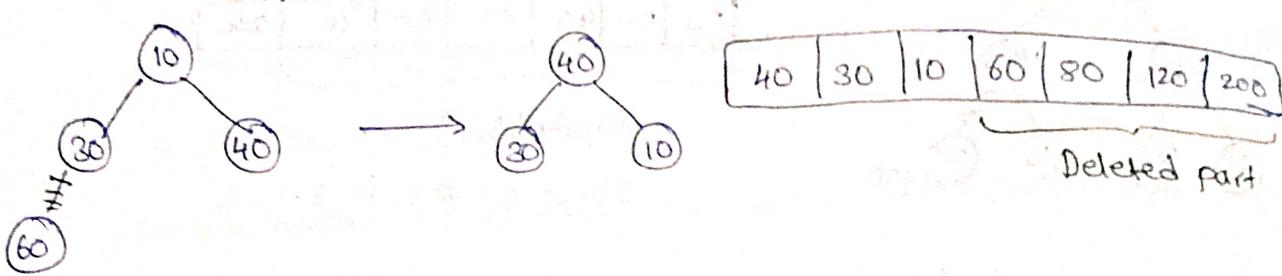
Heapify ($A, 1$)



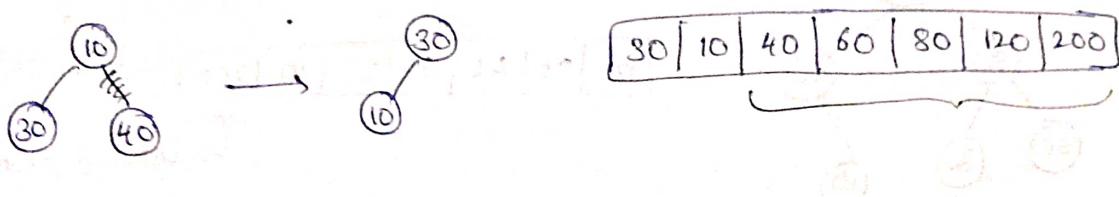
60	30	40	10	80	120	200
----	----	----	----	----	-----	-----

↳ Deleted part

i=4



i=5



i=6:



Now this deletion of last but one element forms the sorted array.

Analysis:

Here ~~deleting~~ we perform n deletions where each de

• Here we first construct heap $\dots O(n)$

later we perform n deletions where each $\dots O(n \log n)$
deletion takes $\log n$ time

\therefore Time complexity of heapsort = $O(n \log n)$

Finding maximum & minimum elements in heap tree.

	maxheap	minheap
max element	$O(1)$	$O(n)$
Min element	$O(n)$	$O(1)$

Note: In max heap, the min element is always at leaf node

The leaf node index starts from $\lfloor \frac{n}{2} \rfloor + 1$ and ends at n

$$\therefore \text{no of comparisions} = n - \lfloor \frac{n}{2} \rfloor - 1 \quad \text{no of leaf nodes} - 1$$

$$\cancel{\lfloor \frac{n}{2} \rfloor} = n - \lfloor \frac{n}{2} \rfloor - 1$$

Similarly for min heap also $\lfloor \frac{n}{2} \rfloor - 1$.

$\therefore T.C$ for finding min element in max heap & finding max element in min heap $= O(n)$

(Q31) Consider array representation of min heap containing 1023 elements.

The minimum no of comparisions required to find maximum in the heap is _____

Sol:

~~leaf~~ ^{node} index starts at $\lfloor \frac{n}{2} \rfloor + 1 = 511 + 1 = 512$ minimum to

leaf node index ends at $n = 1023$

$$\therefore \text{no of leaf nodes} = 1023 - 512 + 1$$

$$= 512$$

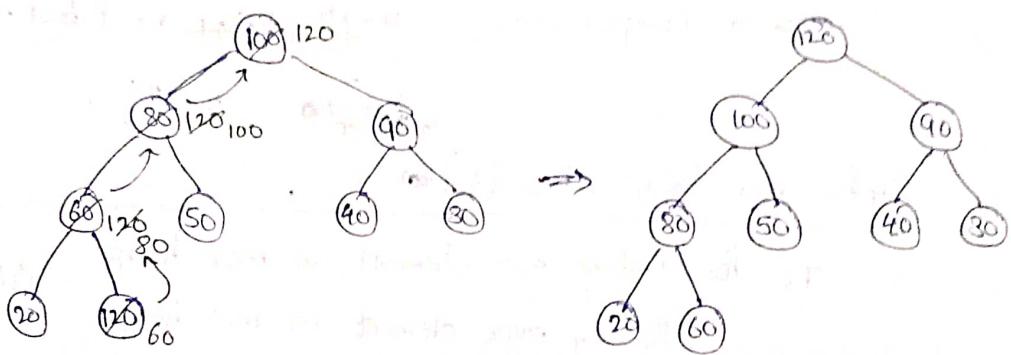
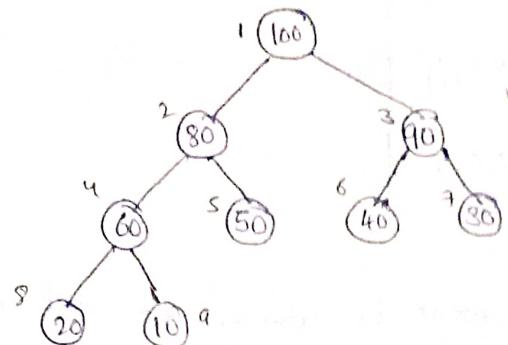
~~512 comparisons.~~

For k nodes we need $k-1$ comparisons.

$$\therefore 512 - 1 = 511 \text{ Comparisons}$$

Increasing key (or) Decreasing key Operations

Consider Increase value of ~~key~~ to $A[9] = 120$



Time Complexity : $O(\log n)$
for increase key &
decrease key

Note :

- Insertion ~~of~~ into heap tree : $O(\log n)$
- Deletion ~~of~~ on heap tree (Root node) : $O(\log n)$
- Increase or Decrease key : $O(\log n)$
- Construction of heap tree
 - insertion elements one after other : $O(n \log n)$
 - Build heap method : $O(n)$
- Heapsort : $O(n \log n)$

Find	max heap	min heap
max element	$O(1)$	$O(n)$
min element	$O(n)$	$O(1)$

In
seq.
bina

(Q32) Consider the process of inserting an element into a max heap represented by an array. Suppose that we perform binary search on the path from the new leaf to root to find the position for the newly inserted element. The no of comparisons performed is

- a) $O(\log_2 n)$ b) $O(\log_2(\log_2 n))$ c) $O(n)$ d) $O(n \log n)$

Sol:

without using binary search we need compare approximately $\log_2 n$ elements.

∴ no of comparison for binary search is $O(\log_2(\log_2 n))$

However Note that here the comparisons required is $O(\log_2(\log_2 n))$ is for find position of newly inserted element but not for entire insertion.

Note:

→ no of swap operations for insertion into heap tree using binary search are $O(\log n)$ i.e., same as using sequential search.

⇒ It is because after ~~finding~~ comparing with 'mid' all the elements under the node 'mid' must be swapped.

Inserction	Comparision	swapping	TC
seq. search	$O(\log n)$	$O(\log n)$	$O(\log n)$
binary search	$O(\log_2 \log_2 n)$	$O(\log n)$	$O(\log n)$

(Q33) Suppose we have a balanced BST holding n numbers. We are given two numbers L and H and wish to sum up all the numbers in T that lie b/w L and H . Suppose there are m such numbers in T . If the tightest upper bound on the time to compute the sum is $O(n^a \log^b m + m^c \log^d n)$ then the value of $a + 10b + 100c + 1000d$

Sol:

- a) 60 b) 110 c) 210 d) 50

Sol:

$$\text{given } O(n^a \log^b m + m^c \log^d n)$$

we first search L in $O(\log n)$ time and then we search H in $O(\log n)$ time.

Now we perform inorder on L till H i.e. for m elements this can be done in $O(m)$ time

\therefore time complexity is $O(\log n + m)$

$$\Rightarrow a=0, b=1, c=1, d=0$$

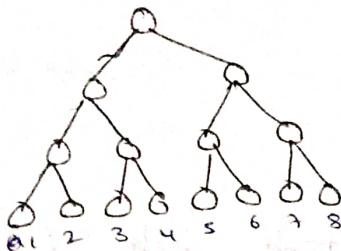
$$\Rightarrow a + 10b + 100c + 1000d = 110$$

(Q34) Let T be a full binary tree with 8 leaves. Suppose two leaves a and b of T are chosen uniformly and independently at random. The expected value of the distance b/w a and b in T (i.e., the no of edges in the unique path b/w a and b) is —

Sol:

The nodes are chosen independently
i.e., repetition is allowed.

$$\therefore \text{total ways for choosing} = 8 \times 8 = 64$$



~~For 2nd if~~

two nodes chosen can be same --- path length = 0

this can be done in 8 ways

two nodes chosen can be sibling --- path length = 2

this can be done in $2^2 \cdot 4 = 8$ ways

two nodes chosen may have path length = 4

~~this is possible if one node is from (1,2) and other~~

this is possible if one node is from (1,2) & other is from (3,4)

say we have for (5,6) & (7,8)

$$(1,2) \text{ & } (3,4) \cdots 2^2 \cdot 2^2 = 8$$

$$(5,6) \text{ & } (7,8) \cdots 2^2 \cdot 2^2 = 8$$

$$\frac{16}{16}$$

two nodes chosen may have path length = 6

this is possible if one node is from (1,2,3,4) and other is

from (5,6,7,8)

$$\text{no of ways} = 2^4 \cdot 4 = 32$$

$$\therefore \text{expected value} = 0 \cdot \frac{8}{64} + 2 \cdot \frac{8}{64} + 4 \cdot \frac{16}{64} + 6 \cdot \frac{32}{64}$$

$$= \frac{2}{8} + \frac{8}{8} + \frac{8}{8} + \frac{24}{8} = \frac{2}{8} + 1 + 3$$

$$= \frac{42}{8} = 5.25 = 0.25 + 4 = 4.25$$

22/09/20

Hashing:

1. Accessing data items from hash-table with $O(1)$ time, but the goal is not yet achieved.

2. Hash function:

$$H(x) = x \bmod m$$

H: Hash function

x: key

m: hash table size, where index of hash table ranges from 0 to $m-1$.

Eg: Let

$x: 10, 20, 30, 40, 50$

$$h(x) = x \bmod 7$$

$$h(10) = 10 \bmod 7 = 3^{\text{rd}} \text{ location}$$

$$h(20) = 20 \bmod 7 = 6^{\text{th}} \text{ location}$$

$$h(30) = 30 \bmod 7 = 2$$

$$h(40) = 40 \bmod 7 = 5$$

$$h(50) = 50 \bmod 7 = 1$$

Index	key
0	
1	50
2	30
3	10
4	
5	40
6	20

Accessing data:

Assume we need to access key 40

$$H(40) = 40 \bmod 7 = 5$$

so we ~~need~~ obtain 40 in the index 5.

∴ Time complexity for accessing = frequency count of fundamental data instructions. = $O(1)$

Collision

The is co

Eg: Let

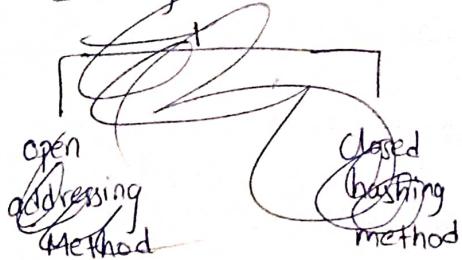
$$x: 10, 20, 30, 40, 50$$

$$h(x) = x \bmod 10$$

$$\begin{aligned} h(10) &= 10 \bmod 10 = 0 \\ h(20) &= 20 \bmod 10 = 0 \end{aligned} \quad \left. \right\} \rightarrow \text{collision}$$

(Collision handling)

resolving collision



index	key
0	10
1	
2	
3	
4	
5	
6	
7	
8	
9	

Separate chaining:

→ Here we resolve collisions by creating

a separate linked list for each hash index

Accessing data:

Assume we need to access 50.

$$H(50) = 50 \bmod 10 = 0$$

So at 0th index we perform linear search.

Here

∴ Time complexity $\theta = O(n)$

→ The hash function which takes $O(1)$ time for accessing data

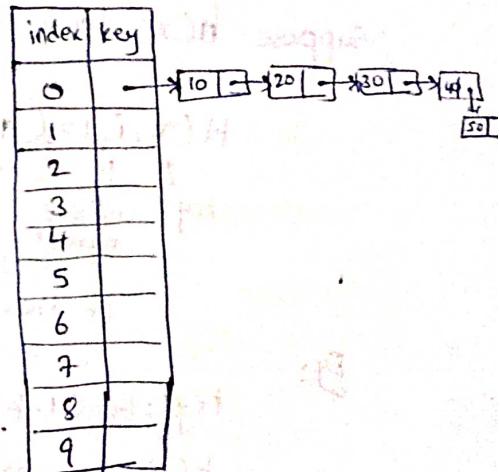
is called good hash function.

→ The hash function which takes $O(n)$ time for accessing data

is called bad hash function.

Collision:

The situation in which more than one key hashes to same location is called collision.



Collisions can be resolved using following techniques.

Collision-resolving Techniques

1. Open addressing (closed hashing)

- ↳ linear probing \leftarrow V.I.M.P
- ↳ Quadratic probing
- ↳ Double Hashing

2. Separate chaining (external hashing)

Open Addressing:

i) Linear probing:

$$\text{let } h(x) = x \bmod m$$

suppose $h(x)$ leads to collision and let collision number is 'i'.

$$h(x, i) = (h(x) + i) \bmod m$$

↑
key collision
 number.

If there are no collisions then value of 'i' is 0.

Eg:

keys: 14, 21, 28, 33, 54, 61

$$h(x) = x \bmod 7$$

Here, load factor = $\frac{6}{7} \leq 1$

∴ we can use linear probing.

If m = hash table size and
 n = no of keys then

load factor of hash table,

$$\lambda = \frac{n}{m}$$

To apply linear probing,

only for any closed hashing, $\lambda \leq 1$

Rule: If k^{th} location is already occupied

by some key then we have to search

in $(k+1)^{\text{th}}$ location, if $(k+1)$ is also occupied then we search in

$(k+2)^{\text{th}}$ location and so on till $(m-1)^{\text{th}}$ and then we start
from '0' to $(k-1)^{\text{th}}$ location

0, 1, 2, 3, ..., k-1, k, k+1, ..., m-1

Q35

$$h(x) = x \bmod 7$$

$$\underline{14}: h(14) = 14 \bmod 7 = 0^{\text{th}} \text{ location} \quad \} \text{ collision}$$

$$\underline{21}: h(21) = 21 \bmod 7 = 0^{\text{th}} \text{ location} \quad \} \text{ collision}$$

now $i=1$ (collision number)

Now we resolve this using linear probing

$$H(21, 1) : (h(21) + 1) \bmod 7 \\ = (0 + 1) \bmod 7 = 1^{\text{st}} \text{ location}$$

$$h(28) = 28 \bmod 7 = 0^{\text{th}} \text{ location}$$

↓
1st collision

~~28~~:

$$i=1, H(28, 1) = (h(28) + 1) \bmod 7 \\ = (0 + 1) \bmod 7 = 1^{\text{st}} \text{ location}$$

↓
2nd collision

0	14
1	21
2	28
3	
4	
5	33
6	54

$$i=2, H(28, 2) = (h(28) + 2) \bmod 7 \\ = (0 + 2) \bmod 7 \\ = 2 \bmod 7 = 2^{\text{nd}} \text{ location}$$

~~33~~:

$$h(33) = 33 \bmod 7 = 5^{\text{th}} \text{ location}$$

~~54~~:

$$h(54) = 54 \bmod 7 = 5^{\text{th}} \text{ location}$$

∴ we store in 6th location

(Q35) Given the following input (1322, 1331, 1171, 9679, 1989, 6171, 8173, 1199)

and the hash function $x \bmod 10$. which of the following statements are true.

~~(Q35)~~ i) 9679, 1989, 4199 hash to same value.

ii) 1471, 6171 hash to the same value.

iii) All elements hash to the same value

iv) Each element hashes to a different value.

- a) i only b) iii only c) i, ii, iii only d) iii & iv or iv

Sol:

(Q36) A hash table contains 10 buckets and uses linear probing to resolve collisions. The key values are integers and hash function is $\text{key} \% 10$. If the values 43, 165, 62, 123, 142 are inserted in the table, in what location 142 will be inserted?

- a) 2 b) 3 c) 4 d) 6

Sol:

43 --- 3rd loc

165 --- 5th loc

62 --- 2nd loc

123 --- 3 is filled

\therefore 4th loc

142 --- 2nd is filled

3rd is filled

4th is filled

5th is filled

\therefore 6th loc

(P/4) $h(x) = (3x+4) \bmod 7$

$$h(1) = (3+4) \bmod 7 = 0^{\text{th}}$$

$$h(3) = 13 \bmod 7 = 6$$

$$h(8) = 28 \bmod 7 = 0 \cdots \text{collision}$$

\therefore insert at 1

$$h(10) = 34 \bmod 7 = 6 \cdots \text{collision}$$

\therefore insert at 2

0	1
1	8
2	10
3	
4	
5	
6	3

$\therefore 1, 8, 10, -, -, -3$

(P/6)

0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

(P/3) load factor = $\frac{\text{no of elements}}{\text{no of slots}} = \frac{2000}{25} = 80$

(P/7)

Q3

	0	1	2	3	4	5	6	7	8	9
a			42	52	34	23	46	33		
b			42	23	34	52	33	46		
c			42	23	34	52	46	33		
d			42	33	23	34	46	52		

∴ opt (c)

(P/8)

0	
1	42
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

~~from hash +~~

33 hashes to index '3'

However in the given table it present in the index 33.

This means 33 must be the last key of the sequence.

keys:

				33
--	--	--	--	----

42 & 52 map hashes to 2nd index.

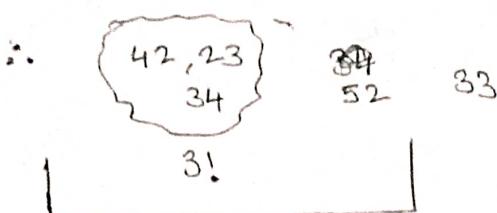
From figure 42 must have come first.

In fig, 23 is present in 3rd loc.

this means 23 must have come earlier than 52

Also 34 is in 4th index & 52 is in 5th index.
so 34 must have come earlier.

The only restriction for key 46 is that it must come before 33.



46 can be inserted
in any of these 5 loc
in 5 ways

$$\therefore \text{no of ways} = 3! \times 5 = 30 \text{ ways}$$

23/09/20

Disadvantage of Linear probing

Consider $h(x) = x \bmod 11$

keys: 29 46 18 36 43 21 24 54
 $h(x)$: 7 2 8 3 10 0 4 1

0	21
1	54
2	46
3	36
4	24
5	-
6	-
7	29
8	18
9	-
10	43

→ Now if a key to be inserted maps to 10th index, then it has to be stored in 5th index.

→ In this example, 10, 0, 1, 2, 3, 4 forms a cluster of indices.

$$76 \bmod 11 \quad 76 \bmod 11 = 10$$

∴ it is stored at 5th index.

→ The no of probes for search 76 (or) inserting 76 = 7

→ The main disadvantage of linear probing is ~~clustering~~ primary clustering i.e., group of records stored next to each other.

Quadratic Probing:

- In linear probing, the colliding keys are stored near the initial collision point, it results formation of clusters.
- In quadratic probing, this problem is solved by storing the colliding key away from the initial collision point.
- The formula for quadratic probing is

$$H(x, i) = (h(x) + i^2) \bmod m$$

In linear probing searching process is $h(x), h(x)+1, h(x)+2, \dots \bmod m$

In quadratic probing searching process is $h(x), h(x)+1, h(x)+4, \dots \bmod m$

Eg: let $h(x) = x \bmod 11$ use quadratic probing.

Keys:	46	28	21	35	57	39	19	50
$h(x)$:	2	6	10	3	0	7	8	4

0	57
1	
2	46
3	35
4	50
5	
6	28
7	39
8	19
9	
10	21

$$h(35) = 35 \bmod 11 = 2$$

$$\Rightarrow H(35, 1) = (2+1^2) \bmod 11 = 3$$

$$\rightarrow h(57) = 57 \bmod 11 = 2$$

$$H(57, 1) = (2+1^2) \bmod 11 = 3$$

$$H(57, 2) = (2+2^2) \bmod 11 = 8$$

$$H(57, 3) = (2+3^2) \bmod 11 = 0$$

$$\rightarrow h(50) = 50 \bmod 11 = 6$$

$$H(50, 1) = (6+1^2) \bmod 11 = 7$$

$$H(50, 2) = (6+2^2) \bmod 11 = 10$$

$$H(50, 3) = (6+3^2) \bmod 11 = 4$$

Consider for some key, k ,

$$h(k) = 2$$

$$\text{Now } H(k, 1) = (2+1) \bmod 11 = 3$$

$$H(k, 2) = (2+4) \bmod 11 = 6$$

$$H(k, 3) = (2+9) \bmod 11 = 0$$

Double

$$H(k, 4) = (2+16) \bmod 11 = 7$$

$$H(k, 5) = 5$$

$$H(k, 6) = 5$$

$$H(k, 7) = 7$$

$$H(k, 8) = 0$$

$$H(k, 9) = 6$$

Here any key that follows collision path $0, 2, 3, 5, 6, 7$ has the problem of secondary clustering occurs.

(2, 3, 6, 0, 7, 5, 5, 7, 0, 6, 3, 2, 3, 6, 0, 7, 5, 5, ...)
(observe the sequence)

So any key k with $h(k) = 2$, we search slot for it

only in 6 distinct locations i.e., $0, 2, 3, 5, 6, 7$.

Although has If the slots $0, 2, 3, 5, 6, 7$ are full, we cannot insert key ' k ' despite having other free slots.

This problem is known as secondary clustering.

Here only half of the hash table is used.

Note:

→ Quadratic probing overcomes the problem of second primary clustering, but it has another problem i.e., secondary clustering.

Note:

In quadratic probing, if the hash table size is m , then

the no of location searched for finding empty location using quadratic probing is atmost $\frac{m+1}{2}$

Eg: In previous example for $m=11$,

we can search only $\frac{m+1}{2} = 6$ locations /
(i.e., $0, 2, 3, 5, 6, 7$)

→ Hence only half of the hash table is searched.

Note: To get more no of locations for search, it is better to have hash table of prime size.

Double Hashing:

→ In double hashing, the increment factor is not constant as in linear and quadratic but depends on the key.

The increment factor is hash function.

so it is called double hashing.

→ let $h(x) = x \bmod m$

The formula for double hashing is

$$H(x, i) = [h(x) + i(h'(x))] \bmod m$$

where i : probe number (collision number,
i.e., $i=0 \Rightarrow$ no collision)

$h(x)$: primary hash function

$h'(x)$: secondary hash function.

Search for empty locations is $h(x)$

$$[h(x) + 1 * h'(x)] \bmod m$$

$$[h(x) + 2 * h'(x)] \bmod m$$

$$[h(x) + 3 * h'(x)] \bmod m$$

⋮

Eg: Let $h(x) = x \bmod 11$

$$h'(x) = 7 - x \bmod 7$$

$$H(x, i) = [h(x) + i * h'(x)] \bmod m$$

(key)x	46	28	21	35	57	39	19	50
h(x)	2	6	10	9	8	1	3	7

$$\rightarrow h(46) = 46 \bmod 11 = 2$$

$$\rightarrow h(28) = 28 \bmod 11 = 6$$

$$\rightarrow h(21) = 21 \bmod 11 = 10$$

$$\rightarrow h(35) = 35 \bmod 11 = 2 \text{ (collision)}, h'(35) = 7 - 35 \bmod 7 = 7$$

$$H(35, 1) = (2 + 1 * 7) \bmod 11 = 9$$

$$\rightarrow h(57) = 57 \bmod 11 = 2$$

$$h'(57) = 7 - 57 \bmod 7 = 7 - \frac{1}{8} = 6$$

$$H(57,1) = (2 + 1 * 6) \bmod 11 = 8$$

$$\rightarrow h(39) = 39 \bmod 11 = 6$$

$$h'(39) = 7 - 39 \bmod 7 = 7 - 4 = 3$$

$$H(39,1) = (6 + 1 * 3) \bmod 11 = 9$$

$$H(39,2) = (6 + 2 * 3) \bmod 11 = 1$$

$$\rightarrow h(19) = 19 \bmod 11 = 8$$

$$h'(19) = 7 - 19 \bmod 7 = 2$$

$$H(19,1) = (8 + 1 * 2) \bmod 11 = 10$$

$$H(19,2) = (8 + 4) \bmod 11 = 1$$

$$H(19,3) = (8 + 6) \bmod 11 = 3$$

\therefore 3 probes for 19

↳ collisions

$$\rightarrow h(50) = 50 \bmod 11 = 6$$

$$h'(50) = 7 - 50 \bmod 7 = 7 - 1 = 6$$

$$H(50,1) = (6 + 6) \bmod 11 = 1$$

$$H(50,2) = (6 + 12) \bmod 11 = 2$$

\therefore 2 probes for 50

Note: Double hashing

\rightarrow The problem of secondary clustering is resolved in double hashing because the keys that have the same hash address probe different sequence of locations.

\rightarrow The disadvantage of double hashing is that we need to compute two hash functions and hence it takes more time compared to linear and quadratic probing.

Note:

while computing secondary hash function $h'(x)$, we must keep the following points in our mind.

- $h'(x)$ does not give value '0'.
- The value of $h'(x)$ has to be relatively prime to the size of hash table. (\because to make distribution uniform)

* * Q37 Consider a double hashing scheme in which the primary hash function is $h_1(k) = k \bmod 23$ and the necessary secondary hash function is $h_2(k) = 1 + (k \bmod 19)$. Assume that the size of hash table is 23. Then the address returned by probe 1 in probe sequence (assume that the probe sequence begins at probe 0) for key value $k=90$ is _____

Sol:

$$\therefore h_1(90) = 90 \bmod 3 = 21$$

$$h_2(90) = 1 + (90 \bmod 19) = 1 + 14 = 15$$

$$h(x) = b(21 + 1 * 15) \bmod 23$$

↓ ↴ size of hash table
probe 1

$$= 36 \bmod 23 = 13$$

Load factor: If hash table size is ' m ' and the no of keys which are having to hashtable = n .

$$\Rightarrow \text{Load factor, } \lambda = \frac{n}{m}$$

In general if $\lambda \leq 1$, then only we can apply linear probing or quadratic probing or double hashing

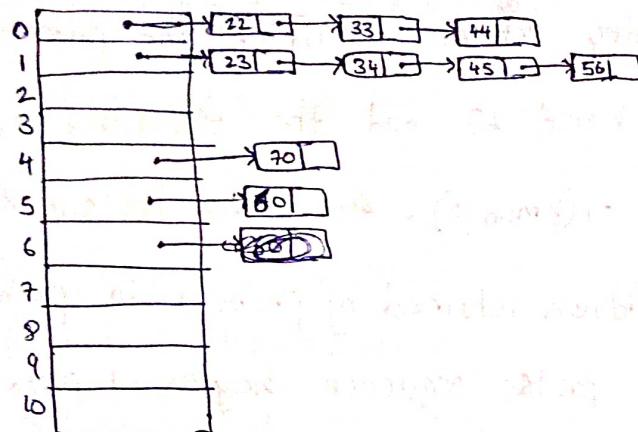
Separate Chaining

→ Separate Chaining can be applied for any value of load factor.

$$h(x) = x \bmod 11$$

$x: 22 \quad 33 \quad 44 \quad 23 \quad 34 \quad 45 \quad 56 \quad 60 \quad 70$

$$h(x):$$



Here each index hold
Starting address of the
list

Disadvantage:

→ More space is required for pointers.

Linear Probing

$$(h(k)+i) \bmod m$$

Quadratic Probing

$$(h(k)+i^2) \bmod m$$

Double Hashing

$$(h(k)+i \cdot h'(k)) \bmod m$$

Separate Chaining

$$h(x) = x \bmod m$$

→ Primary clustering

Secondary clustering

works slow

More space is required

→ $O(n)$ for insertion & searching in worst case

$O(n)$
(worst case)

$O(n)$ length of pointer

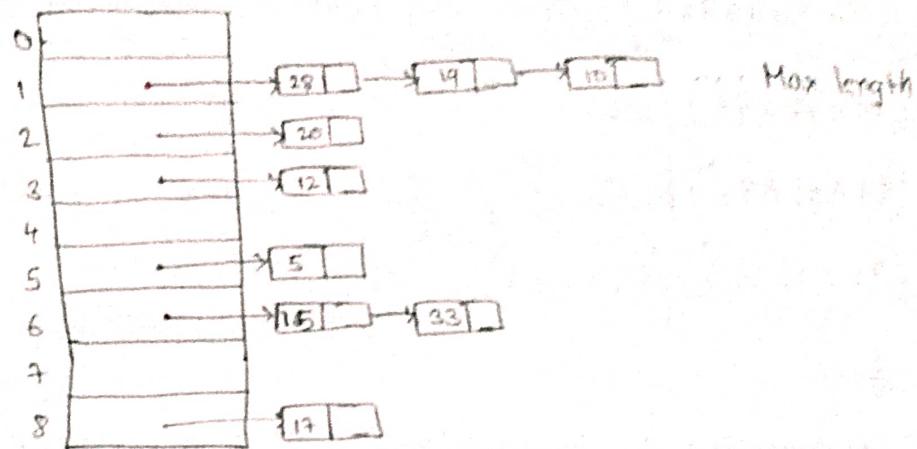
→ Deletion is difficult
(Rehashing is required)

Deletion is difficult.
(rehashing is req)

Deletion is difficult
(rehashing is req)

Deletion is easy

P/2



$$\text{Max length} = 3$$

$$\text{Min length} = 0$$

$$\text{avg length} = \frac{\text{total keys}}{\text{no of slots}} = \frac{9}{9} = 1$$

Note :

→ If hash indices are $0, 1, 2, 3, \dots, m-1$

then we use hash function

$$h(x) = x \bmod m$$

→ If hash indices are $1, 2, 3, \dots, m$

then we use hash function

$$h(x) = (x \bmod m) + 1$$

because $x \bmod m$ never return hash index m .

P/1

hash index address range ... 1 to 1000

⇒ Hash function is ~~x~~ $(x \bmod m) + 1$

Q33 Consider a hash table with 100 slots. Collisions are resolved using chaining. Assuming simple uniform hashing, what is the probability that the first 3 slots are unfilled after the first 3 insertions?

- a) $(97 \times 97 \times 97) / 100^3$
 b) $(99 \times 98 \times 97) / 100^3$
 c) $(97 \times 96 \times 96) / 100^3$
 d) $(97 \times 96 \times 95) / (8! \times 100^3)$

Sol:

Every insertion is independent of each other.

\therefore probability that the inserted key doesn't fall into

$$\text{First 3 slots} = \frac{100-3}{100} = \frac{97}{100}$$

$$\therefore \text{required probability} = \left(\frac{97}{100}\right)^3$$

\therefore opt (a)

- Q39) which one of the following hash function on integers will distribute keys most uniformly over 10 buckets numbered 0 to 9 for i ranging from 0 to 2020?

- a) $h(i) = i^2 \bmod 10$
 b) $h(i) = i^3 \bmod 10$
 c) $h(i) = (11 * i^2) \bmod 10$
 d) $h(i) = (12 * i) \bmod 10$

Sol:

<u>i</u>	<u>$i^2 \bmod 10$</u>	<u>$i^3 \bmod 10$</u>	<u>$(11 + i^2) \bmod 10$</u>	<u>$(12 * i) \bmod 10$</u>
0	0	0	0	0
1	1	1	1	2
2	4	8	6	4
3	9	7	9	6
4	6	4	6	8
5	5	5	5	0
6	6	6	6	2
7	9	3	9	4
8	4	2	4	6
9	1	9	1	8
10	0	0	0	0
11	1	1	1	2
...

$i^3 \bmod 10$ hashes to every hash index for keys 0 to 9
 " " 10 to 19
 " " 20 to 29

$\therefore i^3 \bmod 10$ distributes keys uniformly.

∴ opt (b)

24/09/20

- (Q40) Consider a hash table with n buckets where external chaining is used to resolve collisions. The hash function is such that the probability that a key value is hashed to a particular bucket is $\frac{1}{n}$. The hash table is initially empty and k distinct values are inserted in the table.

a) what is the probability that bucket number 1 is empty after the k^{th} insertion

Sol:

Probability that a key is not inserted into bucket 1 is $\frac{n-1}{n}$

$$\therefore \text{req probability} = \left(\frac{n-1}{n}\right)^k$$

b) what is the probability that no collision has occurred in any of the k insertions?

Sol:

total no of ways to place k keys into n buckets is n^k

no of ways to place k keys into n buckets such that there is no collision = $n P_k$

$$\therefore \text{req probability} = \frac{n P_k}{n^k}$$

c) what is the probability that the first collision occurs at the k^{th} insertion?

Sol:

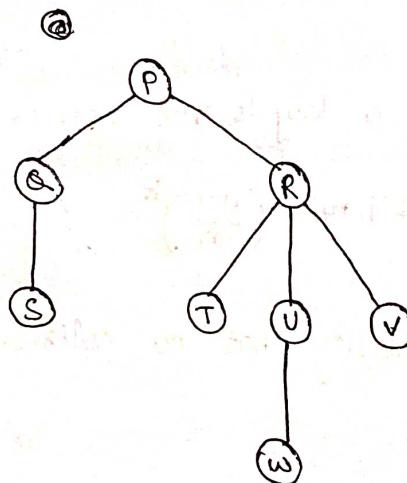
Probability that there is no collision for first $k-1$ insertions

$$= \left(\frac{n}{n} \right)^{k-1} \frac{n p_{k-1}}{n^{k-1}}$$

Probability that k^{th} insertion is a collision is $\frac{k-1}{n}$

$$\therefore \text{req probability} = \left(\frac{n-1}{n} \right)^{k-1} \frac{n p_{k-1}}{n^{k-1}} \cdot \frac{k-1}{n} = \frac{n p_{k-1}(k-1)}{n^k}$$

- (Q41) Consider the following rooted tree with the vertex P labeled as root



The order in which the nodes are visited during inorder traversal is

- a) SQPTRWUV
- b) SQTURWU
- c) SQPTTWUVR
- d) SQPTRUWV

Sol:

sol?
Here

Q & V

Graph

- (i) De
(ii) Br

Represen

G(V)

A

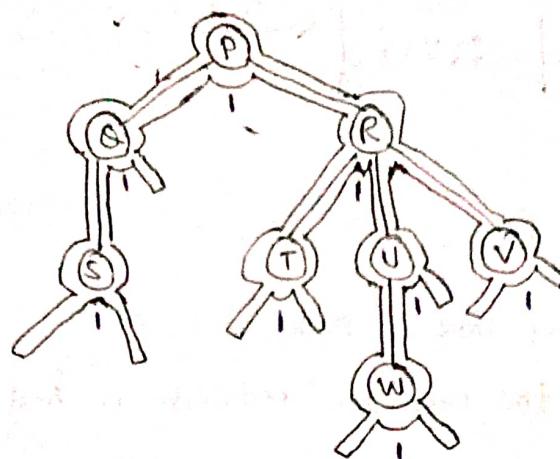
f

Weight

A	10
B	10

Sol:

Here S & W must be treated at leftmost children of Q & V respectively

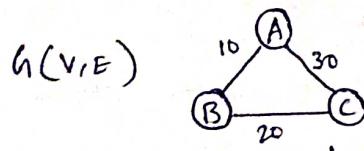


Result: S Q P T R W U V

Graph Traversals:

- (i) Depth First Search (DFS)
- (ii) Breadth First Search (BFS)

Representation of Graph:



Adjacency Matrix [$O(v^2)$]

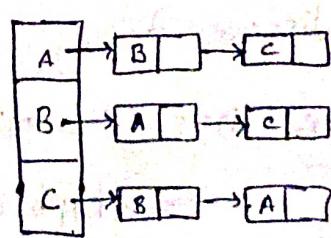
weighted
graph

	A	B	C
A	0	10	30
B	10	0	20
C	30	20	0

unweighted
graph

	A	B	C
A	0	1	1
B	1	0	1
C	1	1	0

Adjacency List [$O(v+E)$]



Note :

Representation	TC
Adjacency Matrix	$O(V^2)$
Adjacency List	$O(V+E)$

Depth First Search (DFS) :

$d[u]$: Discovery time of node ' u '.

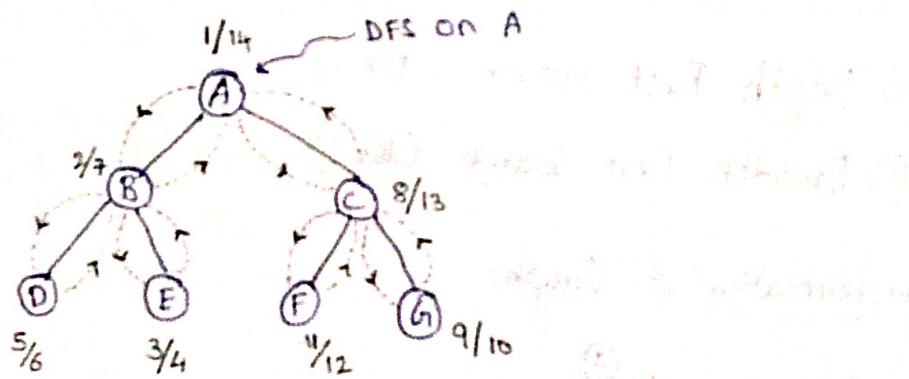
i.e., the time the node ' u ' is first visited.

$f[u]$: finishing time of a node ' u '

i.e., the time the node ' u ' is last visited.

→ At every node, let us follow the general convention $\frac{d[u]}{f[u]}$

Also finishing time of a node is greater than discovery time of that node.



DFS : A B E D C G F

→ we visit only those nodes which are yet to be discovered.

→ Here DFS is applied only once. i.e., on a graph is connected.

Note :

(i) If

we

no

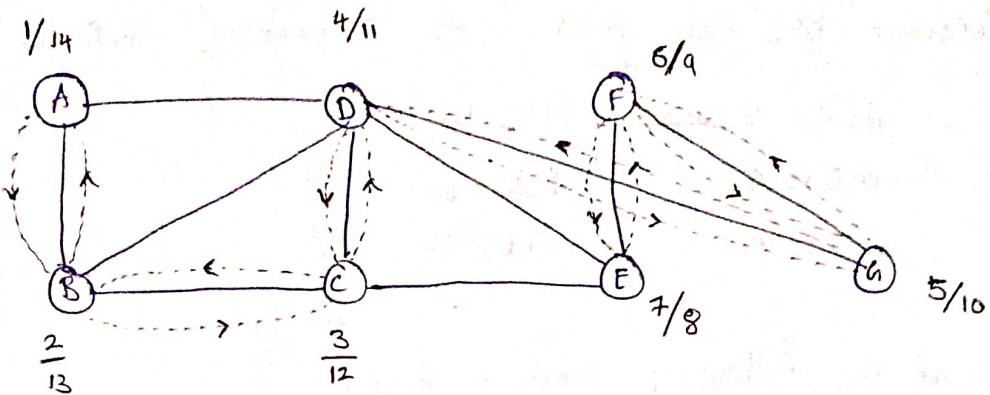
(ii) It

(iii) The

(iv)

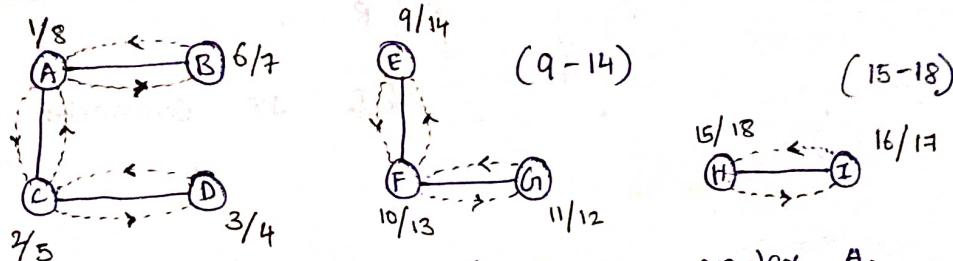
If

B



DFS: A B C D G F E

Consider the below disconnected graph



(1-8) Now let us apply DFS on vertex A.

DFS: Here we applied DFS 3 times (on A, on E, on H)
 \Rightarrow the graph is disconnected with 3 connected components.

Note:

- (i) In DFS nodes get visited in deeper direction until we reach either a leaf node or a node whose adjacent nodes have already been visited.
- (ii) It uses stack data structure.
- (iii) The time complexity for DFS using adjacency list is $O(V+E)$
adjacency matrix is $O(V^2)$
- (iv) If the finishing time of starting vertex is greater than finishing time of remaining all vertices, then the graph is said to be connected.

Q42 Consider DFS on graph with 3 vertices P, Q, R.

$$d(P) = 5 \text{ units} \quad f(P) = 12$$

$$d(Q) = 6 \quad f(Q) = 10$$

$$d(R) = 14 \quad f(R) = 18$$

which of the following stmts is true?

- a) graph is connected.
- b) two connected components, P & R are connected.
- c) two connected components, Q & R are connected.
- d) two connected components, P & Q are connected.

Sol:

P is discovered first and P's finishing time is greater than finishing time of Q only.

\Rightarrow P & Q are connected.

R is discovered at 14 and finished at 18.

\therefore R forms a connected component.

P/6

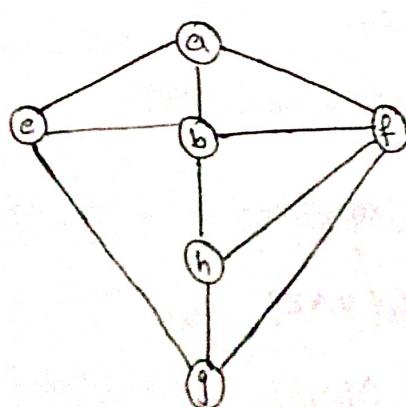
Among the sequences

I) a b e g h f

II) a b f e h g

III) a b f a h g e

IV) a f g h b c

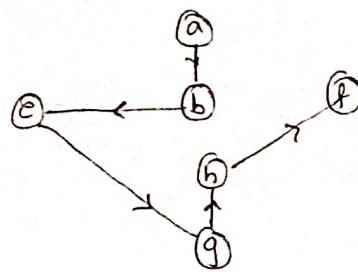


which of the above are valid DFS traversals

- a) I, II, IV
- b) I & IV
- c) II, III, IV
- d) I, III, II

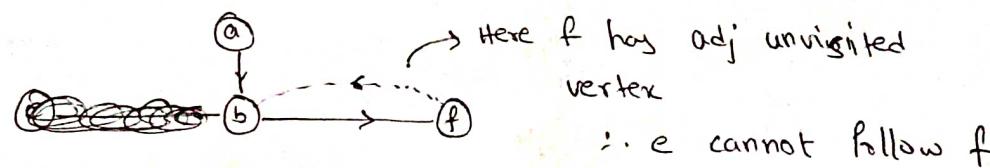
Sol:

a) a b e g h f



∴ valid

b)



∴ e cannot follow f

∴ Invalid

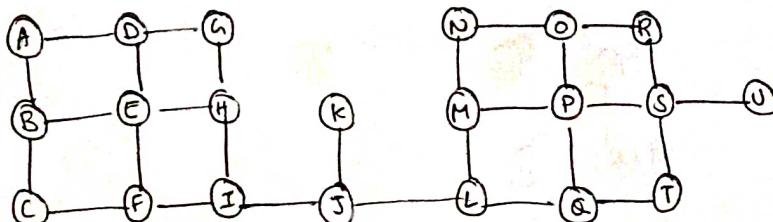
c) $a \rightarrow b \rightarrow f \rightarrow h \rightarrow g \rightarrow e$

∴ valid

d) silly ④ is valid

∴ opt ④

P/6



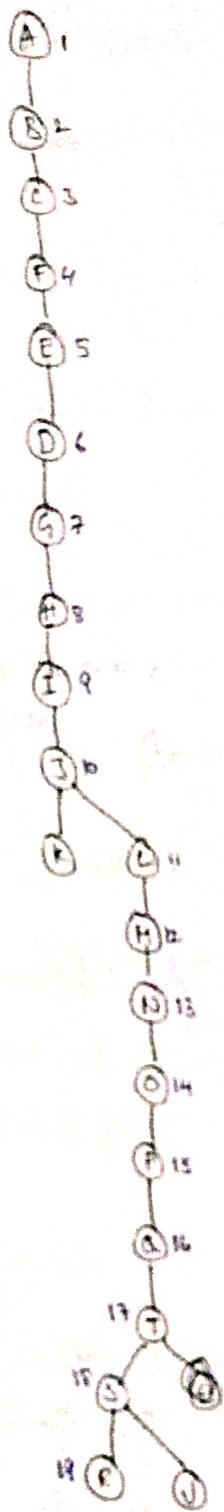
Consider starting ~~near~~ DFS on A

~~A = B = C = F = E = D = G = H = I = J = L = M = N = O = P = Q = T = S = U~~

∴ Max recursion depth = 19

Recursion depth is no of recursive calls from the top most recursive call to the bottom most one.

P1

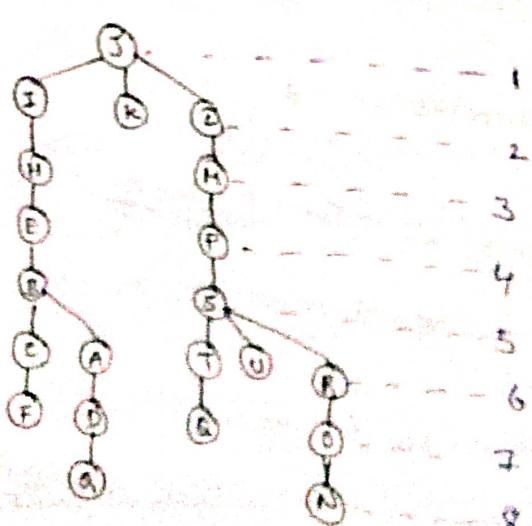


Max recursion depth = 19

we

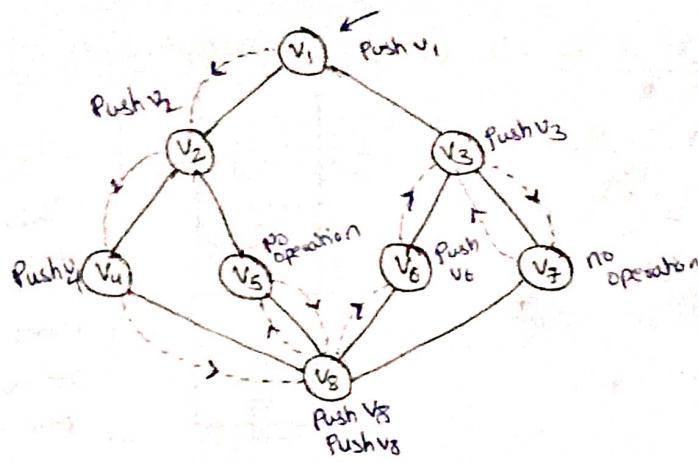
un

P2



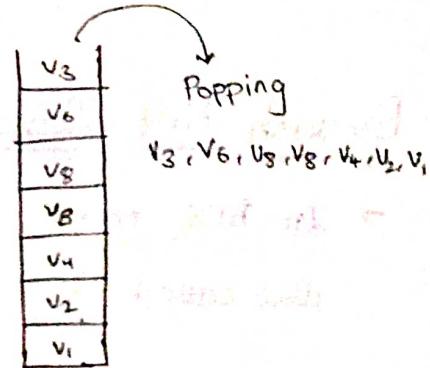
Min recursion depth = 8

P3



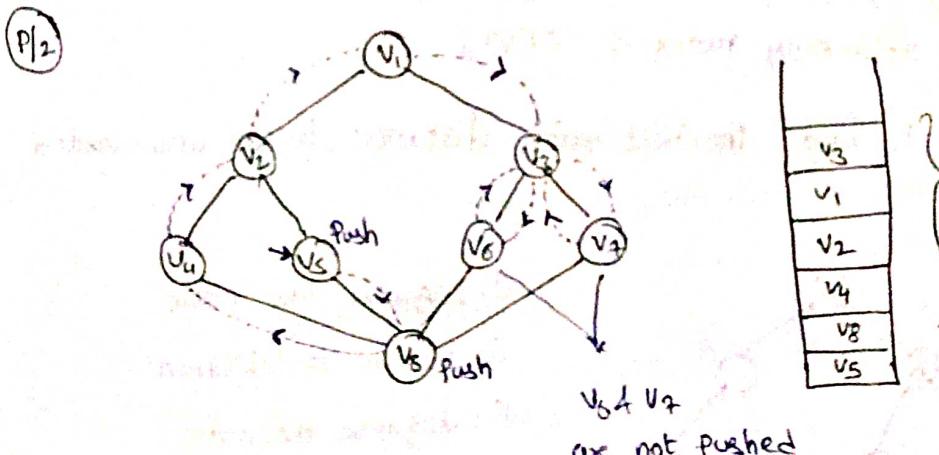
whenever we visit a node, for DFS traversal, we push a node into stack iff it has atleast one adjacent node which is unvisited.

$$\underline{DFS} = V_1, V_2, V_4, V_8, V_5, V_6, V_3, V_7$$



We pop a node, if there is no any unvisited adjacent node left.

$\therefore u_8$ is pushed more than once.

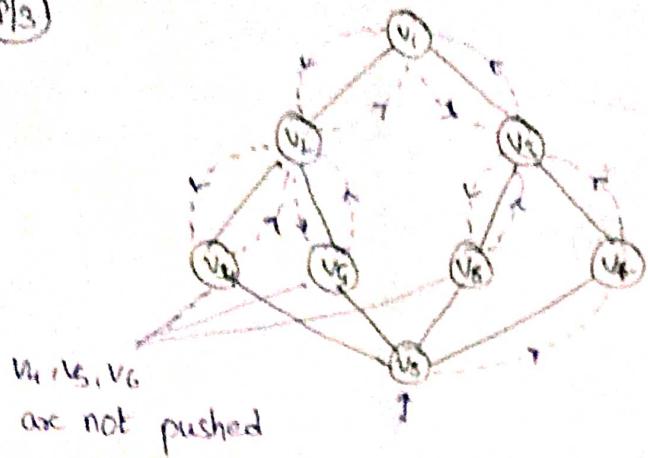


This stack doesn't show nodes that are pushed more than once

DFS: $v_5, v_8, v_4, v_2, v_1, v_3, v_6, v_7$

\therefore 2 nodes

P13



v ₂
v ₁
v ₃
v ₄
v ₅

This stack
doesn't store
nodes that
are pushed
more than once.

DFS: v₈ v₉ v₂ v₁ v₂ v₄ v₅ v₆

1. 3 nodes

Breadth First Search (BFS):

→ In BFS, nodes get visited level by level. so BFS is also called level order traversal.

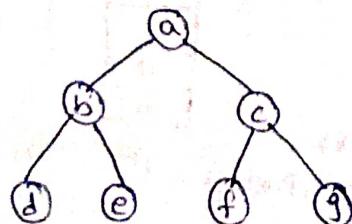
→ To implement BFS, we can use queue data structure.

→ Time complexity for finding BFS using

adjacency list is $O(V+E)$

adjacency matrix is $O(V^2)$

→ BFS is used to find shortest path distance in an unweighted graph.



- (i) Enqueue the node
- (ii) Explore its children
- (iii) dequeue the node.

Applying BFS on vertex 'a'.

BFS(a) ↪

a	b	c
---	---	---

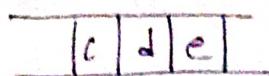
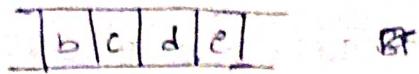
BFS: a b c ↪

• now enqueue b & c

now we have no more unvisited children of a.

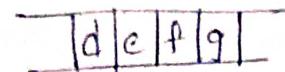
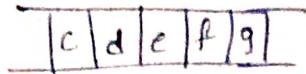
∴ dequeue(a): [b c]

Now explore unvisited adjacent nodes of 'b'.



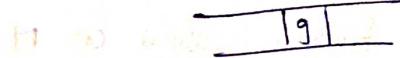
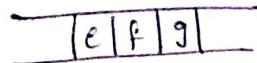
BFS: a b c d e

Now explore unvisited adjacent nodes of 'c'.



BFS: a b c d e f g

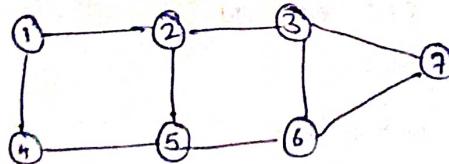
Now explore children of d, e, f, g.



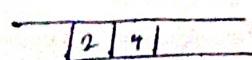
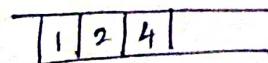
... empty queue

∴ BFS: a b c d e f g

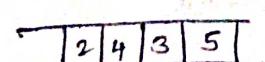
E:



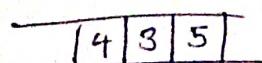
Applying BFS on '1'



BFS: 1 2 4



BFS: 2 4 3 5



3	5
---	---

3	5	6	7
---	---	---	---

5	6	7
---	---	---

6	7
---	---

7

--

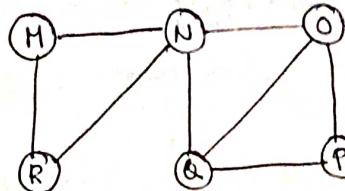
--

BFS: 1 2 4 3 5 6 7

empty queue

∴ BFS traversal: 1 2 4 3 5 6 7

P/10



a) M N O P Q R \Rightarrow BFS is called on M

O is not adjacent to M and O is visited before R

which is adjacent to M.

\therefore not valid

b) N Q M P O R \Rightarrow BFS on N



P is visited before R

but R is adjacent to P

\therefore not valid

c) Q M N R O P \Rightarrow BFS on Q

M is not adjacent to Q

\therefore invalid

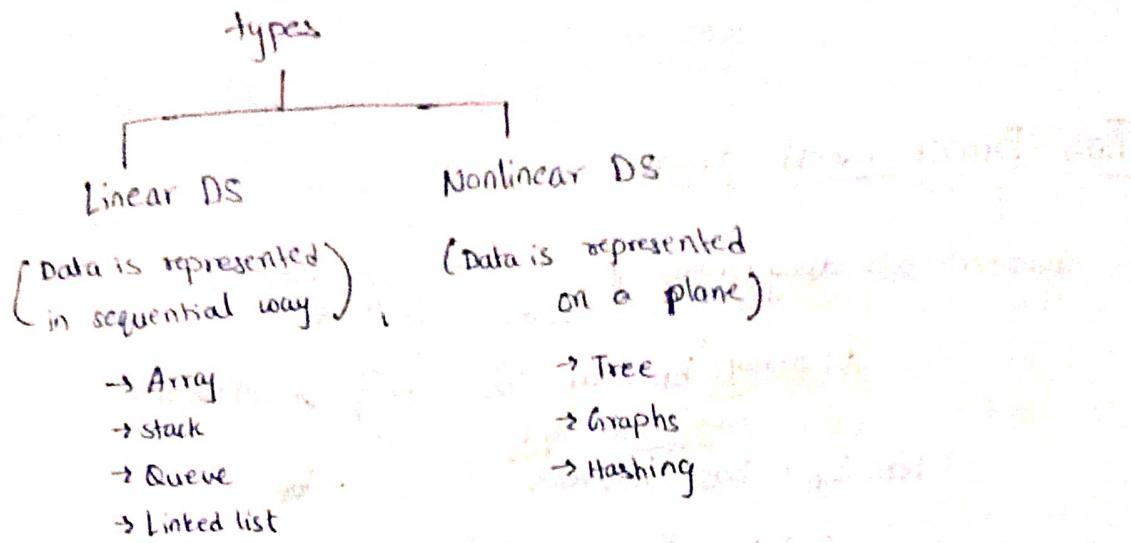
d) P O Q N M R



\therefore valid

Data Structure

Data structure deals with physical representation of data
It gives information about storage representation of data, accessing data from the memory in efficient manner.



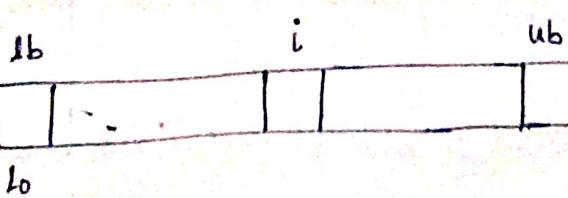
Arrays

Formula for 1-D array:

Consider $A[lb \dots ub]$ of elements $lb \rightarrow$ lower boundary (or) lower index
 $ub \rightarrow$ upper boundary (or) upper index

let $c = \text{size data at an index}$

L_0 = Base address of array



$$\Rightarrow \text{Loc}(A[i]) = L_0 + (\text{no of elements crossed before reaching } i) \times c$$

$$\boxed{\text{Loc}(A[i]) = L_0 + (i - lb) * c}$$

P/1

$$lb = -5 \quad ub = 5$$

$$L_0 = 1000 \quad c = 2$$

$$\begin{aligned} \text{Loc}(A[0]) &= L_0 + (i-lb) * c \\ &= 1000 + (0 - (-5)) * 2 \\ &= 1010 \end{aligned}$$

Two Dimensional array:

Consider 2D array A

A: array $[b_1, \dots, u_1, b_2, \dots, u_2]$ of elements
 lower bound of rows UB of rows

let 'L₀' : Base address

c : size of an element

$$r_1 : \text{no of rows} = u_1 - b_1 + 1$$

$$r_2 : \text{no of columns} = u_2 - b_2 + 1$$

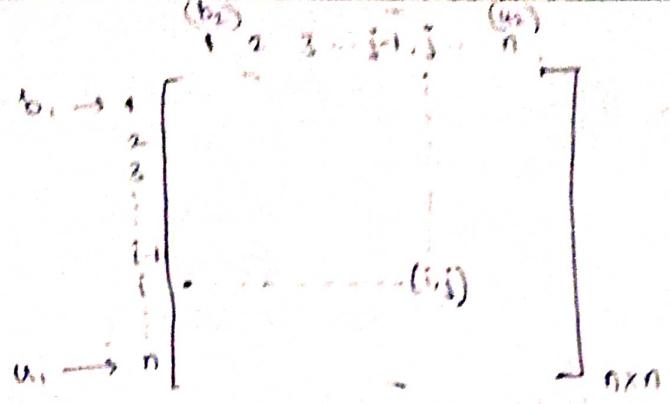
b ₂	1	2	3	4	5	u ₂
b ₁ →	a ₁₁	a ₁₂	a ₁₃	a ₁₄	a ₁₅	
2	a ₂₁	a ₂₂	a ₂₃	a ₂₄	a ₂₅	
3	a ₃₁	a ₃₂	a ₃₃	a ₃₄	a ₃₅	
4	a ₄₁	a ₄₂	a ₄₃	a ₄₄	a ₄₅	
u ₁ → 5	a ₅₁	a ₅₂	a ₅₃	a ₅₄	a ₅₅	

Row Major Ordering (RMO): (By default we consider RMO)

a ₁₁ a ₁₂ a ₁₃ a ₁₄ a ₁₅	a ₂₁ a ₂₂ a ₂₃ a ₂₄ a ₂₅	a ₃₁ a ₃₂ a ₃₃ a ₃₄ a ₃₅	a ₄₁ a ₄₂ a ₄₃ a ₄₄ a ₄₅	a ₅₁ a ₅₂ a ₅₃ a ₅₄ a ₅₅
---	---	---	---	---

Column Major Ordering (CMO):

a ₁₁ a ₂₁ a ₃₁ a ₄₁ a ₅₁	a ₁₂ a ₂₂ a ₃₂ a ₄₂ a ₅₂	a ₁₃ a ₂₃ a ₃₃ a ₄₃ a ₅₃	a ₁₄ a ₂₄ a ₃₄ a ₄₄ a ₅₄	a ₁₅ a ₂₅ a ₃₅ a ₄₅ a ₅₅
---	---	---	---	---



Finding Loc(A[i][j]):

RMO:

$$\text{Loc}(A[i][j]) = L_0 + (\text{no of rows crossed before reaching } i\text{th row})^* (\text{elements per row})^* c$$

$$+ (\text{no of elements crossed in } i\text{th row before reaching } j\text{th element})^* c$$

$$\text{Ans: } \text{Loc}(A[i][j]) = L_0 + [(i-1)^* n + (j-1)]^* c$$

$$\boxed{\text{Loc}(A[i][j]) = L_0 + \underbrace{[(i-b_1)^*(u_2-b_2+1) + (j-b_2)]}_\text{no of elements per row}^* c}$$

$$\text{no of elements per row} = \text{no of columns}$$

CMD:

$$\text{Loc}(A[i][j]) = L_0 + (\text{no of columns crossed before reaching } j\text{th column})^* (\text{elements per column})^* c$$

$$+ (\text{no of elements to be crossed in } j\text{th column to reach element in } i\text{th row})^* c$$

$$= L_0 + [(j-1)^* n + (i-1)]^* c$$

$$\boxed{\text{Loc}(A[i][j]) = L_0 + \underbrace{[(j-b_2)^*(u_1-b_1+1) + (i-b_1)]}_\text{no of elements per column = no of rows}^* c}$$

$$\text{no of elements per column} = \text{no of rows}$$

P/2

A: array [-2...2, 3...7]

$$b_1 = -2 \quad u_1 = 2$$

$$b_2 = 3 \quad u_2 = 7$$

$$L_0 = 1000$$

$$\text{no of rows} = u_1 - b_1 + 1 = 2 - (-2) + 1 = 5$$

$$\text{no of columns} = u_2 - b_2 + 1 = 7 - 3 + 1 = 5$$

$$c = 2$$

RMO:

$$\text{Loc}(A(0,5)) = L_0 + ([0 - (-2)] * 5 * 2) + (5 - 3) * 2$$

$$= 1000 + 20 + 4$$

$$= 1024$$

CMO:

$$\text{Loc}(A(0,5)) = L_0 + [(5 - 3) * (\text{no of rows}) * 2] + (0 - (-2)) * 2$$

$$= 1000 + 2 * 5 * 2 + 2 * 2$$

$$= 1024$$

P/3

A: array [1...10][1..15]

$$b_1 = 1 \quad u_1 = 10$$

$$\text{no of rows} = u_1 - b_1 + 1 = 10$$

$$b_2 = 1 \quad u_2 = 15$$

$$\text{no of cols} = u_2 - b_2 + 1 = 15$$

$$c = 1$$

$$L_0 = 100$$

given RMO

$$A[i][j] = 100 + (i-1) * (15) * 1 + (j-1) * 1$$

↓
no of cols

$$= 100 + 15(i-1) + j-1$$

$$= 15i + j + 84$$

Lower

RI

no

RMO for

Lower Triangular matrix:

$$\begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

RMO:

a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	a_{33}
Row1		Row2		Row3	

CMO:

a_{11}	a_{21}	a_{31}	a_{22}	a_{32}	a_{33}
----------	----------	----------	----------	----------	----------

no of non-zero elements in 1st row = 1

" " " " 2nd row = 2

" " " " 3rd row = 3

n^{th} row = n

\therefore no of non-zero elements in a lower triangular matrix

$$\text{of order } n \times n = \frac{n(n+1)}{2}$$

RMO for lower triangular matrix:

$$\text{Loc}(A[i][j]) = L_0 + \left(\text{no of non-zero elements in } (i-1) \text{ row} \right) * C + \left(\text{no of non-zero elements crossed in } i^{\text{th}} \text{ row} \right) * C$$

$$\text{Loc}(A[i][j]) = L_0 + \left[(\sum(i-1) + (j-1)) * C \right], \text{ if } i \geq j$$

$$= 0 \quad , \text{ if } i < j$$

CMO formula for lower triangular matrix:

$$\begin{aligned}
 \text{Loc}(A[i][j]) &= L_0 + \left(\begin{array}{l} \text{no of non-zero} \\ \text{elements in } (j-1) \\ \text{columns} \end{array} \right) * c + \left(\begin{array}{l} \text{no of non-zero} \\ \text{elements crossed} \\ \text{in } j\text{th column} \\ \text{to reach } i\text{th row} \end{array} \right) * c \\
 &= L_0 + [n + (n-1) + \dots + (n-(j-1)+1)] * c + \underbrace{[i - (j-1)]}_{\begin{array}{l} \text{no of} \\ \text{zeroes in} \\ \text{jth col} \end{array}} * c \\
 &= L_0 + [n + (n-1) + \dots + (n-(j-2))] * c + \underbrace{[i - (j-1)]}_{\begin{array}{l} \text{no of non-zero} \\ \text{elements crossed} \\ \text{in } j\text{th column} \end{array}} * c \\
 &= L_0 + \cancel{[(j-1)n + \sum(j-2) + i - (j-1)]} * c \\
 &= L_0 + \cancel{[n(j-1) + i - \sum(j-1)]} * c \\
 &= \cancel{L_0} + \cancel{n(j-1)} + \cancel{i - \sum(j-1)} * c \\
 &= L_0 + [n(j-1) - \sum(j-2) + i - j] * c
 \end{aligned}$$

$$\boxed{\Delta \text{Loc}(A[i][j]) = L_0 + \left[\frac{n(j-1)}{2} - \frac{(j-2)(j-1)}{2} + i - j \right] * c}$$

Upper Triangular matrix:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$

elements in
 j-1 columns zero elements
 in j-1 cols non-zero
 elements crossed
 in jth column

RMO:

$$\boxed{a_{11} \ a_{12} \ a_{13} \ a_{14} \ | \ a_{22} \ a_{23} \ a_{24} \ | \ a_{33} \ a_{34} \ | \ a_{44}}$$

CMD:

a_{11}	$a_{12} \ a_{22}$	$a_{23} \ a_{22} \ a_{33}$	$a_{34} \ a_{24} \ a_{34} \ a_{44}$
----------	-------------------	----------------------------	-------------------------------------

- From the formulae of $\text{Loc}(A[i][j])$ in lower triangular matrix replace i by j and j by i.

RHO:

$$\text{Loc}(A[i][j]) = L_0 + \left[n(i-1) - \frac{(i-1)(i-2)}{2} + (j-i) \right] * c$$

CMD:

$$\text{Loc}(A[i][j]) = L_0 + \left[\sum(j-1) + (i-1) \right] * c$$

Tridiagonal Matrix:

The matrix in which the elements except principle diagonal, elements above & below principle diagonal, rest of the elements are zeroes.

i.e.,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} \rightarrow \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix}$$

RHO:

$a_{11} \ a_{12}$	$a_{21} \ a_{22} \ a_{23}$	$a_{32} \ a_{33} \ a_{34}$	$a_{43} \ a_{44} \ a_{45}$	$a_{54} \ a_{55}$
-------------------	----------------------------	----------------------------	----------------------------	-------------------

In tridiagonal matrix, except first row & last row, every other row contains 3 non-zero elements.

$$\text{Loc}(A[i][j]) = L_0 + \left(\begin{array}{l} \text{no of non-zero} \\ \text{elements crossed} \\ \text{in } (i-1) \text{ rows} \end{array} \right) * c + \left(\begin{array}{l} \text{no of non-zero} \\ \text{elements crossed} \\ \text{in } i^{\text{th}} \text{ row} \end{array} \right) * c$$

$$= L_0 + [3(i-1) - 1] * c + [(j-1) - (i-2)] * c$$

↓ ↓ ↓
 every row has total no of zeroes
 3 non-zero elements elements in i^{th} row
 except 1st row :
 1st row has only 2
 non-zero elements

$$= L_0 + [3(i-1) - 4 + j - i + 1] * c$$

RMO:

$$\Rightarrow \boxed{\text{Loc}(A[i][j]) = L_0 + (2i + j - 3), \quad 0 \leq |i-j| \leq 1}$$

$$= 0 \quad , \quad |i-j| > 1$$

CMO for tridiagonal matrix:

$$\text{Loc}(A[i][j]) = L_0 + \left(\begin{array}{l} \text{no of non-zero} \\ \text{elements crossed} \\ \text{in } (j-1) \text{ columns} \end{array} \right) * c + \left(\begin{array}{l} \text{no of non-zero} \\ \text{elements crossed} \\ \text{in } j^{\text{th}} \text{ row} \end{array} \right) * c$$

$$= L_0 + [3(j-1) - 1] * c + [(i-1) - (j-2)] * c$$

$$= L_0 + [3j - 4 + i - j + 1] * c$$

$$= L_0 + [i + 2j - 3] * c$$

$$\boxed{\text{Loc}(A[i][j]) = L_0 + (i + 2j - 3) * c, \quad \text{for } 0 \leq |i-j| \leq 1}$$

$$= 0 \quad , \quad |i-j| > 1$$

P/8

lower diagonal	principle diagonal	upper diagonal
$a_{21} a_{32} a_{43} \dots a_{n,n-1}$	$a_{11} a_{22} a_{33} \dots a_{nn}$	$a_{12} a_{23} a_{34} \dots a_{n-1,n}$

$\underbrace{\hspace{1cm}}_n$ $\underbrace{\hspace{1cm}}_n$ $\underbrace{\hspace{1cm}}_n$

(n-1) elements n elements

it is clear that we need to cross $(n+n-1)$ no of elements to reach upper diagonal element.

To reach an upper diagonal element $a_{i,i+1}$ we need to cross $(i-1)$ upper diagonal elements.

$$\therefore \text{total no of elements crossed} = (2n-1) + (i-1)$$

$$\therefore \text{Loc}(A[i][j]) = L_0 + (2n-1) + (i-1)$$

P/9

Before accessing the elements of 1st column we must cross the elements of all 3 rows.

now for element in i th of 1st column we need to cross $(i-1) - \frac{1}{4}$

this element is already included in row 1
is already included in row 1

$$\therefore \text{no of elements crossed} = 3 \cdot n + i - 2$$

\therefore index of $A[i][j]$ in 1-D array is $3 \cdot n + i - 2$

Before accessing the elements of n th column we must cross 3n elements (3 rows) and $\frac{n}{2}-1$ elements (1st column)

In the last column to access ~~1st~~ elements element in i th row we need to cross $(i-1) - (\frac{n}{2} + 1)$

\downarrow or
elements that '0' ~~and~~ already allocated in 1-D array

total no of elements crossed

$$= 3n + \frac{n}{2} - 1 + (i-1) = \left(\frac{n+1}{2}\right)$$

$$= 3n + \frac{n}{2} - 1 + i - 1 = \frac{7n}{2} - 2$$

$$\therefore L = 3n + i - 3$$

index of $\alpha A[i][j]$ (which is in n^{th} column) is $3n - i - 3$

Toeplitz Matrix:

A $n \times n$ matrix T is a Top Toeplitz matrix iff

$$T(i,j) = T(i-1, j-1) \text{ for all } i > 1, j > 1$$

For a ~~rectangular~~ toeplitz matrix of order n ,

$$\text{no of independent (distinct) elements} = 2n - 1$$

(P/10)

For $i=1$ or $j=1$

$\alpha A[i][j]$ value is independent

Let $\alpha A[1][1] = a$

$$A[1][2] = b$$

$$A[1][3] = c$$

$$A[1][4] = d$$

$$A[1][5] = e$$

$$A[2][1] = f$$

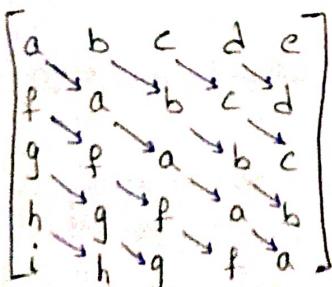
$$A[3][1] = g$$

$$A[4][1] = h$$

$$A[5][1] = i$$

~~is not ok~~

: optimum no of elements stored = 9



Square Bo

A sq

all r

main

a-1

P/11

RHO:

$$\text{Loc}(AE[i][j]) = L_0 + (j-i)^* c \quad \text{for } i \leq j$$
$$= L_0 + (n+i-j-1)^* c \quad \text{for } i > j$$

Square Band Matrix:

A square Band Matrix $D_{n,n}$ is an $n \times n$ matrix in which all non-zero elements lie in a band centered around the main diagonal. The band includes the main diagonal and $a-1$ diagonal below and above the main diagonal.

RPO
P/11

In a ^{square} matrix of order n , ~~as~~ away no of elements in each diagonal as we move ~~to~~ to the main diagonal ~~are~~ are $(n-1), (n-2), \dots$

In $D_{1000,100}$ we have $100-1 = 100$ diagonals other than main diagonal.

$$\begin{aligned} \text{no of elements in band} &= [(n-1) + (n-2) + \dots + (n-100)]^* 2 + 1000 \\ &= 100n - \frac{100(100+1)}{2}^* 2 + 1000 \\ &= 100000 - 10100 + 1000 \\ &= 190900 \end{aligned}$$

P/12

$n - (a-1), n - (a-2), n - (a-3), \dots, n-1, n$ are the ~~element no~~ of elements in each diagonal from lowest diagonal.

No of elements before k^{th} diagonal

$$= [n - (a-1)] + [n - (a-2)] + \dots + [n - (a - (k-1))]$$

$$= [(n-a)+1] + [(n-a)+2] + \dots + [(n-a)+(k-1)]$$

$$= (k-1)(n-a) + \frac{k(k-1)}{2}$$

elements:

$a_{21}, a_{32}, a_{43}, a_{54}$

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

In any lower diagonal the column index starts from '1'

i.e., 'j' starts from 1.

: within k^{th} diagonal to access $A[i][j]$ we need to cross $(j-1)$ elements

⇒ No total no of elements crossed

$$= (k-1)(n-a) + \frac{k(k-1)}{2} + (j-1)$$

Method 2:

Consider $\begin{bmatrix} a & b \\ b & c \end{bmatrix}$

Here ~~k=2~~ $n=2$

$$a-1=1 \Rightarrow a=2$$

let $i=2 j=1$ i.e., we are accessing 'b'.

$$\Rightarrow k=1$$

The elements are stored in 1-D array as

b	a	c
---	---	---

Note

In

a) $L_0 + [0+0+0] = L_0 \checkmark$

b) $L_0 + [0+0+0] = L_0 \checkmark$

c) $L_0 + [0+1+0] = L_0 + 1 \times$

d) $L_0 + [0+0+0] = L_0 \checkmark$

Now let $i=1$ $j=1$ $\Rightarrow k=1$ (main diagonal)

a) $L_0 + [0+1+0] = L_0 + 1 \checkmark$

b) $L_0 + [0+1+0] = L_0 + 1 \checkmark$

c) $L_0 + [0+3+0] = L_0 + 3 \times$

d) $L_0 + [4+1+0] = L_0 + 5 \times$

So taking some other example we can eliminate opt ⑥

Notes:

In general for a Toeplitz matrix, optimum no of elements to be stored in 1-D array = $n + (a-1)(2n-a)$

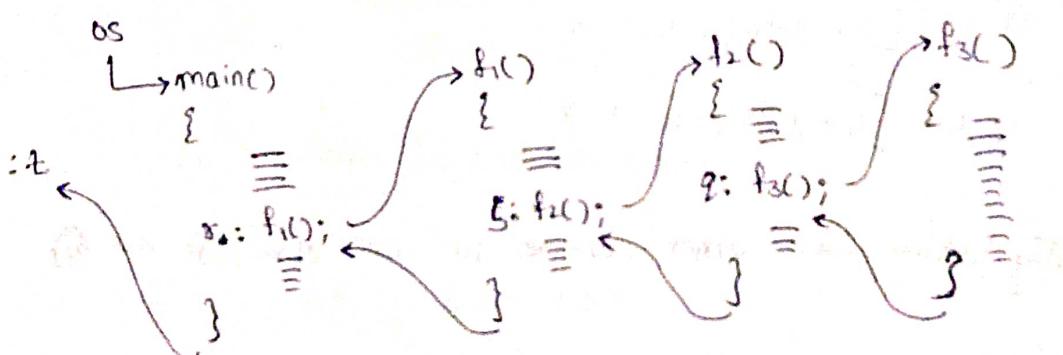
Stack & Queue

stack P
we C
the give

Stack:

- Stack is an ordered list in which insertion & deletion can be performed only at one end i.e., at top end.
- Stack uses Last In First Out.
- Operations on stack
 - * push : Insertion
 - * pop : Deletion

Importance of stack:



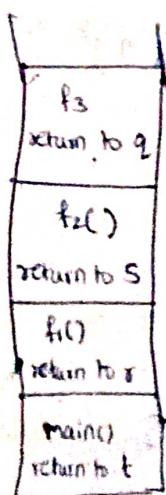
Let 't' return address given by OS to main

Note:

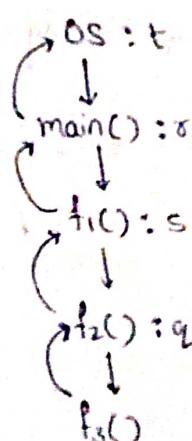
- In ger
- no of

we

one



Stack



(P/5)

- Stack is used to maintain return addresses of procedure calls.

Stack Permutation:

we can generate stack permutation by inserting n keys in the given order but pop off in any order.

Eg: Consider following keys & the stack permutations possible.

1, 2, 3

we print an element after we pop it

1 2 3 : PUSH 1, POP, PUSH 2, POP, PUSH 3, POP

1 3 2 : PUSH 1, POP, PUSH 2, PUSH 3, POP, POP

2 3 1 : PUSH, PUSH, POP, PUSH, POP, POP

2 1 3 : PUSH, PUSH, POP, POP, PUSH, POP

3 1 2 : PUSH, PUSH, PUSH, POP(3), ----- not possible

3 2 1 : PUSH, PUSH, PUSH, POP, POP, POP

Note:

→ In general no of permutations possible with n distinct keys = $n!$

→ no of stack permutation possible with n distinct key where

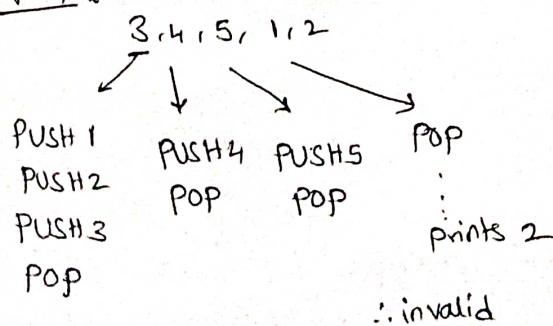
we can push elements in given order and pop off in

any order = $\frac{1}{n+1} 2^n C_n$

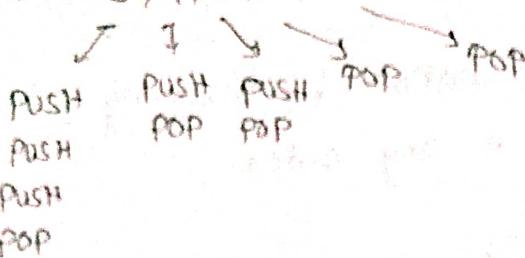
(P/5)

keys: 1, 2, 3, 4, 5

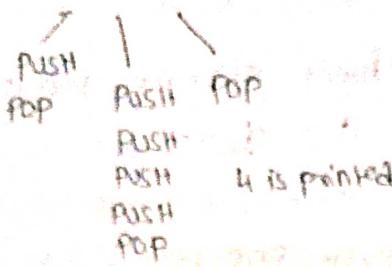
Ques:



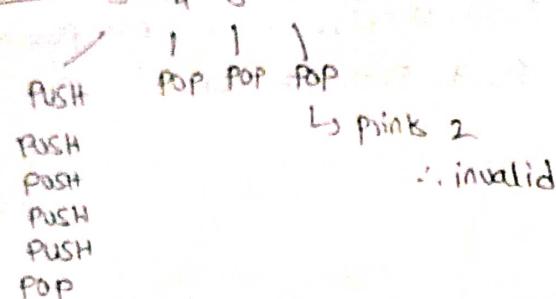
opt b: 3, 4, 5, 2, 1



opt c: 1, 5, 2, 3, 4



9+4: 5 4 3 1 2

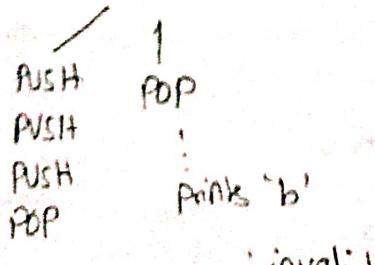


(Q4) A program attempt to generate as many permutations as possible for the string abcd by pushing characters in same order but pop off top character at any time. Which of the following cannot be a stack permutation.

- a) a b c d b) cbad c) dcba d) cabd

93

cpt d



- opt(d)

Recursion:

- Head recursion
- Tail "
- Head & tail recursion
- Excessive recursion
- Nested "

Head Recursion:

Consider

head (int x)

```
{
    if (x > 0)
    {
        head(x-1);
        printf(".1.d", x);
    }
}
```

O/P

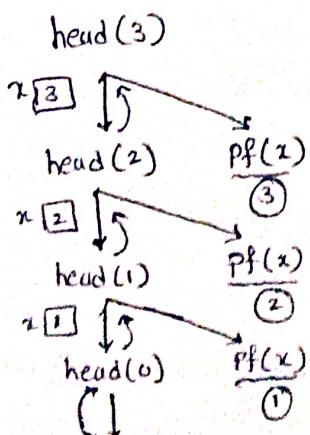
- i) Find return value of head(3)
- ii) find no of function calls for head(n)
- iii) find depth of function calls for head(n)
- iv) If $T(n)$ = no of function calls for head(n)

then what is the recurrence equation?

- v) what is time complexity?
- vi) Solution to the recurrence eqn of $T(n)$ is (iv)

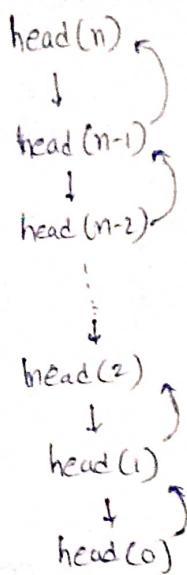
Sol:

j)



∴ O/P: 1, 2, 3

(iii)



head(0)	return to 100
head(1)	return to 100
head(2)	return to 100
head(3)	return to main

return
addresses are
same

Stack

$$\therefore \text{no of function calls} = n+1$$

$$\text{(iii) depth of function calls} = \cancel{n+1}$$

$$\text{(iv) } T(n) = \begin{cases} 1 + T(n-1) & \text{if } n > 0 \\ 1 & \text{if } n \leq 0 \end{cases}$$

(v) time complexity of a recursive function is bounded by
no of recursive calls
 $\Rightarrow O(n)$

$$\text{(vi) } T(n) = \begin{cases} 1 + T(n-1) & \text{if } n > 0 \\ 1 & \text{if } n \leq 0 \end{cases}$$

$$T(n) = 1 + T(n-1)$$

$$= 1 + 1 + T(n-2)$$

$$= 2 + T(n-2)$$

$$= 2 + 1 + T(n-3)$$

$$= 3 + T(n-3)$$

⋮

$$= n + T(n-n)$$

$$= n + T(0)$$

$$\Rightarrow T(n) = n+1$$

25/09/22

Head

Tail Recursion:

tail(int x)

```
{
  if(x>0)
    {
      printf("%d", x);
      tail(x-1);
    }
}
```

i) find o/p of tail(3).

tail(3)

if(n)

tail(2)

③

if(x)

tail(1)

②

if(x)

tail(0)

①

ii) no. of function calls: n+1

iii) depth of function calls = n+1

iv) $T(n) = 1 + T(n-1)$ if $n > 0$
 = 1 if $n \leq 0$

v) time complexity = $O(n)$

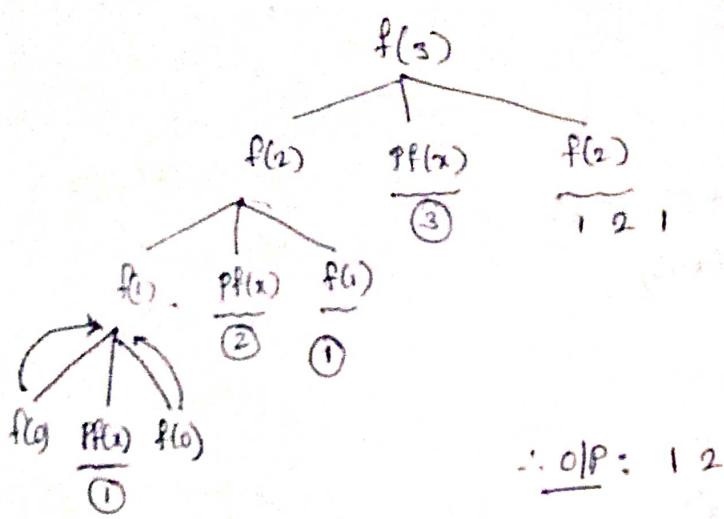
25/09/20

Head - Tail Recursion

f(int x)

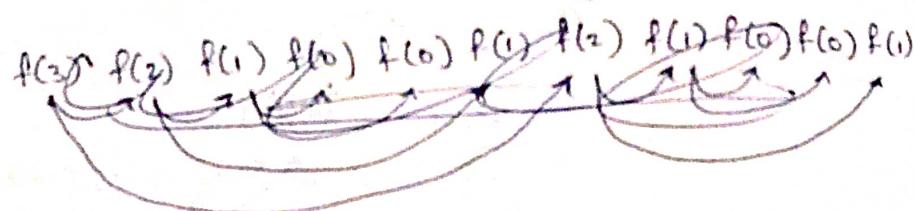
```
{
  if(x>0)
    {
      f(x-1);
      printf("%d", x);
    }
}
```

i) Find o/p of $f(3)$.



$$\therefore \underline{\text{O/P: } 1 \ 2 \ 1 \ 3 \ 1 \ 2 \ 1}$$

Seq of function calls



$$f(3) \quad f(2) \quad f(1) \quad f(0) \quad f(0) \quad f(1) \quad f(2) \quad f(1) \quad f(0) \quad f(0) \quad f(1) \\ \cdot \quad \cdot$$

for $n=3$, no. of function calls = 15

ii) Find no. of function calls for $f(n)$?

$$2^{n+1} - 1$$

iii) Depth of function calls for $f(n)$.

$$n+1$$

iv) Find time complexity.

time complexity is bounded by no. of function calls

$$\text{i.e., } O(2^n)$$

v) Find space complexity.

We need to find maximum no. of activation records stored in the stack.

∴ when $f(0)$ is executing,

Stack may contain $n+1$ activation records

∴ $O(n)$ is space complexity

(ii) write and solve recursive relation for no of function calls for $f(n)$.

Sol:

let $T(n)$ be no of function calls in $f(n)$.

$$T(n) = 1 + T(n-1) + T(n-1) \quad \text{if } n > 0$$

and $T(n) = 1 \quad \text{if } n \leq 0$

$$\Rightarrow T(n) = \begin{cases} 1 + 2T(n-1) & , \text{ if } n > 0 \\ 1 & , \text{ if } n \leq 0 \end{cases}$$

$$T(n) = 1 + 2T(n-1) \xrightarrow{\text{2nd call}}$$

$$= 1 + 2(1 + 2T(n-2))$$

$$= 1 + 2 + 2^2 T(n-2) \xrightarrow{\text{3rd call}}$$

$$= 1 + 2 + 2^2 (1 + 2T(n-3))$$

$$= 1 + 2 + 2^2 + 2^3 T(n-3) \xrightarrow{\text{4th call}}$$

$$\vdots$$
$$= 1 + 2 + 2^2 + \dots + 2^{k-2} + 2^{k-1} T(n-(k-1)) \xrightarrow{\text{kth call}}$$

$$= 2^{k-1} - 1 + 2^{k-1} T(n-(k-1))$$

Assuming k^{th} call is base call

$$\Rightarrow n - (k-1) = 0$$

$$\Rightarrow n - k + 1 = 0$$

$$\Rightarrow k = n + 1$$

$$\Rightarrow T(n) = 2^{n+1-1} - 1 + 2^{n+1-1} T(0)$$

$$= 2^n + 2^n - 1 = 2^{n+1} - 1$$

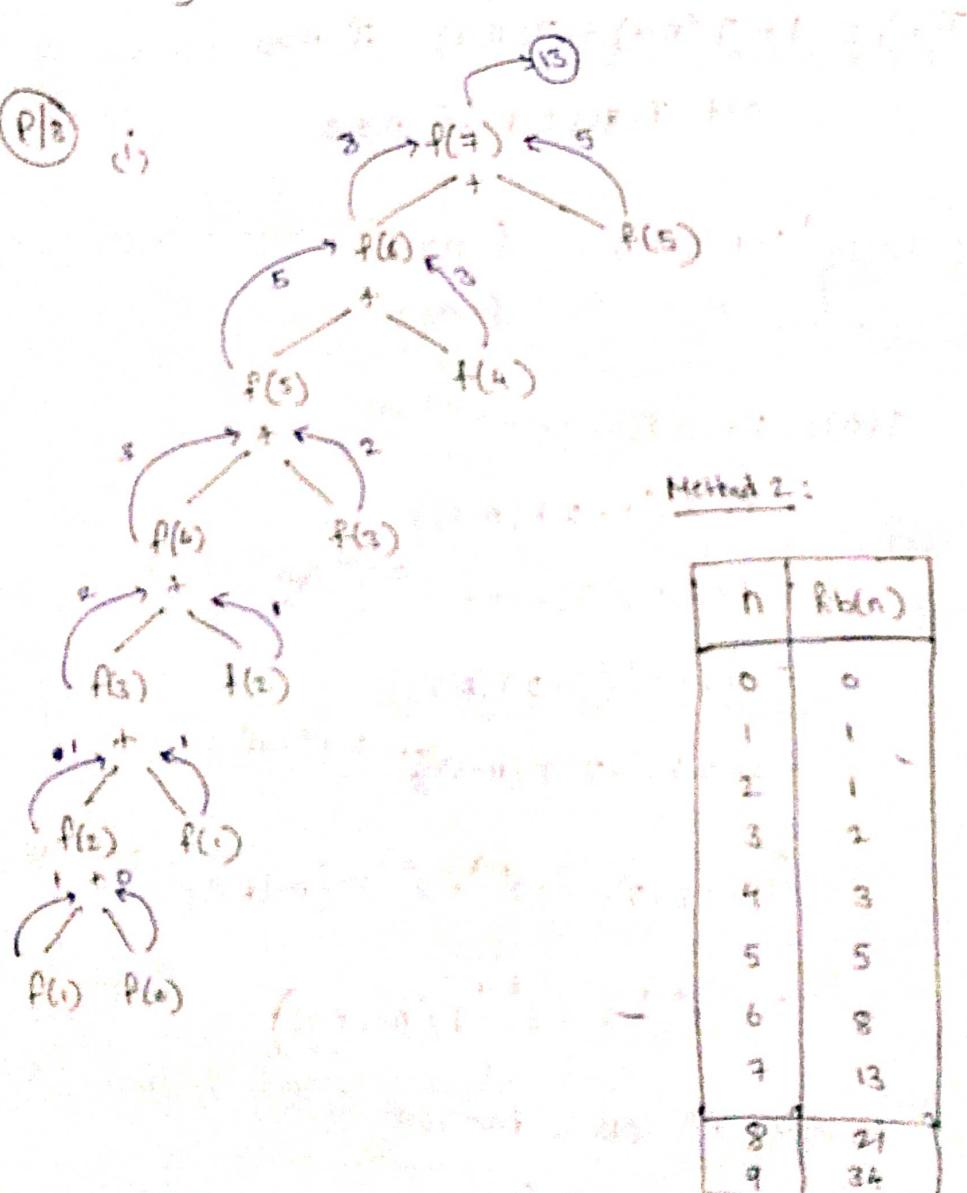
Excessive Recursion

```

int f(n)
{
    if (n==0 || n==1)
        return 0/n;
    else
        return f(n-1) + f(n-2);
}

```

(P/2)



Method 2:

n	Rib(n)
0	0
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34

If $T(n)$ is no. of function calls in $Rib(n)$ then

$$T(n) = \cancel{2}f(n+2) - 2Rib(n+1) - 1$$

$$(i) \quad \therefore T(8) = 2Rib(9) - 1$$

$$= 2(34) - 1$$

$$= 67$$

Total no. of 'i' operation for fib(n) is fib(n+1)

$$\text{(iii) no. of 'i' operations in fib(9) = fib(10) - 1}$$

$$= 55 - 1 = 54$$

Ackermann's number:

Ackermann's number $A(x,y)$ is defined as

$$A(x,y) = \begin{cases} y+1 & \text{if } x=0 \\ A(x-1,1) & \text{if } y=0 \\ A(x-1, A(x,y-1)) & \text{otherwise} \end{cases}$$

\hookrightarrow non-primitive recursive function

\hookrightarrow nested recursion

Note:

$$\rightarrow A(0,y) = y+1$$

$$\rightarrow A(1,y) = y+2$$

$$\rightarrow A(2,y) = 2y+3$$

$$\rightarrow A(3,y) = 2^{y+3} - 3$$

Eg:

$$A(2,3) = ?$$

~~$$A(2,3) = 2^{3+3}$$~~

$$\Rightarrow A(2,3) = A(2-1, A(2,2))$$

$$= A(1, A(2,2))$$

$$= A(1, 7)$$

$$= 7+2$$

$$= 9$$

$$\therefore A(2,3) = 9$$

$$A(2,2) = 2^2 + 3 = 7$$

$$\Rightarrow A(1,7) = 7+2 = 9$$

$$A(1,y) = y+2$$

$$\text{Eq: } A(2,5) = ?$$

Sol:

$$\text{WKT } A(2,y) = 2y + 3$$

$$\begin{aligned} A(2,5) &= 2(5) + 3 \\ &= 13 \end{aligned}$$

$$\text{Eq: } A(0,3) = ?$$

Sol:

$$A(0,y) = y + 1$$

$$\Rightarrow A(0,3) = 3 + 1 = 4$$

$$\text{Eq: } A(3,0) = ?$$

Sol:

$$A(3,y) = 2^{y+3} - 3$$

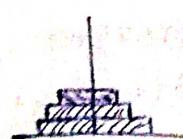
$$A(3,0) = 2^3 - 3$$

$$= 2^3 - 3$$

$$= 5$$

Towers of Hanoi

- Only one disk could be moved at a time.
- A larger disk must never be stacked above smaller one.
- One and only one auxiliary needle could be used for the intermediate storage of disk.



Source (L)



Auxiliary (M)



Destination (R)

Tower (L)

Auxiliary (M)

Destination (R)

Assume that source has $n-1$ disks.

Step 1: Move $(n-1)$ disks from source to source to auxiliary.

Step 2: Move one disk from source to destination.

Step 3: Move $(n-1)$ disks from auxiliary to destination

source auxiliary destination
↑ ↑ ↑
void TOH (n, L, M, R)

{

if ($n == 1$)

{
 printf("Move disk from ");

 printf(LR);

}

else

{

 TOH ($n-1$, L, R, M);

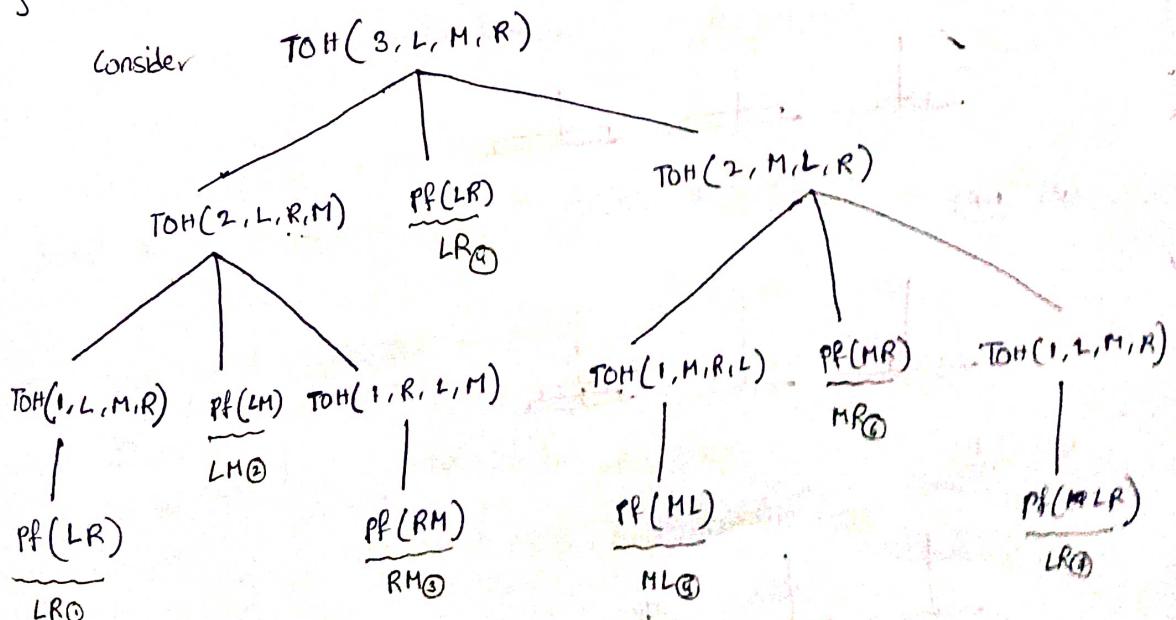
 printf("Move disk from ");

 TOH ($n-1$, M, L, R);

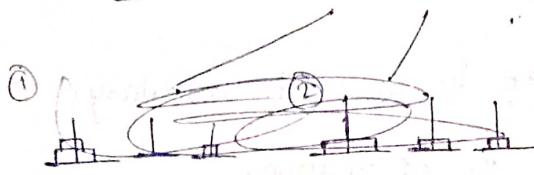
}

}

Consider



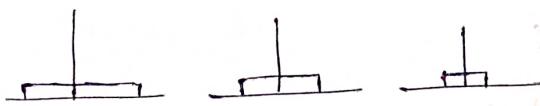
OIP: ~~LR~~, LR, LM, RM, LR, ML, MR, LR



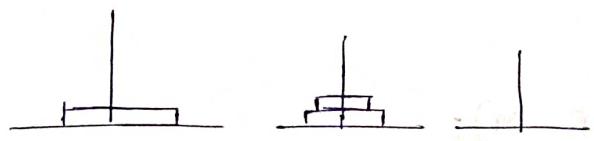
(i) LR:



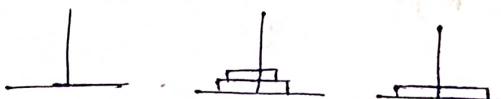
(ii) LM:



(iii) RM:



(iv) LR:



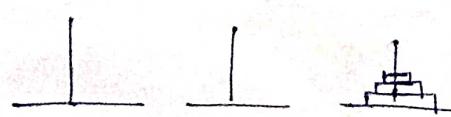
(v) ML:



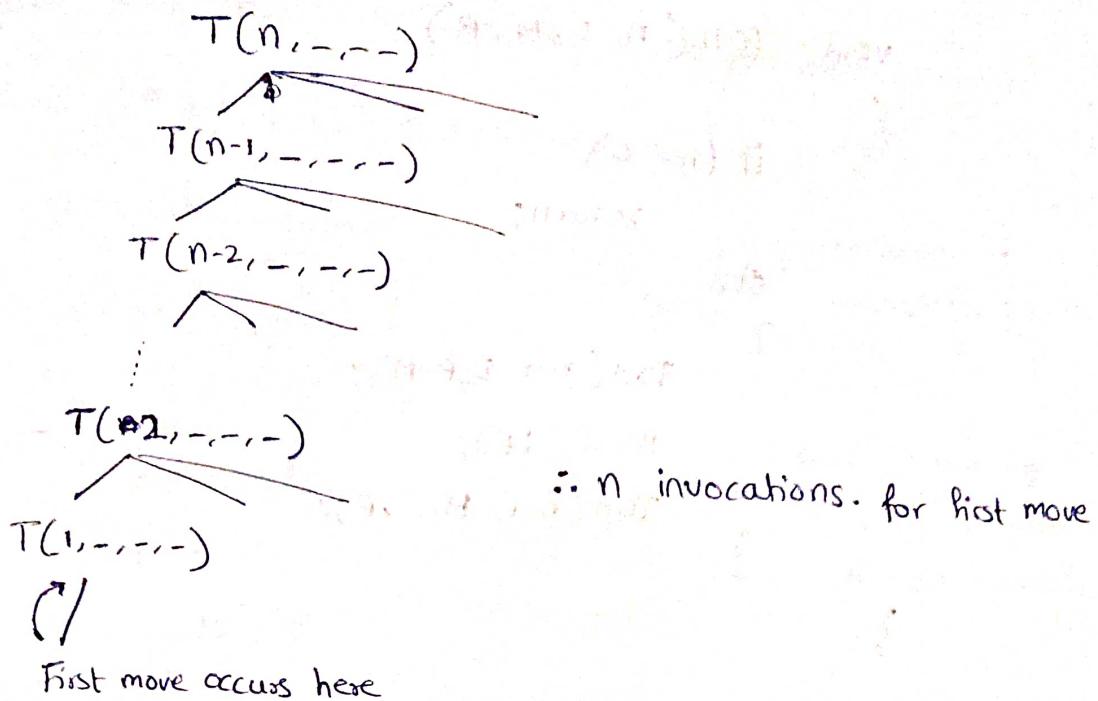
(vi) MR:



(vii) LR:



Eg: find no of invocations required for towers of hanoi to make first move. ~~first~~



Eg: Find no of invocations required for last move.

Sol: we obtain a binary tree of height $n-1$, for n disks.

last move is done at last invocations ~~of~~

\Rightarrow we need to find no of invocations.

= no of nodes in FBT of height $n-1$

$$= 2^{(n-1)} - 1 = 2^n - 1$$

Eg: Total no of invocations for ' n ' disks $= 2^n - 1$

Eg: Find total no of moves for ' n ' disks.

Sol:

each function invocation has one move

\Rightarrow no of moves = no of function calls

$$= 2^n - 1$$

→ The below program is also possible for towers of hanoi.

```
void TOH(n, L, M, R){  
    if (n==0)  
        return;  
    else  
    {  
        TOH(n-1, L, R, M);  
        printf(LR);  
        TOH(n-1, M, L, R);  
    }  
}
```

However, this has more function invocations

$$\text{Here total move} = 2^n - 1 \quad (\because \text{moves must remain same})$$

- no of invocations of last move = $2^{n+1} - 1 - 1$
 $= 2^{n+1} - 2 \quad (\because \text{last move occurs in last invocation, so we subtract one but one invocation})$

$$\text{no of invocations for first move} = n+1$$

- (Q45) which of the following is recurrence eqn of tower of hanoi for computing its optimal execution time.

a) $T(n) = T(n-2) + 2$

b) $T(n) = 2T(n-1) + n$

c) $T(n) = 2T(n/2) + 1$

d) $T(n) = 2T(n-1) + 1$

Sol:

In func $\text{TOH}(n, L, M, R)$ we call ~~2 times~~ $\text{TOH}()$ twice

$$T(n) = 2T(n-1) + 1 \quad \text{calling called func} \rightarrow \text{calling func} \quad \therefore \text{opt(d)}$$

Converting Infix to Postfix

Consider below infix notation & convert it into postfix

$$2 * 3 / (2 - 1) + 5 * (4 - 1)$$

write the given infix expression using paralysis using precedence & associativity

$$\left[\left(\left(2^3 \right) / (2 - 1) \right) + \left(5 * (4 - 1) \right) \right]$$

↓ ↓ ↓ ↓ ↓
 2 3 * 2 1 - + 5 4 1 - *
 ↓ ↓ ↓ ↓ ↓
 2 3 * 2 1 - / 5 4 1 - *

Here we put \oplus operator in place of right parenthesis

$$\text{Postfix: } 2 3 * 2 1 - / 5 4 1 - * +$$

(Q4b) The postfix expression for the infix expression

$$A + B * (C + D) / F + D * E$$

a) $A B + C D + * F / D + E ^ *$

b) $A B C D + * F / + D E ^ * +$

c) $A ^ * B + C D / F ^ * D E + +$

d) $A + ^ * B C D / F ^ * D E + +$

Sol:

writing given expression using paralysis

$$\left(A + \left(\left(B * (C + D) \right) / F \right) \right) + (D * E)$$

↓ ↓ ↓ ↓ ↓
 A + B C D + * (C + D) / F + D * E
 ↓ ↓ ↓ ↓ ↓
 A B C D + * F / + D E *

$$A B C D + * F / +$$

$$\therefore A B C D + * F / + D E ^ * +$$

opt(b)

Converting infix to Prefix

consider

$$2 * 3 / (2-1) + 5 * (4-1)$$

write in the exp using parenthesis

$$((2 * 3) / (2-1)) + (5 * (4-1))$$

replace left parenthesis with the operator

$$+/*23-21*5-41$$

$$\text{Eq: } A + B * (C+D)/F + D * E$$

$$+A/*B+CDF$$

∴ final prefix expression is

$$++A/*B+CDF*DE$$

Pl/12

a = b + c * d / e + f + g + h - i * j

↑ has highest precedence and it is right associative

+ f= b/*cde ↑f↑gh

$$= a - \cancel{t+b} * cde \uparrow f \uparrow gh * ij$$

∴ opt @

Infix to postfix using stack:

Consider

$$2 * \underline{3} | (2-1) + 5 * (4-1)$$

i) Examine the input sequence. (one character)

- (i) Examine the input sequence. (One character)
If it is operand then print it and goto step (ii)
- (ii) If it is operator then
a) If it is + then add the value of the stack to the result
b) If it is - then subtract the value of the stack from the result
c) If it is * then multiply the value of the stack with the result
d) If it is / then divide the value of the stack with the result
e) If it is % then find the remainder of the stack with the result
f) If it is ^ then find the power of the stack with the result
g) If it is < then compare the value of the stack with the result
h) If it is > then compare the value of the stack with the result
i) If it is = then compare the value of the stack with the result
j) If it is <= then compare the value of the stack with the result
k) If it is >= then compare the value of the stack with the result
l) If it is != then compare the value of the stack with the result
m) If it is == then compare the value of the stack with the result

- (ii) If it is operand then ignore it.
- (iii) If it is opening parenthesis then push it into stack & goto step (i).

(iv) If it is an operator

- a) If stack is empty then push operator into stack and

b) If top of the stack is opening parenthesis then push

operator into the stack and goto step(i)

c) If ~~top of the stack~~ it has higher priority than the operator on top of the stack, then push operator on to the stack.

stack, otherwise pop off operator from the stack

and goto step(iv) - (If the operator is right associative
then and has equal precedence to the operator on top of stack then push
onto stack (goto step(i))

(v) If it is closed parenthesis then pop off operators from stack until we encounter opening parenthesis and pop the opening parenthesis and discard it.

(vi) After scanning entire string pop out remaining contents of stack

Consider below infix expression

$$\rightarrow 2 * 3 / (2-1) + 5 * (4-1)$$

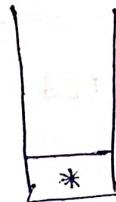
↑
O/P: 2

$$\rightarrow 2 * 3 / (2-1) + 5 * (4-1)$$

↑

Stack is empty.

so push '*'



$$\rightarrow 2 * 3 / (2-1) + 5 * (4-1)$$

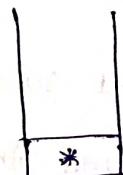
↑

O/P: 2 3

$$\rightarrow 2 * 3 / (2-1) + 5 * (4-1)$$

↑

/ doesn't have high priority



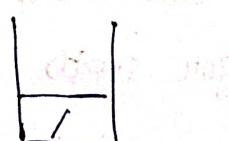
than *

so pop *



now stack is empty.

so push /



$$\rightarrow 2 * 3 / (2-1) + 5 * (4-1)$$

↑

push '(' on to stack



$$\rightarrow 2 * 3 / (2-1) + 5 * (4-1)$$

O/P: 2 3 * 2 /

$$\rightarrow 2 * 3 / (2-1) + 5 * (4-1)$$

push '-'



$$\rightarrow 2 * 3 / (2-1) + 5 * (4-1)$$

O/P: 2 3 * 2 1

$$\rightarrow 2 * 3 / (2-1) + 5 * (4-1)$$

Now pop the stack until
we encounter 'C'



O/P: 2 3 * 2 1 -

$$\rightarrow 2 * 3 / (2-1) + 5 * (4-1)$$

↑

+ too doesn't have higher
precedence than '*'
∴ pop '*'



O/P: 2 3 * 2 1 - /

now stack is empty.



so push '+'

→ O/P: 2 3 * 2 1 - / 5

$$\rightarrow 2 * 3 / (2-1) + 5 * (4-1)$$

* has higher precedence
than +



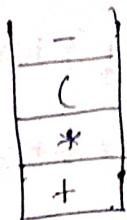
$$\rightarrow 2 * 3 / (2 - 1) + 5 * (4 - 1)$$

$$\text{O/P: } 23 * 21 - / 541$$



$$\rightarrow 2 * 3 / (2 - 1) + 5 * (4 - 1)$$

$$\text{O/P: } 23 * 21 - / 541$$



$$\rightarrow 2 * 3 / (2 - 1) + 5 * (4 - 1)$$

Pop until C is encountered

$$\Rightarrow \text{O/P: } 23 * 21 - / 541 -$$



→ Now entire string is scanned.

Now pop out the remaining stack

$$\therefore \text{O/P: } 23 * 21 - / 541 - * +$$



Q: Convert below infix expression into postfix

$$A^B * C / (D * E - F)$$

Symbol

A

\wedge

B

*

C

/

Stack

\wedge

\wedge

*

*

/

Postfix

A

A

A B

A B \wedge

A B \wedge C

A B \wedge C *

<u>Symbol</u>	<u>Stack</u>	<u>Postfix</u>
C	/C	AB^C*
D	/C	AB^C*D
*	/C*	AB^C*D
E	/C*E	AB^C*D E
-	/C*-	AB^C*D E
F	/C-	AB^C*D E F
)	/	AB^C*D E F -

Pop remaining symbols AB^C*D E F - /

Ex: Convert below infix expression into postfix

$$8 * (5^4 + 2) - 6^2 / 9 * 3$$

<u>symbol</u>	<u>stack</u>	<u>postfix</u>
8		8
*	8*	8*
(8*	8*
5	8*	85
^	85	85
4	854	854
+	854^	854^
2	854^2	854^2
)	854^2	854^2
-	854^2*	854^2*
6	854^2*	854^2*
^	854^2*	854^2*
2	854^2*	854^2*
/	854^2*	854^2*
9	854^2*	854^2*
*	854^2*	854^2*
3	854^2*	854^2*

Pop remaining symbols

$$854^2 + * 6^2 9 / 3 *$$

∴ postfix: $854^2 + * 6^2 9 / 3 *$

Infix to prefix using stack:

Eg: Convert

- (i) Reverse the input string.
- (ii) Examine the input sequence (one character)
- (iii) If it is an operand then pop it and goto step (ii).
- (iv) If it is opening para closing parenthesis then push it onto stack and goto step (ii)
- (v) If it is an operator
 - a) If stack is empty push operator into stack & goto step (ii)
 - b) If top of stack is having closing parenthesis then push operator into stack
 - c) If the operator has higher precedence than the operator on top of the stack, then push it.
If it has lower precedence then pop the operator and goto step (v)
If the precedence is same and operator is right associative then pop the operator & goto step (v). If it is left associative then push it onto stack and goto step (ii)
- (vi) If it is opening parenthesis then pop off operator until we encounter a closing parenthesis and pop it & discard it.
- (vii) If there is no more i/p then pop out the remaining symbols from stack.
- (viii) Reverse the o/p

sq:
Revers

symbol

)

1

-

4

(

*

5

+

)

1

-

2

(

1

3

2

pop remain

Reverse the

o/p

Eg: Convert below infix into ~~postfix~~ prefix

$$2 * 3 / (2 - 1) + 5 * (4 - 1)$$

Sq: 03 27 35 41

Reverse the i/p string

$$) 1 - 4 (* 5 +) 1 - 2 (/ 3 * 2$$

stack

rep (ii)

push

operator

operator and

+ associativity

it is

no step (ii)

we

it.

mbols

<u>Symbol</u>	<u>stack</u>	<u>Prefix</u>
))	
1)	1
-) -	1 -
4) -	1 - 4
() -	1 - 4 -
*) - *	1 - 4 - 5
5) - *	1 - 4 - 5 *
+) - *	1 - 4 - 5 *
)) - *	1 - 4 - 5 *
1) - *	1 - 4 - 5 *
-) - *	1 - 4 - 5 *
2) - *	1 - 4 - 5 * 12
() - *	1 - 4 - 5 * 12 -
/) - *	1 - 4 - 5 * 12 - 3
3) - *	1 - 4 - 5 * 12 - 3 /

~~pop remaining elements~~

~~Reverse the o/p~~

$$\cancel{1 * 2 / 3 - 1 * 5 + 1}$$

<u>Symbol</u>	<u>stack</u>	<u>Prefix</u>
*	+/*	14-5*12-3
2	+/*	14-5*12-32
pop out remaining elements		14-5*12-32*/+

Reverse the IOP

prefix: +/*23-21*5-41

Postfix evaluation using stack

- Scan the IOP sequence. (one character)
- If it is an operand then push it.
- If it is an operator then apply two pop operation, where the first pop operation element is assigned to the operator as a right child and the second pop operand is assigned to operator as a left child
Never push the result onto stack.
- Without creating tree push the result from step(iii) into stack.
- goto step(i)

Eg: Evaluate below postfix expression using stack

823^/2^3*+51*-

<u>Symbol</u>	<u>stack</u>
8	8
2	82
3	823
^	88
/	1
2	12

$$\begin{array}{c} 1 \\ / \quad \backslash \\ 2 \quad 3 \end{array} = 2^3 = 8$$

$$\begin{array}{c} 1 \\ / \quad \backslash \\ 3 \quad 8 \end{array} = 8/8 = 1$$

<u>Symbol</u>	<u>Stack</u>
3	123
*	16
+	7
5	75
1	75
*	75
-	2



∴ Final result is obtained in the stack.

i.e., 2

Prefix evaluation using stack

- i) Reverse the prefix expression
 - ii) Scan the expression (one character)
 - iii) If it is an operand push it into stack
 - iv) If it is an operator then perform two pop operation
 - v) pop operand from stack and assign it as left child of the operator
 - vi) pop operand from stack and assign it as right child of the operator
 - vii) Evaluate the result & push it onto stack
 - viii) goto step iv

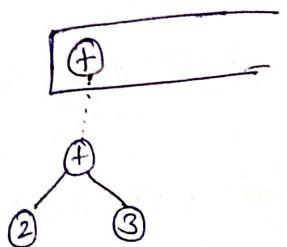
Expression tree from post fix

Eg: $23 + 6 * 4 *$

23

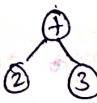
- 5: pop(3) make it left child of (1)
 pop(2) make it right child of (1)

Now push the address of node (1) into the stack



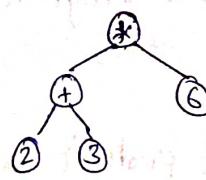
- 6: push onto stack

(+, 6)

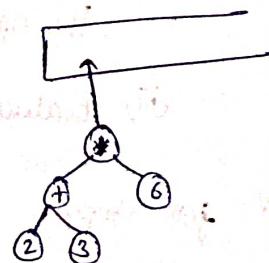


- 7: pop(6) ... make it right child

pop(+) -- make it left child

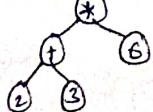


now push address of (*) onto stack



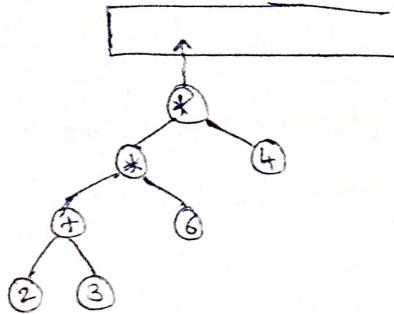
8:

↑ (4)



1st pop
operand
left chi

Ex *:



~~Infix~~

- Note:
- Inorder traversal on expression tree gives infix expression.
 - Preorder traversal on expression tree gives prefix expression.
 - Postorder traversal on expression tree gives postfix expression.

Perform preorder traversal on above expression tree.

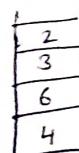
∴ Prefix expression: * * + 2 3 6 4

Eg: Evaluate below prefix expression using stack.

* * + 2 3 6 4

Reverse the expression
4 6 3 2 + * *

After reading 4 6 3 2 :

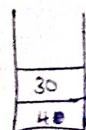
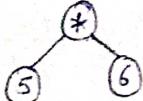


+=

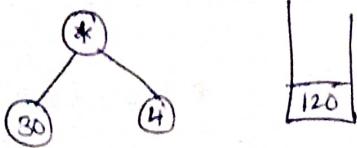


1st popped operand is left child

*



*:



Ans: 120

Expression tree using prefix notation:

Eg: Build expression tree for below prefix expression

~~4 6 3 2 + * *~~ * * + 2 3 6 4

Ans: After reading each character in forward direction

Now Reverse, the expression

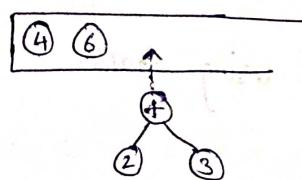
4 6 3 2 + * *

After reading 4, 6, 3, 2:

stack

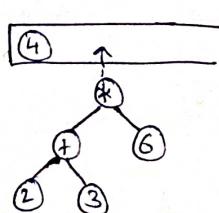
④ ⑥ ③ ②

*:



1st popped node is added as left child.

*



2nd popped node is added as right child.

*

