Sub :

Day

Time :

Date : 13/8/21

Tree

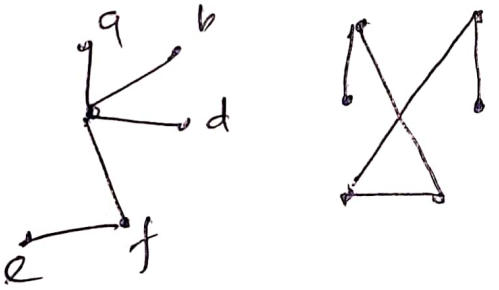
A tree is a connected undirected graph of with no simple circuits

graph will a be a tree - if

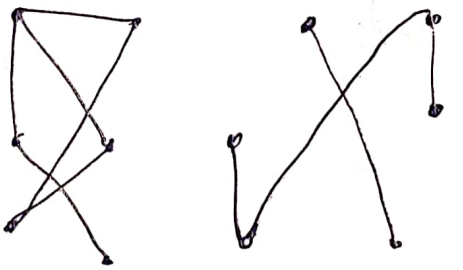
- No circuits
- connected
- consits single vertex

→ graph will be a forest iff and, not ckt, not connect

Tree



Not tree



Sub :

Day

Time :

Date : / /

forest Theorem

→ An undirected graph is a tree if and only if there is a unique path between any two of its vertices

→ For any positive integer n, if G is a connected graph with n vertices and $n-1$ edges, then G is a tree.

Sub :

Day

Time : Date : / /

Rooted tree

A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from root.

→ A rooted tree is a tree with a distaingused vertex called the root.

→ The level of a vertex in a rooted tree is the number of edges from the vertex to the root.

→ The height of a rooted tree is the maximum level of any vertex in the tree.

→ Given an internal vertex in a rooted tree is its children are those vertices adjacent to it and one level higher.

Sub :

Day

Time : Date : / /

→ If u and v are vertices in a rooted tree, with u a child of v, then v is called the parent of u.

→ If u and v are vertices in a a rooted tree, with u

→ Two vertices which are children of the same vertex are called siblings.

→ Given two vertices u and v in a rooted tree, if u lies on the path from v to the root, then u is called an ancestor of v, and v is called descendent of u,

Sub :

Day

Time : Date : / /

Definitions:

Parent: Suppose that T is a rooted tree. If v is a vertex in T other than the root, the parent of v is the unique vertex u such that there is a directed edge from u to v.

child: If u is the parent of v, then v is called a child of u.

Siblings: Vertices with the same parent are called siblings.

Ancestors: The ancestors of vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root.

Sub :

Day

Time : Date : / /

Descendants :

The descendants of a vertex v are those vertices that have v as an ancestor.

leaf : vertex with no children.

subtree : if a is a vertex in a tree, the subtree with a as its root is the subgraph of the tree consisting of a and its descendants and all edges incident to these descendants.

Internal vertices :

vertices that have children.

m-ary tree : A rooted tree is called an m-ary tree if every internal vertex has no more than m children, & if exactly m children. → Full m-ary.

Sub :

Day

Time : Date : / /

Theorem:

→ A full m-ary tree with i internal vertices contains $n = mi + 1$ vertices.

→ n vertices has $i = \frac{n-1}{m}$ internal vertices and $l = \frac{(m-1)n+1}{m}$ leaves.

→ i internal vertices has $n = mi + 1$ vertices an $l = (m-1)i + 1$ leaves.

→ & l leaves has $n = \frac{ml-1}{m-1}$ vertices and $i = \frac{l-1}{m-1}$ int. vertices

Level → Rank of nodes

Height → Number of edges (highest leaf to root)

Balanced tree: rooted m-ary tree of height h is balanced if all leaves are at level h or h-1

Sub :

Day

Time : Date : / /

→ There are at most m^H leaves in m-ary tree of height H.

→ There are at most 2^H leaves in a binary tree.

→ If an m-ary tree of height h has l leaves, then $h \geq \lceil \log_m l \rceil$. If the m-ary tree is full and balanced, then $h = \log_m l$.

→ If a Binary tree of height h has l leaves, then $h \geq \lceil \log_2 l \rceil$, If the binary tree is full and balanced, then $h = \lceil \log_2 l \rceil$.

Sub :

Day

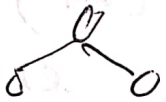
Time : Date : / /

Binary Tree

A binary tree is a special tree data structure in which each node can have at most 2 children. Thus, in a binary tree, each nodes has either 0, 1, child.

Unlabeled Binary Tree:

A Binary tree is unlabeled if its nodes are not assigned any label.



→ Number of different Binary tree possible with 'n' unlabeled nodes $= \frac{2nCn}{n+1}$

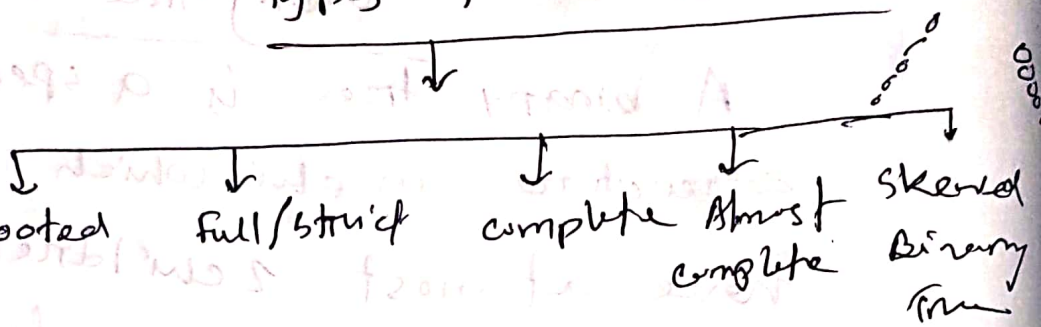
→ for labeled $= \frac{2nCn}{n+1} \times n!$

Sub :

Day

Time : Date : / /

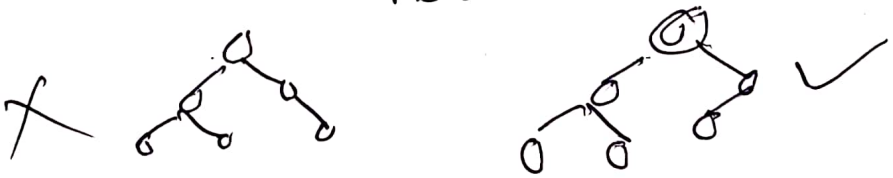
Types of Binary Tree



full → ~~root~~ node has 0 or 2 children

complete → All the leaf node has same level.
→ Every external node has exactly 2 children

Almost complete: → Every level is complete except last level.
→ the last level must be strictly filled from left to right.



Sub :

Day

Time : Date : / /

Binary Tree Properties

(I) Minimum Number of nodes in a binary tree of height $H = H+1$.

↳ To construct a binary tree of height = 4, we need ~~to~~ at least $4+1 = 5$ nodes

(II) Maximum number of nodes in a binary tree of height $H = 2^{h+1} - 1$

(III) total Number of leaf nodes in a Binary Tree. = Total number of nodes with 2 children + 1

(IV) Maximum number of nodes of any level 'L' in a binary tree $= 2^L$.

Sub :

Day

Time : Date : / /

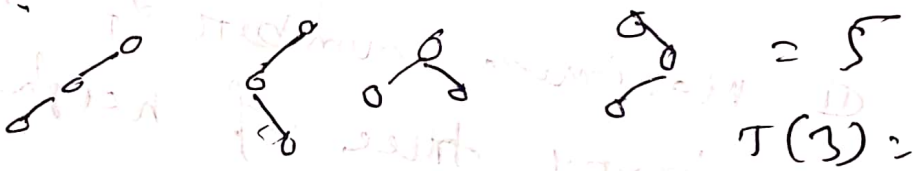
Number of Binary Tree

① Unlevelled Nodes

② Levelled Nodes

How many binary trees can be drawn ?

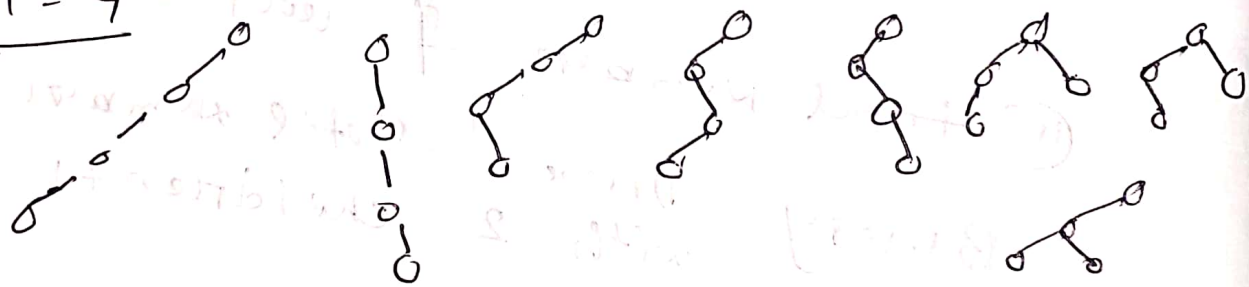
$n = 3$?



$= 5$

$T(3) = 5$

$n = 4$



$T(4) = 14$

$$T_n = \frac{2n C_n}{n+1}$$

Sub :

Day

Time : Date : / /

height vs Nodes

height

$h = 1$

Min

$n = 2$

Max

$n = 3$

$h = 2$

min

$n = 3$

max

$n = 7$.

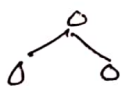
if height is given,

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

Nodes vs hight

$n = 7$

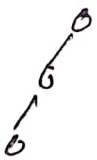
Min



$h = 1$

$n = 3$

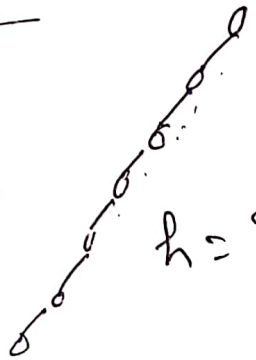
Max



$h = 2$

min

$h = 2$



$h = 6$.

if node is given.

Min hight	$= \log_2(n+1) - 1$
Max "	$= n - 1$

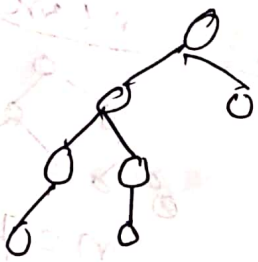
Sub :

Day

Time :

Date : / /

Relation between internal and external Nodes (leaf)

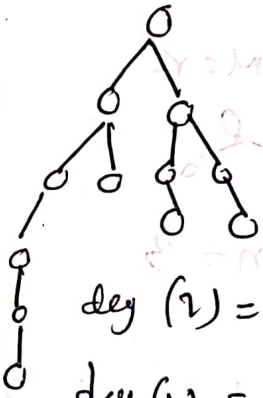


nodes with

degree (2) = 2

degree (1) = 2

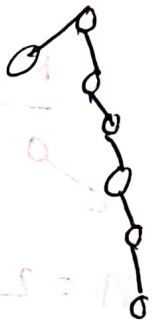
degree (0) = 3



deg (2) = 3

deg (1) = 5

deg (0) = 4



deg (2) = 1

deg (1) = 4

deg (0) = 2.

$\therefore \deg(0) = \deg(2) + 1$

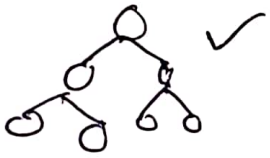
Sub :

Day

Time : Date : / /

Strict Binary Tree

→ Either 0 or 2 children



height versus nodes

Here,

Min Nodes $n = 2h+1$

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

} h is given.

Min height $h = \log(n+1) - 1$

Max height $h = \frac{n-1}{2}$

} n is given

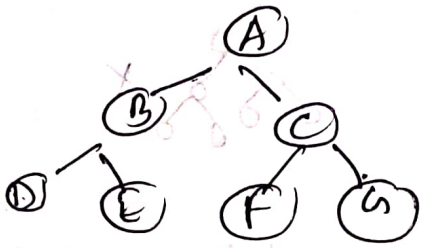
Sub :

Day

Time :

Date : / /

Array Representation of Binary Tree



A	B	C	D	E	F	G

element	index	left child	Right child
A	1	2	3
B	2	4	5
C	3	6	7

∴

i	$2*i$	$2*i+1$
element	left	Right

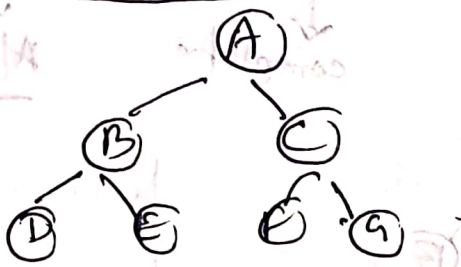
$\therefore$ parent $= \left\lfloor \frac{i}{2} \right\rfloor$

Sub :

Day

Time : Date : / /

full vs complete Binary Tree

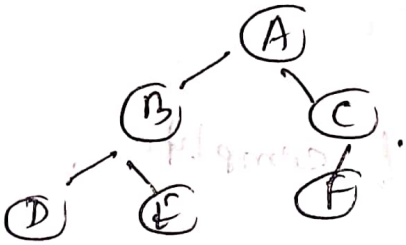


} full of nodes

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

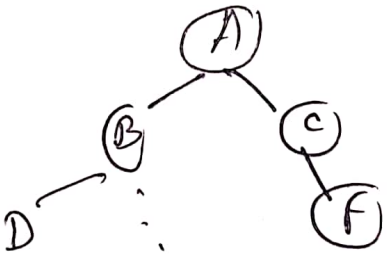
T = | A | B | C | D | E | F | G |

complete: [There will be no blank spaces in between two elements]



??

T = | A | B | C | D | E | F | ? |



not complete.

T = | A | B | C | D | – | – | F |

↑ ↑

Sub :

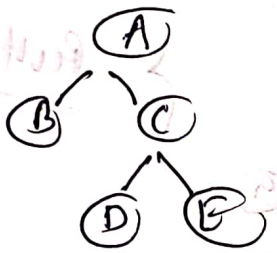
Day

Time :

Date : / /

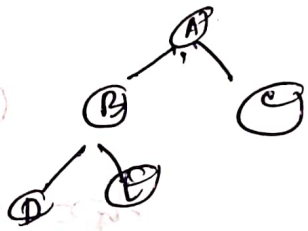
Strict vs complete

complete

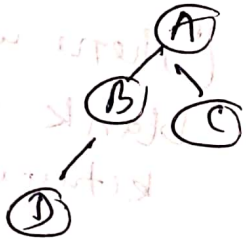


strict,
Not complete

Almost complete



strict, complete



Not strict, complete.

Sub :

Day

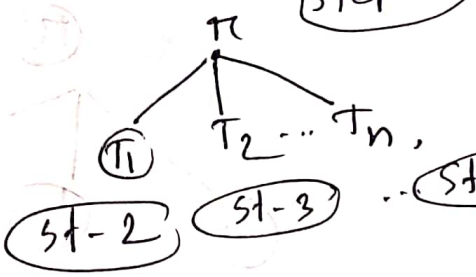
Time : Date : / /

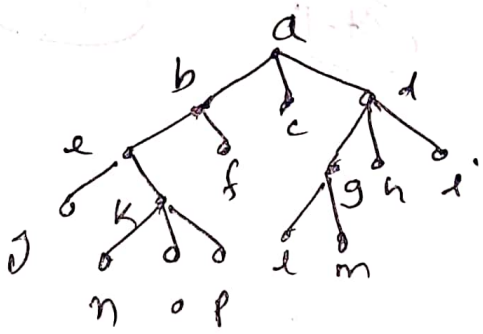
Tree Traversal (n-ary)

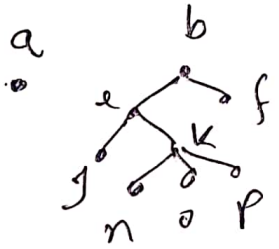
- Preorder
- Inorder
- postorder.

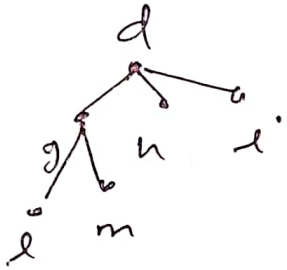
Preorder

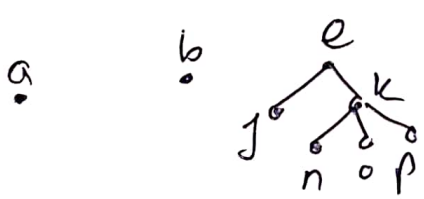
Step-1

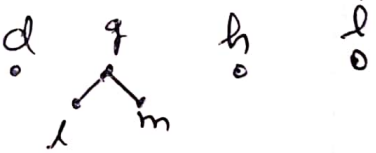












Sub :

Day

Time :

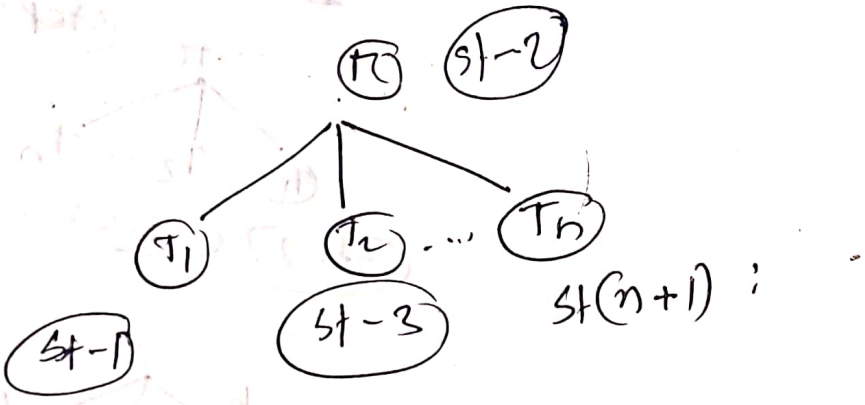
Date : / /

a b c j K f c d g l m h l

n o p

a b c j k n o p f c d g l m h e

Inorder

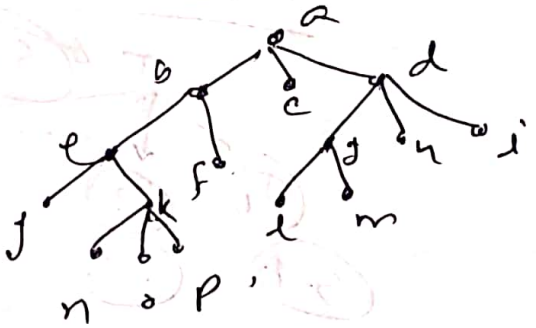


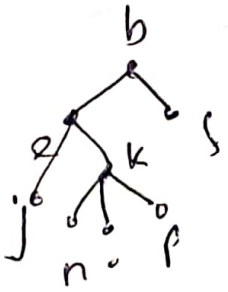
Sub :

Day

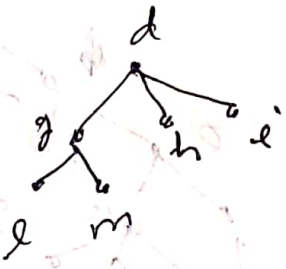
Time : Date : / /

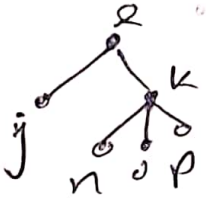
Inorder Traversal





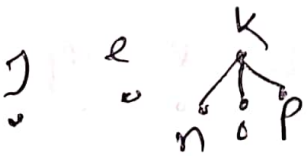
a c





b f g c

d b i



b f g c

l g m d h i

⇒ j e n k o p b f g c l g m d h i

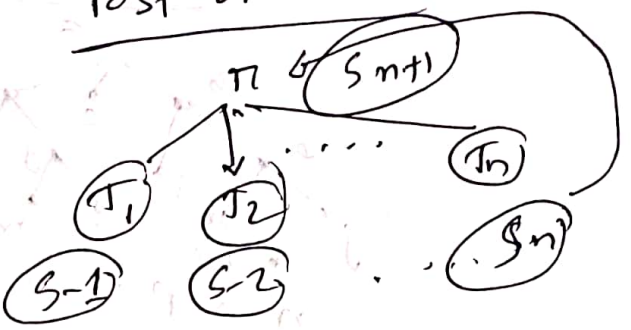
Sub :

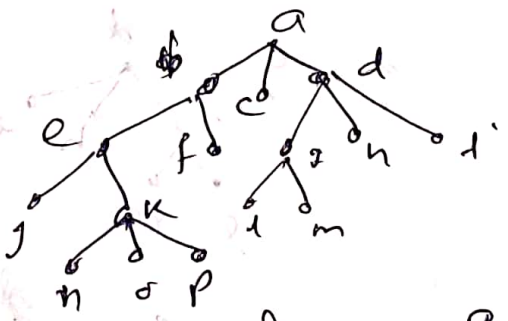
Day

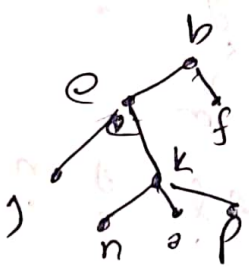
Time :

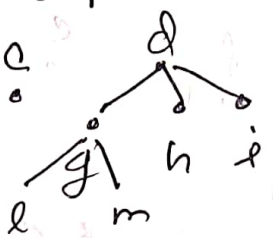
Date : / /

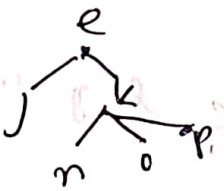
Post order

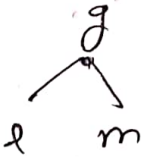












j n o p k e f b c l m g d i a

Sub :

Day

Time : Date : / /

Binary Tree Traversal

- Depth First
  - Pre
  - In
  - Post.
- Breath First

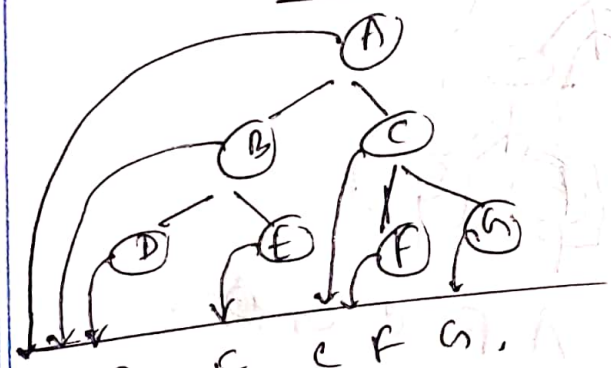
Root	Left	R

L	Root	R

L	R	Root

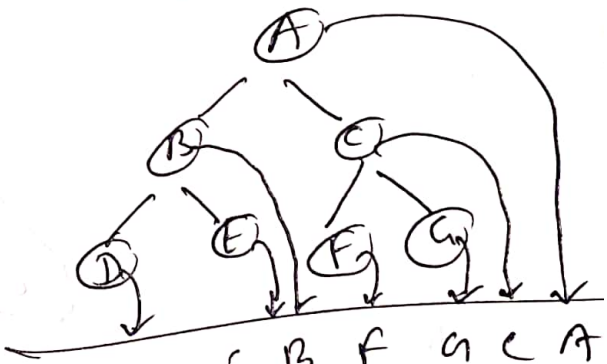
Tree Traversal (short Tech)

Preorder



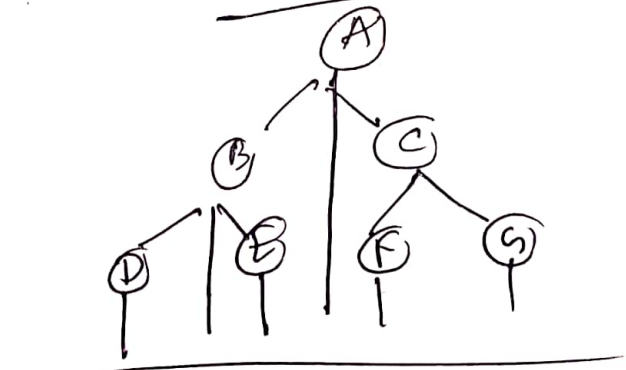
A B D E C F G.

Postorder



D E B F G C A

Inorder



D B E A F C G.

Sub :

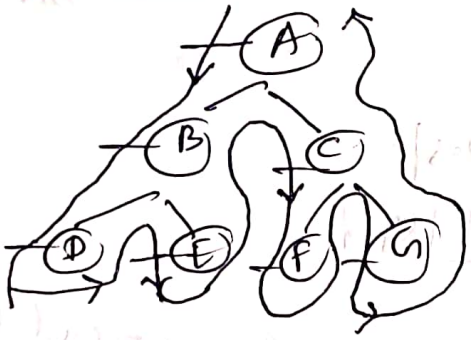
Day

Time :

Date : / /

Methode-2

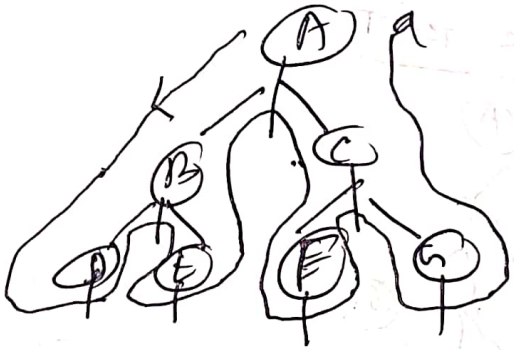
Preorder



A, B, D, E, C, F, G

Post order

D, E, B, F, G, C, A



D B E A F C G A.

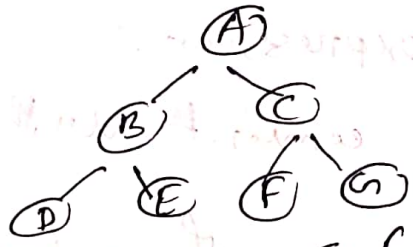
Sub :

Day

Time : Date : / /

Breadth first Traversal

→ Level By level. (level order Traversal)



A B C D, E, F, G

Sub :

Day

Time :

Date : / /

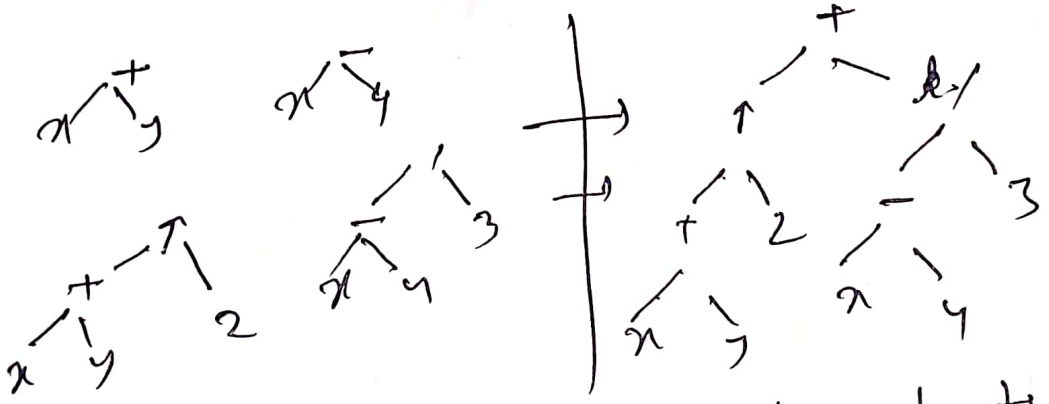
Infix, Prefix and Postfix Notation

→ construct tree from Mathematical expression.

→ convert Mathematical expression using binary tree

→ value of mathematical expression.

→ construct the tree $((x+y)*2)+((x-y)/3)$



if we want to convert the expression into prefix. we just need to go preorder.

prefix $+*+xy2/-xy3$

postfix $xy+2*xy-3/+$.

Sub :

Day

Time : Date : / /

Evaluating Expression

→ +-*235/↑234

+-*235/↑23 4

+-*235/84.

+-*2352

+-652

+12

= 3

→ 723*-4↑93/+

76-4↑93/+

14↑93/+

= 193/+

= 13+

= 4.

Sub :

Day

Time :

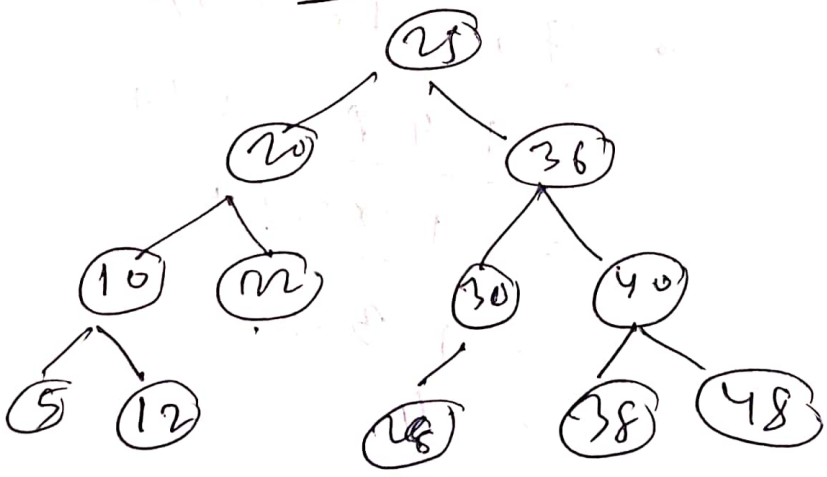
Date : / /

Binary search Tree.

Binary search Tree is a special kind of binary tree in which nodes are arranged in a specific order.

- In a binary search Tree BST, each node contains -
  - only smaller values in its left sub tree.
  - only larger values in its right sub tree.

Example



Sub :

Day

Time : Date : / /

Number of Binary Search Trees:

Number of distinct binary search tree possible with n distinct keys $= \frac{2nCn}{n+1}$

Lets construct of a binary search tree

50, 70, 60, 90, 10, 40, 100.

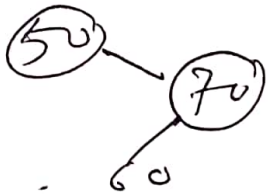
→ Always consider the first element as root node.

→ consider the given elements and insert them in BST one by one

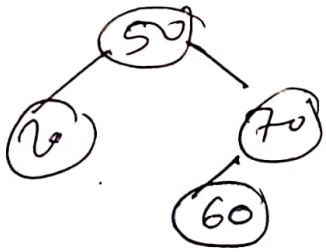
Insert - 50

insert - 70 ; 70 > 50. ...

insert 60: As 60 > 50,

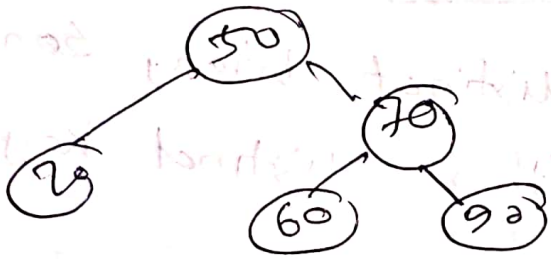


insert 20: 20 < 50,

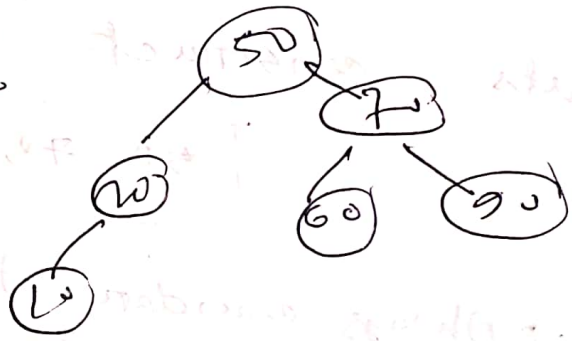


Sub : _____ Day _____ Time : _____ Date : / /

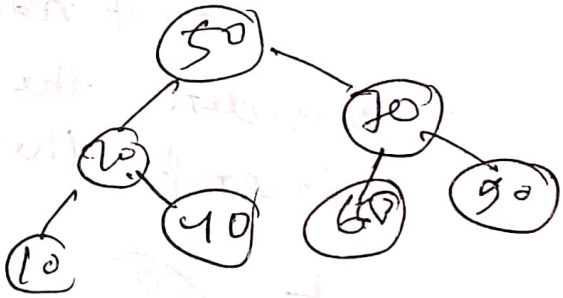
insert 90 :



insert 10



insert 40



Sub :

Day

Time : Date : / /

Binary Search Tree Operation

- Search
- Insert
- Delete.

Search Option :

→ Compare the key with the value of root node.

→ If the key is present at the root node, then return the root

→ If the key is greater than root value, then recurse for the root node's right subtree.

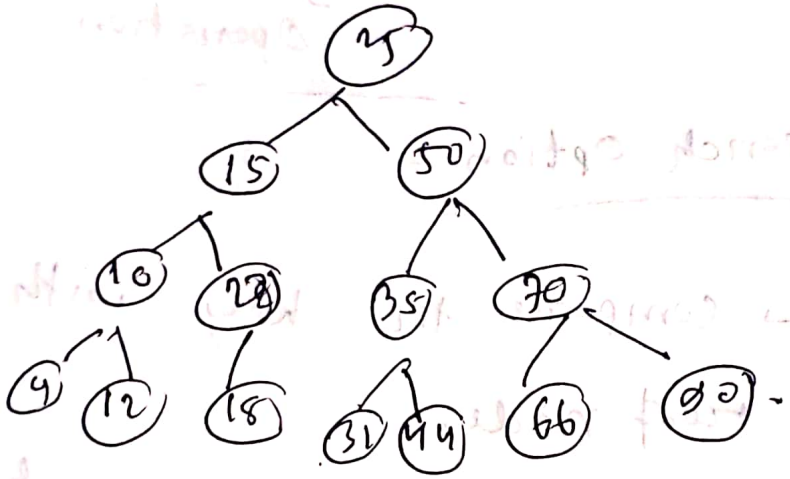
– If the key is ~~similar~~ smaller, then recurse for the left subtree.

Sub :

Day

Time :

Date : / /



if key = 45

→ Start from root node 25.

→ 45 > 25, right.

→ 45 < 50, left.

→ 45 > 35, right.

→ 45 > 44 → right, but 44 has no subtree

So, 45 is not available.

Sub :

Day

Time : Date : / /

Insert

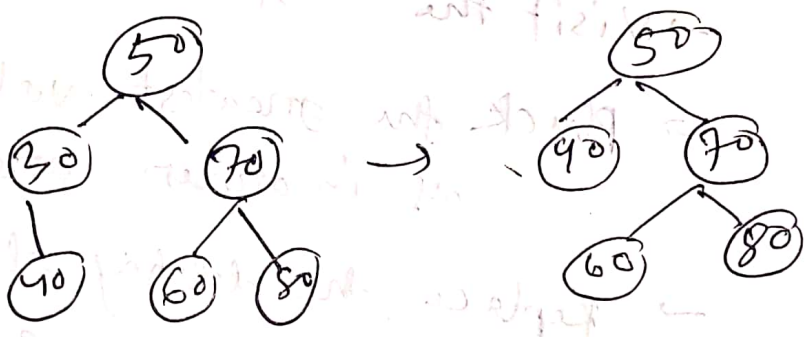
same as search

Deletion

C-1 : Deletion of Node having No child (Leaf node)

→ just remove

C-2 : Deletion of A Node having one child

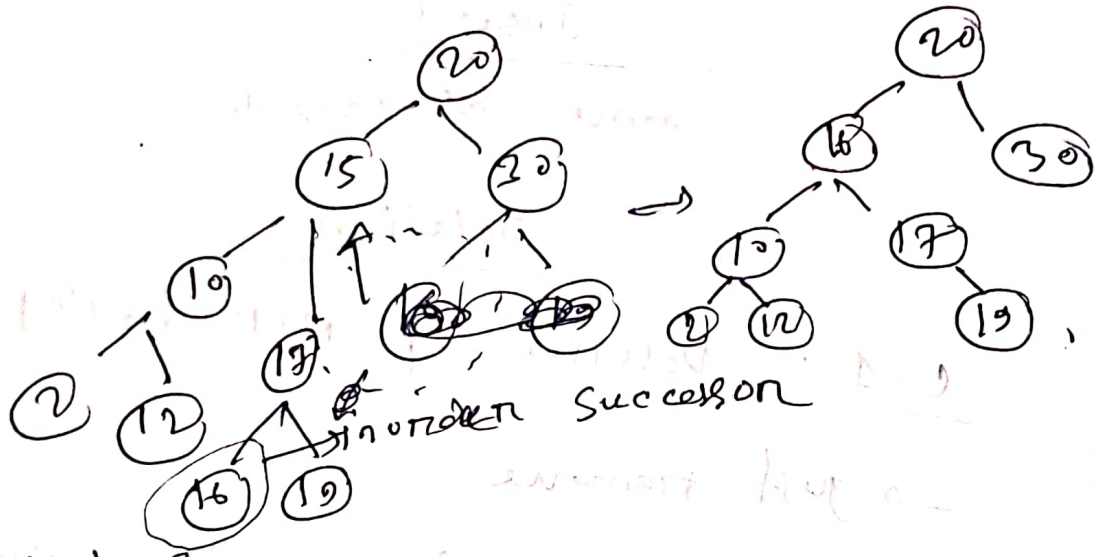


C-3 / Deletion of a node having two children

→ Visit to the right subtree of the deleting node.

→ pluck the least valued element called as inorder successor.

→ Replace the deleting element with its inorder successor.

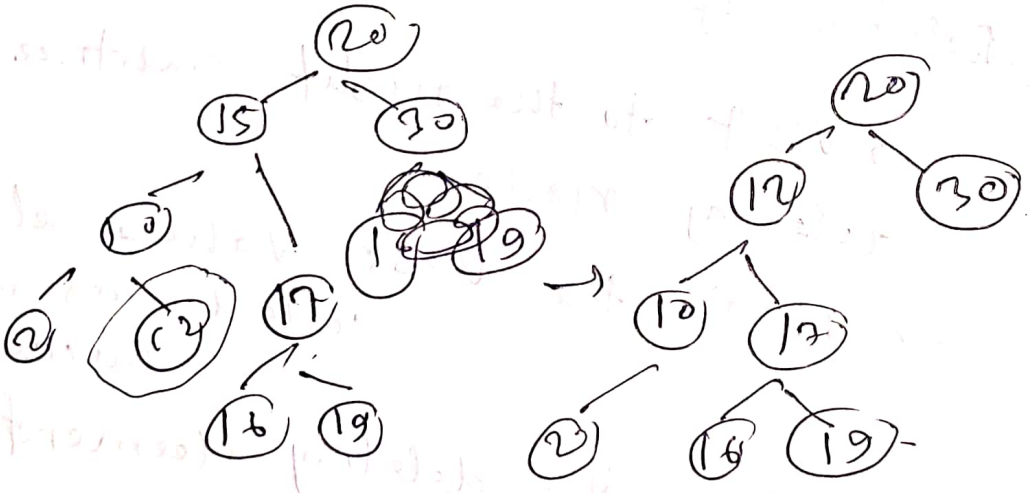


Methode - 2

→ Visit the left subtree.

→ Pluck the greatest value element called as inorder predecessor.

→ Replace the deleting element with its inorder predecessor.



Sub :

Day

Time : Date : / /

→ Time complexity of BST = O(h) ← height.

→ Worst case.

→ The binary search

→ Best case:

→ The binary search tree is a balanced binary tree

→ Height of BST becomes log(n)

→ Time complexity is O(log n)

Sub :

Day

Time : Date : / /

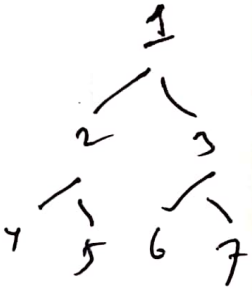
Traversals Using Stacks

Pre-order Traversal.

Steps:

1. Push the root into stack.
2. While the stack is not empty.
   a. Pop the top element and process it.
   b. push its right and then left child.

Example:



Steps:

node	stack	output
push 1	1	
pop, 1; push, 3, 2	3, 2	1
pop 2, push, 4, 5	3, 5, 4	1, 2
pop 4	3, 5	1, 2, 4
pop 5	3	1, 2, 4, 5
pop. 3, push 7, 6	7, 6	1, 2, 4, 5, 3
Pop all		1, 2, 4, 5, 3, 7

Sub :

Day

Time :

Date : / /

In order Traversal

Steps :

(i) Set current node = root. and push root.

(ii) Continue pushing the left child of current node until their is no left child. Then step 3.

(iii) If stack is not empty then,

(a) pop the top item from stack

(b) Print the popped item. Set current node = popped-item→right

(c) Go to step 2

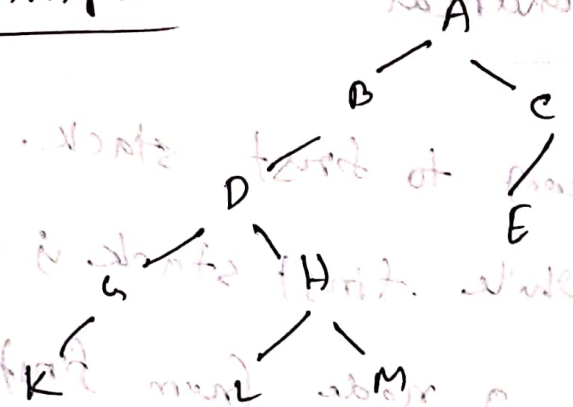
(iv) If current is NULL, and stack is empty. then we are done.

Sub :

Day

Time : Date : / /

Example:



Step	Stack	Print
push A	A	
push B	A, B	
push D	A, B, D	
push G	A, B, D, G	
push K	A, B, D, G, K	
pop till there is node with right child.	A, B	K, G, D
push H	A, B, H	K, G, D
push L	A, B, H, L	K, G, D
	A, B.	K, G, D, L, H
push M	A, B, M	K, G, D, L, H
		K, G D, L, H, M, B, A
push C	C	
push E	C, E	K, G, D, L, H, M, B, A, E, C

Sub :

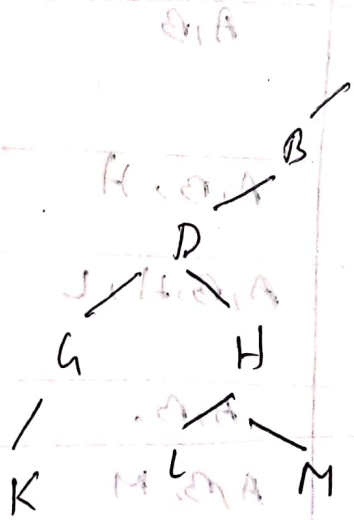
Day

Time : Date : / /

post-order Traversal

steps:

1. push root to first stack.
2. Loop while first stack is not empty,
   - (a) . pop a node from first stack
   - (b) push left and right children of the poped node to first stack.
3. Print contents of second stacks.



Sub :

Day

Time : Date : / /

Step	1st stack	2nd stack.
A	A	
pop A, push B, C	B, C	A
pop C, push E	B, E	A, C
pop E	B	A, C, E
pop B, push D	D	A, C, E, B
pop D, push G, H.	G, H	A, C, E, B, D
pop H, push L and M	G, L, M	A, C, E, B, D, H
pop M	G, L	A, C, E, B, D, H, M
pop L	G	A, C, E, B, D, H, M, L
pop G, push K	K	A, C, E, B, D, H, M, G L, G
pop K		A, C, E, B, D, H, M, L, G, K

Sub :

Day

Time : Date : / /

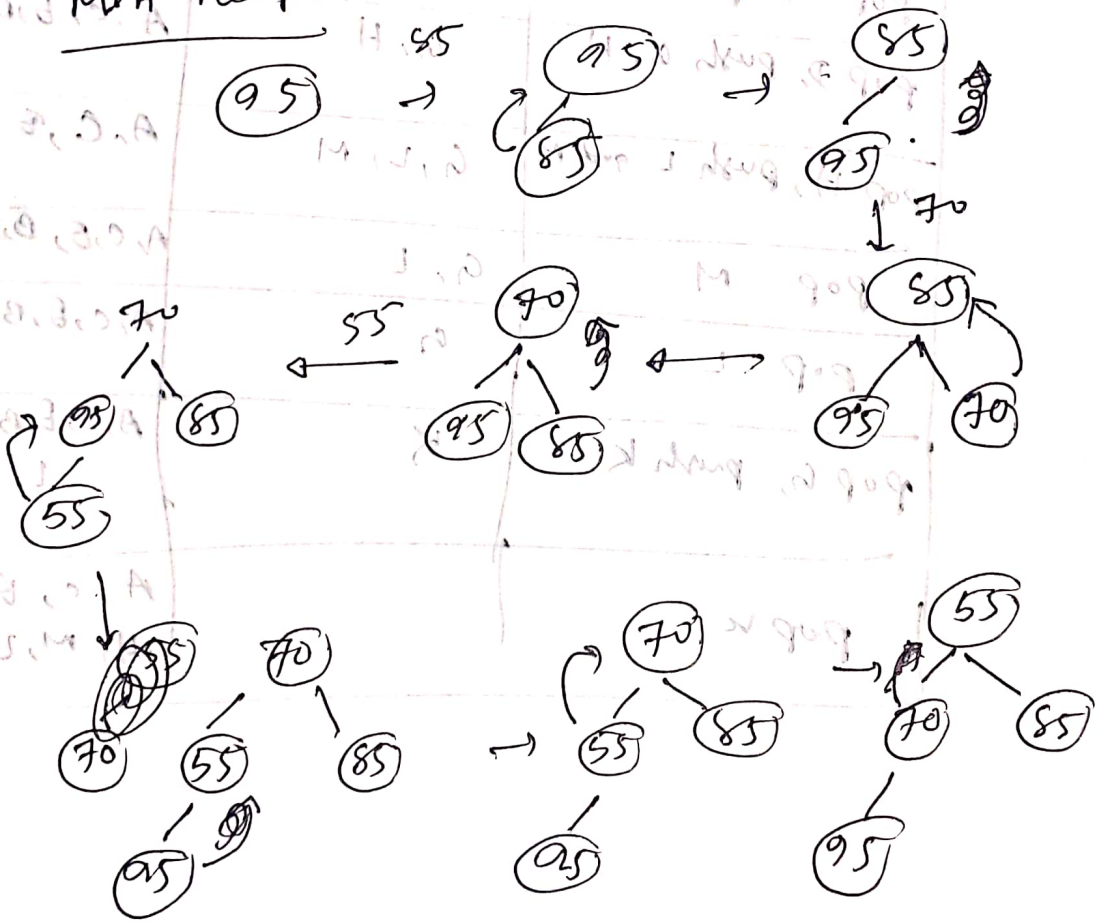
Heap sort

Given element 95, 85, 70, 55, 33, 30, 65,

→ construct the min-heap.

→ Heap sort.

Min heap

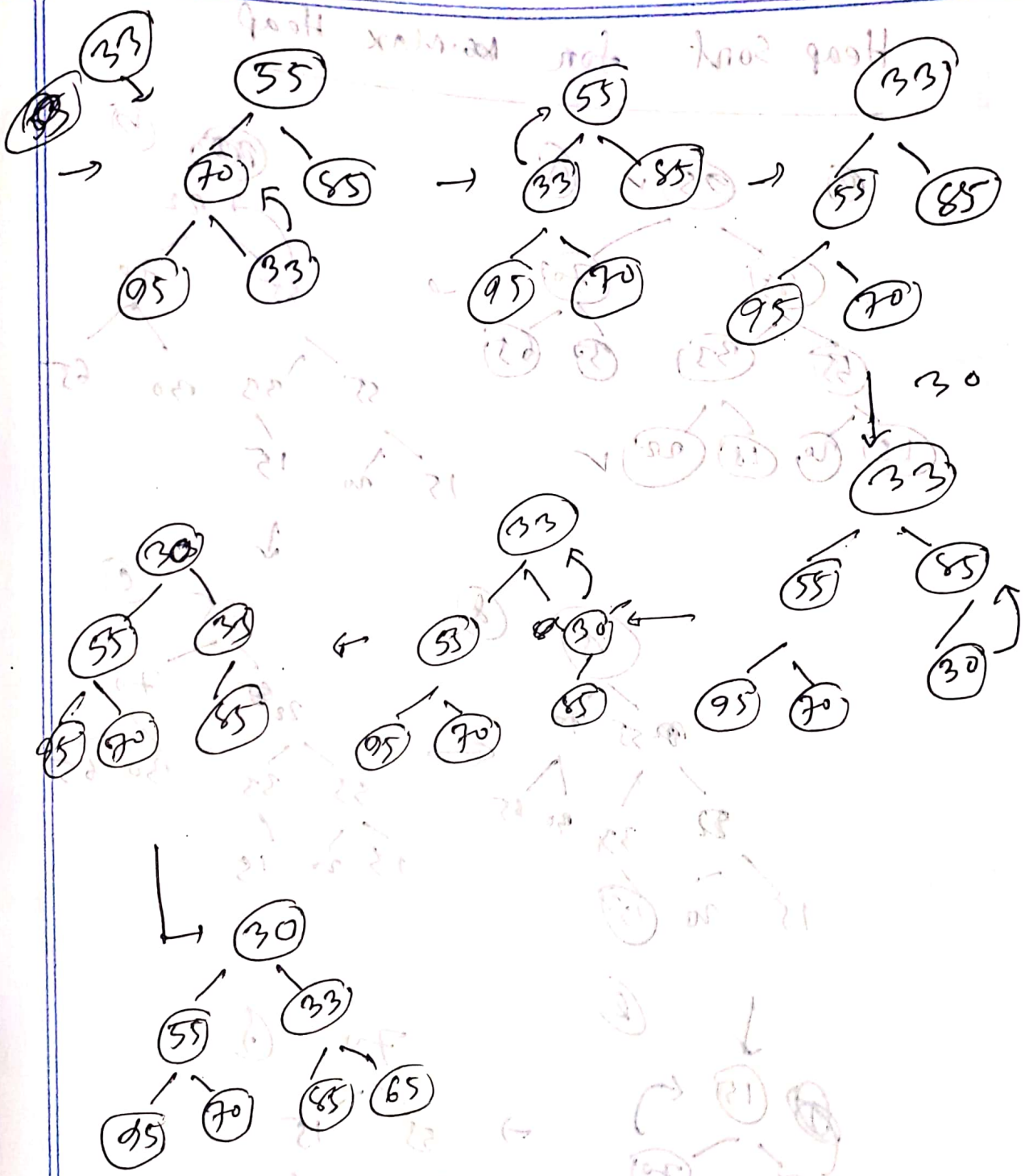


Sub :

Day

Time :

Date : / /



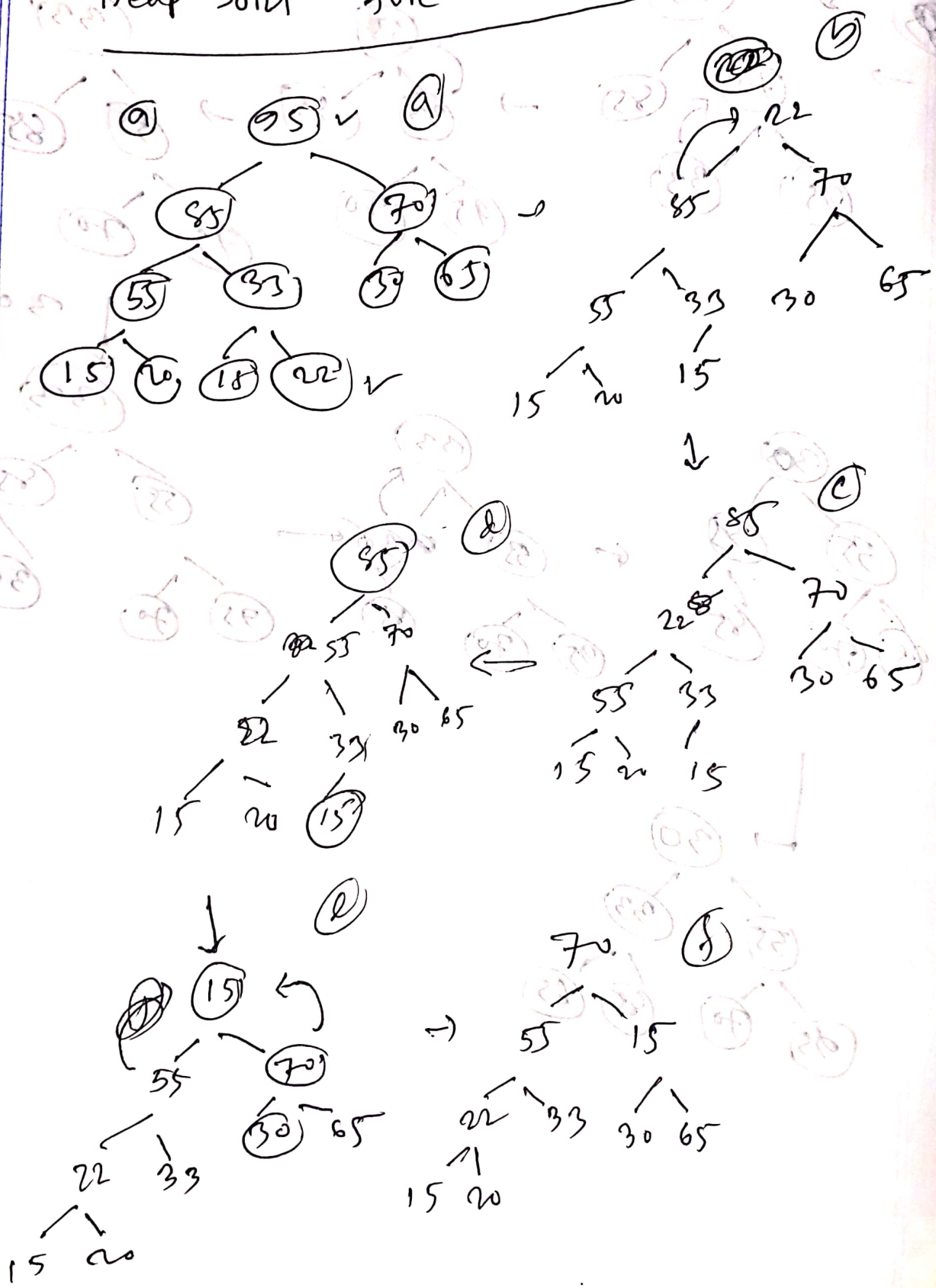
Sub :

Day

Time :

Date : / /

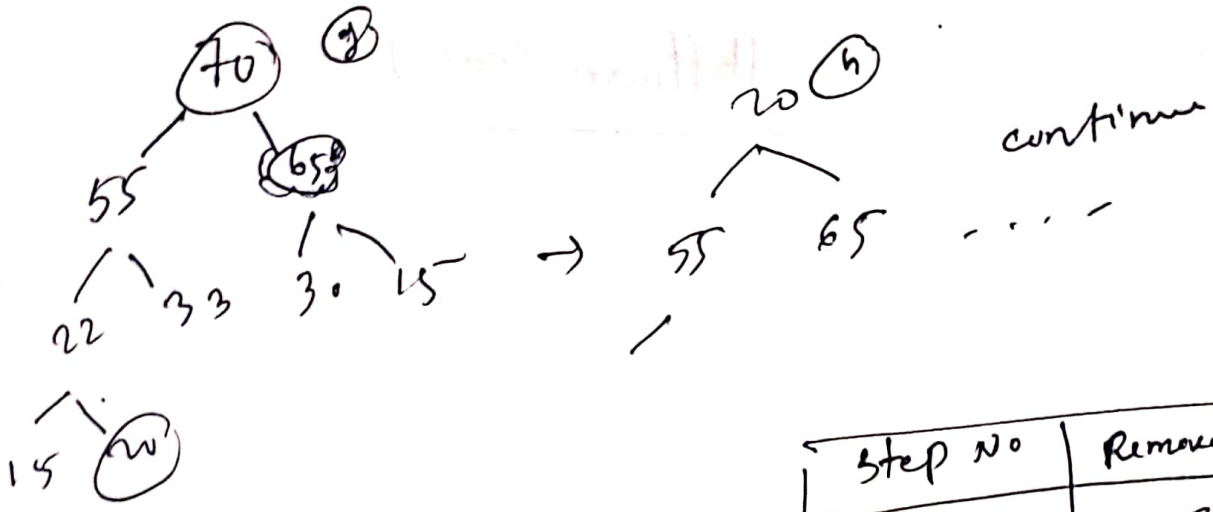
Heap Sort for Max Heap



Sub :

Day

Time : Date : / /



Step No	Removed element
(a)	95
(b)	85
(g)	70
(k)	65
(n)	55
(q)	33
(s)	30
(u)	22
(w)	20
(x)	15
(y)	15