Date: 25-6-18
MON ☑ TUE ☐ WED ☐ THU ☐ FRI ☐ SAT ☐

Chapter-1

Sparse matrix is a matrix that has many elements with a value zero in order to efficiently utilize the memory specialized algorithm and data structure take advantage of the sparse matrix should be used.

Sparse matrix can easily compressed which in turn can significantly reduce memory used.

Two types of sparse matrix

1. Lower triangular
2. Upper triangular

1. Lower Triangular Matrix : In this first type of sparse matrix all the elements above the main diagonal have a value zero.

Ex:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 5 & 2 & 0 & 0 \\ -9 & 7 & 3 & 0 \\ 4 & 6 & 8 & 4 \end{bmatrix}$$

In a lower triangular matrix.

$A_{i,j} = 0 \quad i < j$

And nxn lower triangular matrix A has one non-zero element in the first row, two non-zero element in the second row, likewise 'n' non-zero element in the n^{th} row.

Date:
MON TUE WED THU FRI SAT

the mapping between 2d matrix and 1d matrix can be done in any one of the following ways:

1. Rowwise mapping
   A[] = {1, 5, 2, -9, 7, 3, 4, 6, 8, 4}

2. Columnwise mapping
   A[] = {1, 5, -9, 4, 2, 7, 6, 3, 8, 4}

2. Upper Triangular Matrix: All the elements below the main diagonal have the value ZERO.
   $A_{ij} = 0 \quad ; \quad i > j$

3. Tri-diagonal Matrix: In Tri-diagonal matrix, elements with the non-zero values can appear only on the diagonal of immediate above or below diagonal hence in a tri-diagonal matrix.
   $A_{ij} = 0$; where $|i-j| > 1$

Ex:

4	1				
5	1	2			
	9	3	1		
		9	2	2	
			5	9	9
				6	7

a) In main diagonal, it contains non-zero elements for $i=j$. In all there will be 'n' elements.

Date:
MON TUE WED THU FRI SAT

b) Below the main diagonal, it contains non-zero elements for $i=j+1$, in all there will be $n-1$ elements.

c) Above the main diagonal, it consist non-zero elements for $i=j+1$, in all there will be $n-1$ elements.

→ Row wise

{4, 1, 5, 1, 2, 9, 3, 1, 9, 2, 2, 5, 9, 9, 6, 7}

→ Column wise

{4, 5, 1, 1, 9, 2, 3, 9, 1, 2, 5, 2, 9, 6, 9, 7}

→ Diagonal wise

{5, 9, 9, 5, 6, 4, 1, 3, 2, 9, 7, 1, 2, 1, 2, 9}

Stack : One of the most important linear data structure of variable size is stack.

5	6	10	12	
0	1	2	3	4

There are 3 operations on Stack

→ PUSH

→ POP

→ PEEP

A Pointer TOP keeps track of topmost element of stack. Initially when the stack is empty the TOP has value ZERO, if stacks contains a single element

Date:
MON TUE WED THU FRI SAT

Top has a value 1 and so on.

- PUSH (Vector S, Top, x)

1. [check for stack overflow]
   if top ≥ N
   Print("Stack overflow")
   Return

2. [Increment Top]
   Top ← Top+1

3. [insert element]
   S[Top] ← x
   S[Top] →

d	Vector having N elements
c	
b	
a	

4. [Finished]
   Return

1. a, b, c, d, e, N = 5, Top

i) Top ≥ N
   0 ≥ 5
   False
   Top = 1
   S[1] = a

e	Top = 5
d	
c	
b	
a	

ii) Top ≥ N
   1 ≥ 5 False
   Top = 2
   S[2] = b

Date:
MON TUE WED THU FRI SAT

iii) 2 ≥ 5 False
Top = 3
S[3] = c

iv) 3 ≥ 5 False
Top = 4
S[4] = d

v) 4 ≥ 5 False
Top = 5.
S[5] = e

vi) 5 ≥ 5 True
Stack overflow

vii) ~~If N = 6, then~~

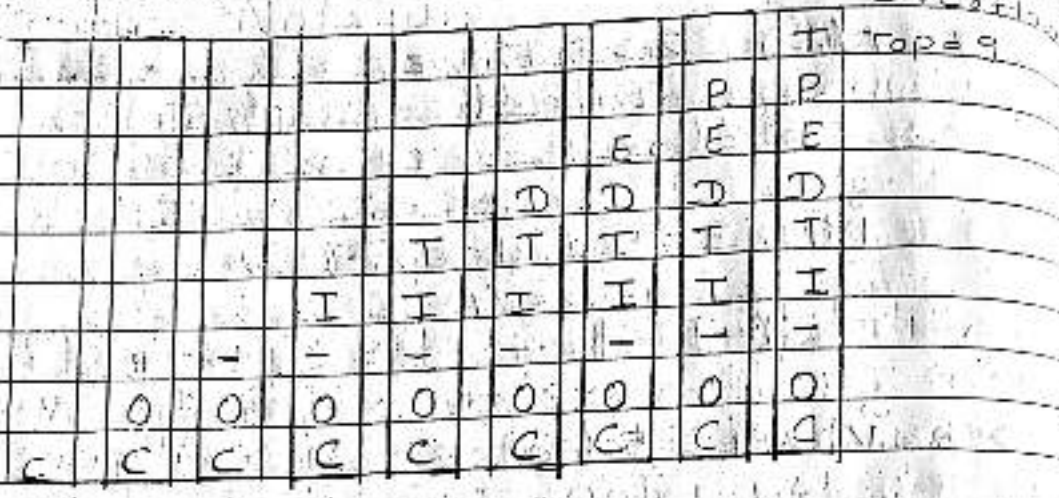
~~6~~ 1

- Reverse a string using stack.

CO-IT DEPT

i) Top = 1
S[1] = C

Date:-
MON TUE WED THU FRI SAT

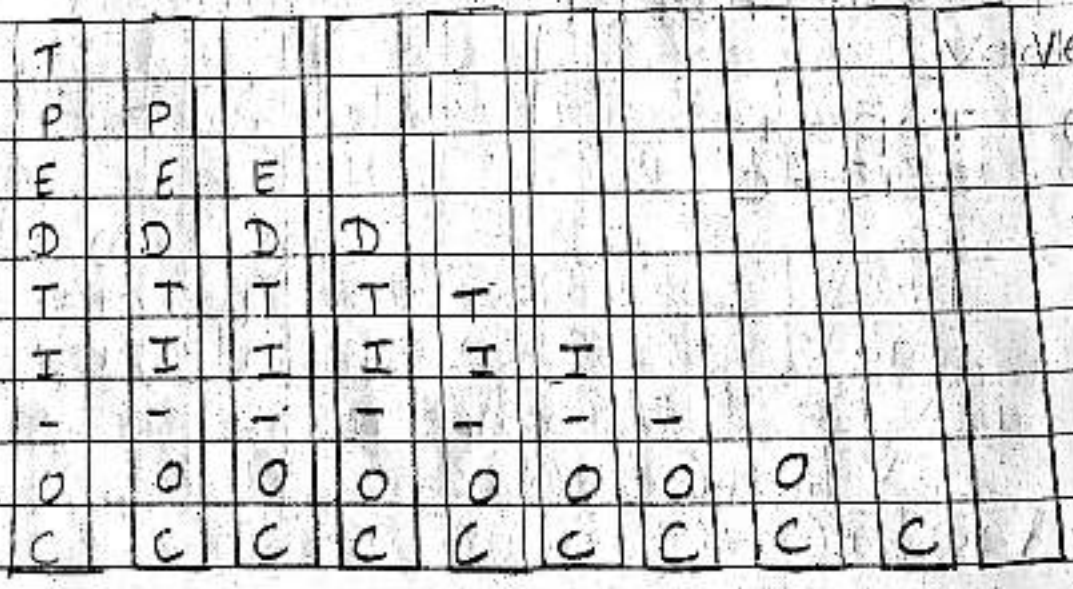


POP (S, Top)

1. [check for underflow]
   if Top == -1
   then write ('STACK UNDEFLOW')

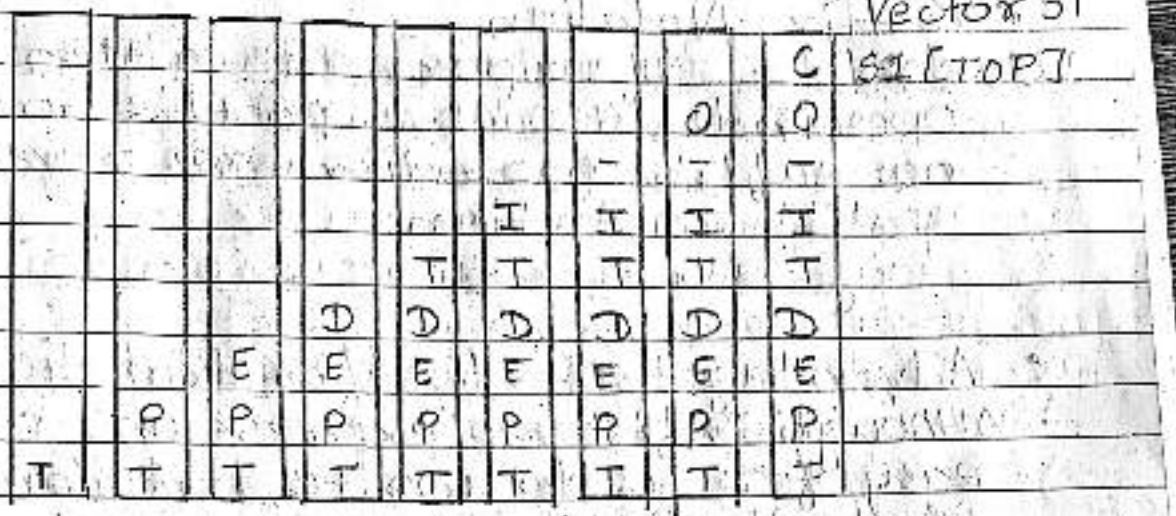
2. [Decrement Pointer]
   Top ← Top - 1

3. [Return element]
   Return [S[Top] + 1]



Date:
MON TUE WED THU FRI SAT

Vector S1



- PEEP

1. [check for stack underflow]
   if Top-i+1 ≤ 0
   print ('Stack Underflow on Peep!)

2. [Return i^{th} element from top of stack]
   Return S[Top]-[+1]

- Infix Notation
  The operator written in bet^n the operand are called infisc notation. Eg: a+b

- Prefix Notation
  The operator written before the operands is called prefix notation or polish notation.
  Eg: +ab

Date:
MON TUE WED THU FRI SAT

- Postfix Notation
  The operator written after operands is called postfix notation or suffix notation or reverse polish notation.
  Eg ab+

- Advantage of Postfix Notation
  - Although infix expressions are easy for us to write but Computer finds it difficult to parse because they need a lot of information (operator precedence, associativity, brackets) to evaluate the expressions so Computers work more efficiently with postfix & prefix notation.

brackets, Exponential ( ), ^ Higher precedence (Right to left)
Multiplications, division *, /, % Next (left to right)
modulo

- Addition, Subtraction +, - (left to right)

★ A + [(B+C) + (D+E) * F] / G]
= A + [[(CB+) + (DE+) * F] / G]
= A + [((CB+)+(DE+)F) *] / G]
= A + [(BC+)(DE+)F *) +] / G]
= A + [(BC+)((DE+)F*)+] G] /
- A[(BC+)(CED+)F*)+]G]/+

Date:
MON TUE WED THU FRI SAT

★ (A+B) * (C+D)
(AB+) * (CD+)
(AB+)(CD+)*

= A(BC+DE+F*+)G1/+ // Fully Parenthize expression

★ A↑B+C*D%E↑F
AB↑+C*D%EF↑
BA↑+DC*%EF↑
AB↑+DC*EF↑%
AB↑DC*EF↑%+

★ A+(B*C-CD/E^F)*G1)*H
A+(B*C-(D/EF^)*G1)*H
A+(B*C-DEF^/*G1)*H
A+(CB*-DEF^/*G1)*H
A+(CB*-DEF^/G1*)*H
A+(BC*DEF^/G1*-)*H
A+(BC*DEF^/G1*-)H*
A(BC*DEF^/G1*-)H*+
ABC*DEF^/G1*-H*+

A+(B*C-(D/^EF)*G1)*H
A+(B*C-/D^EF*G1)*H
A+(*BC-*/D^EFG1)*H
A+-*BC*/D^EFG1*H
A+*-*BC*/D^EFG1H
+A*-*BC*/D^EFG1HA

Date:
MON TUE WED THU FRI SAT

• How to Evaluate Suffix Notation
The method of evaluating suffix expression can be summerized by the four steps which are repeated applied until all operator haven been processed.

1) Find the left most operator in the expression.
2) Select the two operands immediately to the left of the operator found.
3) Perform the indicated operation.
4) Replace the operator & operand with the result. T_1

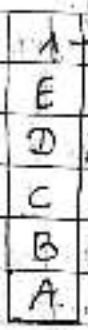
abc/d*+ a=5, b=4, c=2, d=2

			/				*				
		c	c			d	d		+		
	b	b	b	T_1		T_1	T_1		T_2		
a	a	a	a	a		a	a		a		

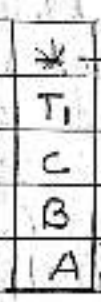
$T_1 = b/c = 2$
$T_2 = T_1 * d = 2*2 = 4$
$T_3 = a + T_2 = 5+4$
$= 9$

A-B/(C*D^E)
= A-B/(C*DE^)
= A-B/(CDE^*)
= A-BCDE^*/
= ABCDE^*/-

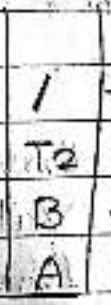
Date:
MON TUE WED THU FRI SAT



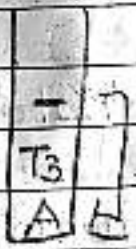
$D^1E = 2^1 = 2 = T_1$



$T_2 = C * T_1 = 2 * 2$
$= 4$



$= B / T_2 = 4/4$
$T_3 = 1$



$T_4 = 5 - 1 = 4$

Date:
MON TUE WED THU FRI SAT

- To Determine whether an expression is Valid we next associate a rank with each expression.

1. The Rank of a Symbol $S_i = 1$
2. The Rank of an operator $O_j = -1$
3. The Rank of an arbitary sequence of Symbols & operators is the sum of the ranks of the individual symbols & operators.

- Theorem

A Polish Suffix formula is well formed if & only if the rank of the formula is '1'.

1)	a + * b	0	Invalid
2)	a - b * c	1	Valid
3)	a + b / d -	0	Invalid

- Convert unparenthized infix into Suffix using stack.

Step 1. Initialize stack Contains to the Symbol #.

Step 2. Scan the left most symbol in the given infix expression & denote it as current Symbol.

Step 3. Repeat through Step-6 while the current input Symbol is #.

step 4.

step 5.

step 6.

S.S ≥ C.S

step 4. Remove an output all stacks symbols whose precedence value are greater than or equal to the precedence of the current input symbol.

step 5. Push the current input symbols on to the stack.

step 6. Scan the left most symbol in the infix expression & let it be the current input symbol.

SYMBOL	Precedence	Rank
+ -	1	-1
* /	2	-1
a,b,c	3	1
#	0	—

a+b*c-d/e*h

(character scanned)	Contents of stack (right most symbol is top of stack)	R P expression	Rank
	# ← TOP		
a	#a	-	-
+	#+	a	1
b	#+b	a	1
*	#+*	ab	2
c	#+*c	ab	2
-	#-	abc*+	2
d	#-d	abc*+	2

Date:
MON TUE WED THU FRI SAT

		abc*+d	3
/	#+-/	abc*+d	3
e	#+-/e	abc*+d	2
*	#+-*	e/	
		abc*+de	3 2
b	#-*b	/+	
		abc*+de	1
#	#	/n*-	

a/e*h-b+c

	#		
a	#a	a	1
/	#/	a	1
e	#/e	ae/	1
*	#*	ae/	1
h	#*h	ae/h*	1
-	#-	ae/h*	1
b	#-b	ae/h*b-	1
+	#+	ae/h*b-	1
c	#+c	ae/h*b-c+	1
#	#		

Date:
MON TUE WED THU FRI SAT

1. [Initialize Stack]
   Top ← 1
   S[TOP] ← '#'

2. [Initialize o/p string & rank count]
   POLISH ← ''
   RANK ← 0

3. [Get first i/p symbol]
   NEXT ← NEXTCHAR (INFIX)

4. [Translate the infix expression]
   Repeat through step 6 while NEXT ≠ #

5. [Remove symbols with greater or equal precedence from stack]

   Repeat while f(NEXT) ≤ f(S[TOP])
   TEMP ← POP(S, TOP)
   POLISH ← POLISH ⊕ TEMP
   RANK ← RANK + r(TEMP)
   if RANK < 1
   then write('Invalid') Exit

6. [Push current symbol or to stack & explain next Symbol]
   Call PUSH (S, TOP, NEXT).
   NEXT ← NEXT CHAR (INFIX)

7. [Remove remaining element from STACK]
Repeat while STACK[TOP] ≠ '#'
TEMP ← POP[STACK, TOP]
POLISH ← POLISH ⊕ TEMP

8. [Is the expression valid?]
if RANK = 1
then write ('VALID')
else write ('Invalid') & exit

a ↑ b ↑ c * d / e

	#	-	-
a	#a	-	1
↑	#↑	a	1
b	#↑b	a	1
↑	#↑	ab↑	1
c	#↑c	ab↑	1
*	#*	ab↑c↑	1
d	#*d	ab↑c↑	1
/	#/	ab↑c↑d*	1
e	#/e	ab↑c↑d*	1
#	#	ab↑c↑d*e/	1

Symbol	Input Precedence fun^n f	Stack Precedence fun^n g	Rank fun r
+, -	1	2	-1
*, /	3	4	-1
↑	6	5	-1
Variable	7	8	1
(	9	0	-
)	0	-	-

(a+b↑c↑d)*(e+f/d))

Current Symbol	Stack Contents	R.P Exp	Rank
	(		
(	((	-	-
a	((a	-	-
+	((+	a	1
b	((+b	a	1
↑	((+↑	ab	2
c	((+↑c	ab	2
↑	((+↑↑	abc	3
d	((+↑↑d	abc	3
)	( )	abcd↑↑+((	1
*	(*	abcd↑↑+	1
(	(*(	abcd↑↑+	1
e	(*(e	abcd↑↑+	1
+	(*(+	abcd↑↑+e	2
f	(*(+f	abcd↑↑+e	2
/	(*(+/	abcd↑↑+ef	3
d	(*(+/d	abcd↑↑+ef	3
)	(*	abcd↑↑+efd/	1
)		abcd↑↑+efd/+*	

Date:
MON TUE WED THU FR SAT

(a+b) * (c↑d)

	C		
(	C(	—	—
a	C(a	—	—
+	C(+	a	1
b	C(+b	a	1
)	C	ab+	1
*	C*	ab+	1
(	C*(	ab+	1
c	C*(c	ab+	1
↑	C*(↑	ab+c	2
d	C*(↑d	ab+c	2
)	C	ab+cd↑*	1

1. [Initialize Stack]
   top = 1
   S[top] = 'C'

2. [Initiculize Output String & Rank Count]
   polish = ''
   Rank = 0

3. [Get First Input Symbol]
   Next ← Nextchar (Infix)

4. [Translate the infix Expression]
   Repeat through Step-7 while
   Next ≠ '\0'

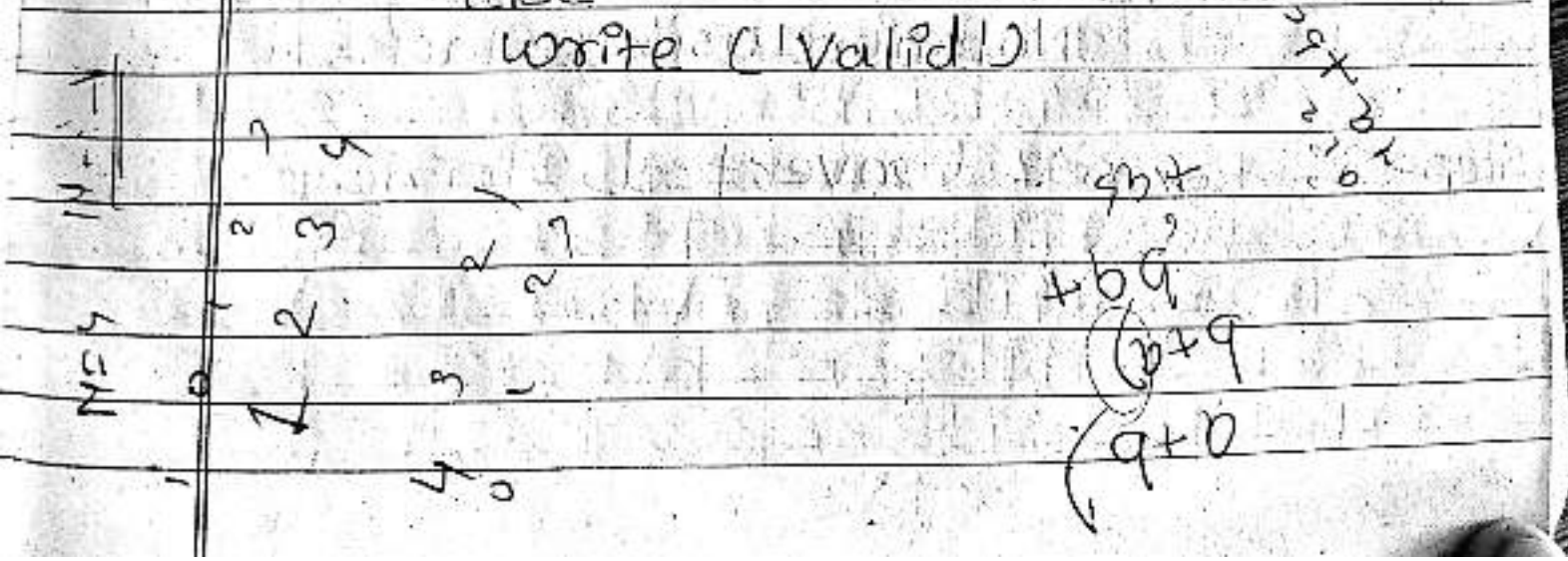
Date: MON TUE WED THU FRI SAT

5. [Remove Symbols with greater precedence]
   If top < 1 then
   write ('Invalid') Exit
   Repeat while (F(Next) < G(S(Top))
   TEMP ← POP(S, TOP)
   POLISH ← POLISH ∘ TEMP
   RANK ← RANK + r(TEMP)
   If RANK < 1
   then write ('Invalid') Exit

6. [Are there matching Parenthesis?]
   If F(Next) ≠ G(S(TOP))
   then Call PUSH(S, TOP, Next)
   else
   pop(S, TOP)

7. [Get Next Input Symbol]
   Next ← Nextchar (Infix)

8. [Is the expression valid]
   If Top ≠ 0 or Rank ≠ 1
   then write ('Invalid')
   else
   write ('Valid')



Convert infix into postfix using stack,

a-b/(c*d↑e))

(	(	-	-
a	(a	a	1
-	(-	a	1
b	(-b	ab	2
/	(-/	ab	2
(	(-/(	ab	2
c	(-/(c	abc	3
*	(-/(*	abc	3
d	(-/(*d	abcd	4
↑	(-/(*↑	abcd	4
e	(-/(*↑e	abcde↑*/-	11
)	(		

Step-1: Reverse the infix exp

Ex: (a+b)*(c-d)

(d-c)*(b+a)

Step-2: Convert in above ⇑ string into RP Notation using stack

Step-3: Reverse converted String.

Date: MON TUE WED THU FRI SAT

(e↑d*c)/b-a)

	C		
(	(C	-	-
e	(Ce	÷	-
↑	(C↑	e	1
d	(C↑d	e	1
*	(C*	ed↑	1
c	(C*c	ed↑	1
)	C	ed↑c*	1
/	C/	ed↑c*	1
b	C/b	ed↑c*-	1
-	C-b-	ed↑c*b/	1
a	C-a	ed↑c*b/	1
)	C	ed↑c*b/a-	1

-a/b*c↑de)

a+(b*c-(d/e↑f)*g)*h)

h*(g*(f↑e/d)-c*b)+a)

	C		
h	Ch	-	-
*	Ch*	h	1
(	(*C	h	1
g	(*Cg	h	1
*	(*(*	hg	2
c	(*(*c	hg	2
f	(*(*cf	hg	2
↑	(*(*(↑	hgf	3
e	(*(*(↑e	hgf	3
/	(*(*(↑/	hgfe	4

d	C*C*C↑/d	hgfe	4
)	C*C*C	hgfed/↑	3
-	C*C*C-	hgfed/↑	3
c	C*C*C-c	hgfed/↑	3
*	C*C*C-*	hgfed/↑c	4
b	C*C*C-*b	hgfed/↑c	4
)	C*C*C	hgfed/↑cb*-	3
+	C*C*C+	hgfed/↑cb*-	3
a	C*C*C+a	hgfed/↑cd*-	3
)	C*C*C	hgfed/↑cd*-a+	3
)	C*	hgfed/↑cd*-a+	1

**

**+a-*dc↑/defgh

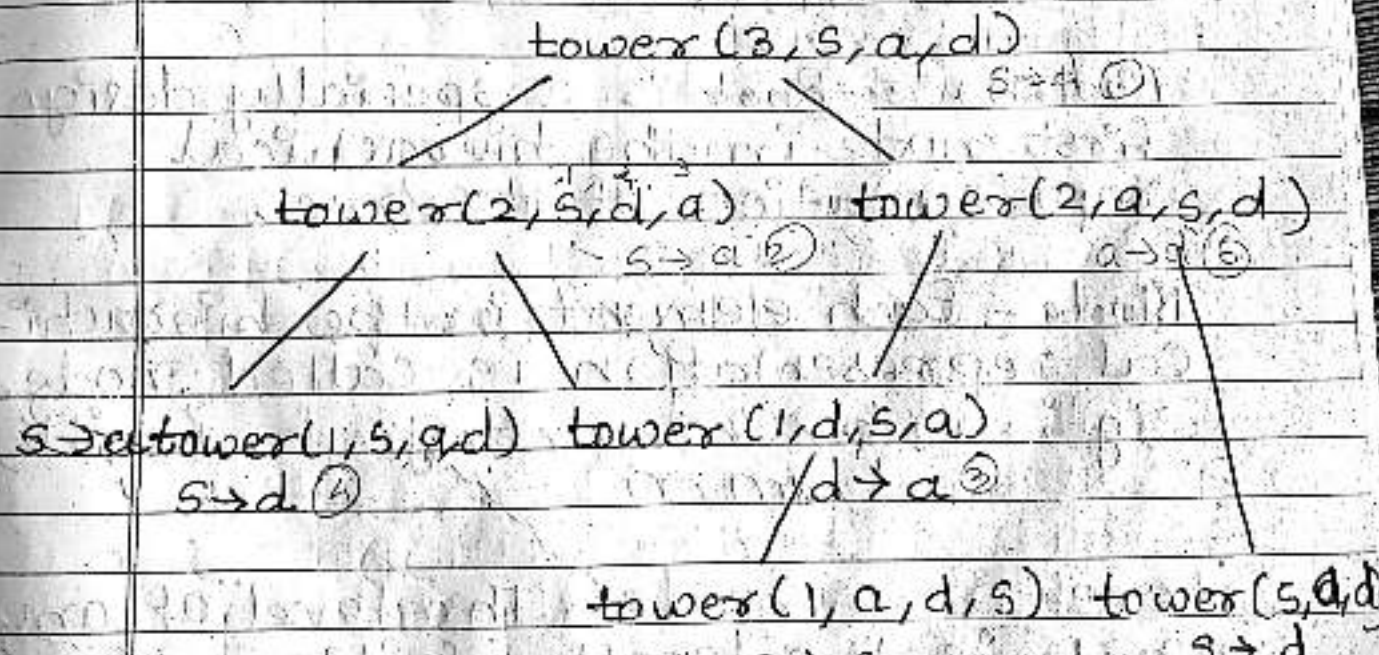
Rules for infix into prefix

1. Do not pop or remove the content of Stack when we recognize the operator having same priority

Towers of Hanoi

1) Move n-1 disk from S to A using D
2) Move n disk from S to D
3) Move n-1 disk from A to D using S

```
tower (n-1, s, a, d)
{
  if (n == 1)
  {
    print "s → d"
    return
  }
call tower (n-1, s, d, a)
    print "s → d"
call tower (n-1, a, s, d)
    return
}
```



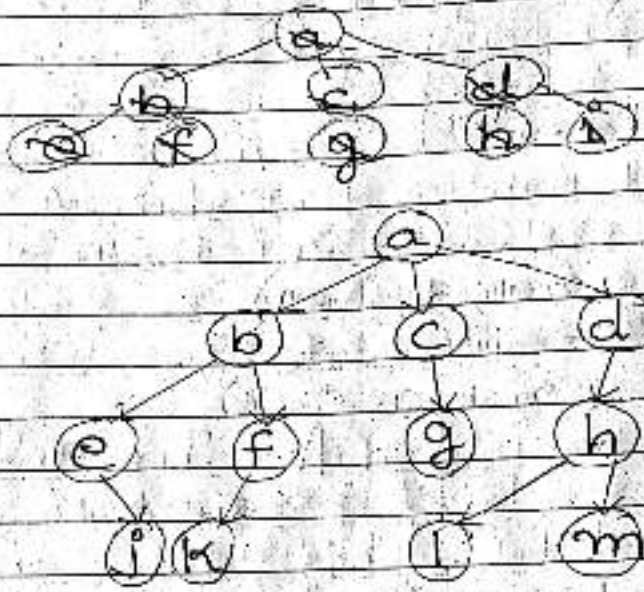
Date:
MON TUE WED THU FRI SAT

9/8/18

Chapter - 2

TREE AND Its Concepts

Tree



Root node - Root is a specially design first node in the hierachical representation of the tree.

Node - Each element in the hierachical representation is called node
Eg: a, b, c, d, e, f, g, h, i, j, k, l, m, n

Level of a node - The level of any node is the length of it's path from the root because root has a zero distance from it's self level of the root is zero. The children of the root are at level one.

Date :
MON TUE WED THU FRI SAT

Degree of node - No. of subtree of a node is called degree of node
The degree of node a is 3. b = 2, c = 1, d = 2, e = 1, f = 1, g = 0, h = 2, i = 1, l = 0, m = 0, n = 0

Indegree - when the branches is directed toward node it is an in-degree of node. It means no. of edges arriving at node.
for ex: Indegree of root node = 0
b = 1, c = 1, d = 1, e = 1, and so on.

Outdegree - when the branches is directed away from the node it is an outdegree of branch. it means no. of edges leaving the node
for ex: outdegree of a = 3, b = 2, c = 1 and so on.

Degree of Tree : The degree of tree is max. degree of a node in a tree.
for ex: max. degree of node a = 3.
So degree of tree = 3.

Edges - connecting link bet^n any two nodes is called edges. (any tree with 'N' nodes their will be max. N-1 edges)

Leaf node - A node which has outdegree zero is called terminal node or leaf
for Ex: g, j, k, l, m, n

Internal nodes / Non-terminal nodes
A node with at least one child is called internal / Non-terminal nodes (except root node)
for Ex: b, c, d, e, f, h, i

Parent: A node is a parent it has successor (child node) node that is if it has outdegree greater than zero.
Ex: a, b, c, d, e, f, h, i

Child: A node with predecessor is a child. A child has indegree one
Ex: b, c, d, e, f, g, h, i, j, k, l, m, n

Sibilings: Two or more nodes with same parents are called as siblings it means all nodes that are at same level and share same parents are called as siblings.
ex; (b, c, d), (e, f), (h, i), (l, m)

Path: A sequence of consecutive edges is called as Path. (Tool)
Ex: a, d, h, l

Ancestor: Ancestor of a node are all the nodes alone the path from the root to that nodes.
for Ancestor of i → a, d, h

Decendent: All nodes in the path from a given node to a leaf.
for ex: decendent of node d
→ h, i, l, m, n

Null / Empty tree: A tree with no nodes is called null / empty tree.

Height of a tree node: The total no. of edges from leaf node to a particular node in longest path is called as height of that node

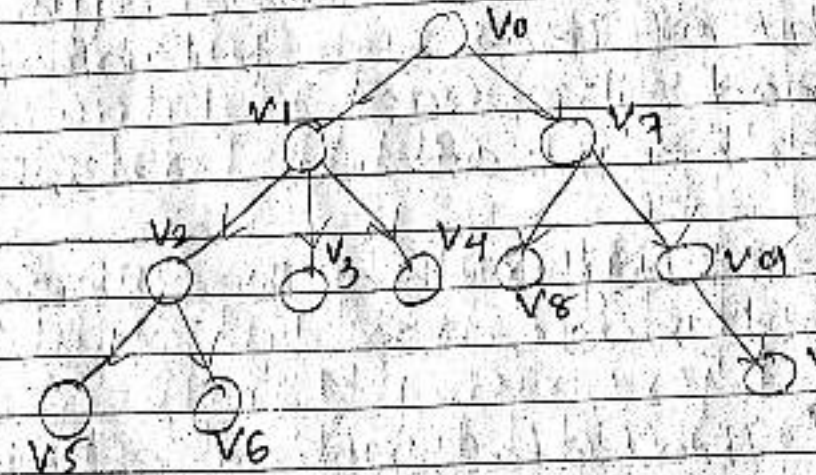
Height of tree: In a tree height of the root node is said to be height of that tree. height of all leaf node is zero.
for ex: height of node a = 3, b = 2, d = 2, c = 1
so that height of a tree = height of root node = 3

Depth: The total no. of edges from root node to a particular node is called as depth of that node.

Depth of a tree: In a tree the total no. of edges from root node to a leaf node in the longest path is said to be depth of the tree.
for ex: depth of node $l = 3, i = 2$
Depth of root node is zero

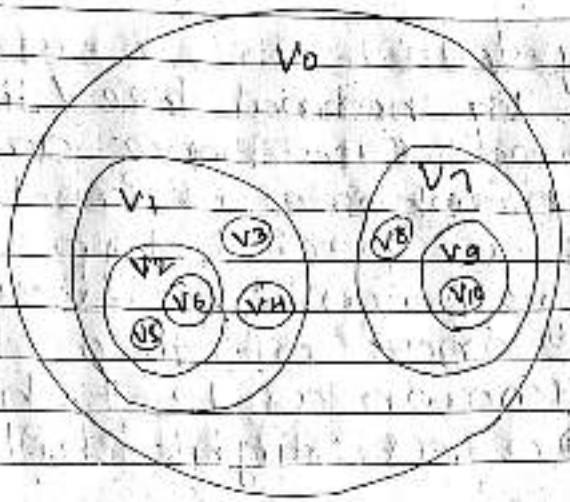
Forest: It is a group of disjoint tree if we remove a root node from a tree it becomes the forest.

Graphical Representation of a tree
There are four methods to represent tree
1) Ven diagrams to show subtrees
2) Nesting Parenthesis
3) Table of Contains (ex: book)
4) Level no. format

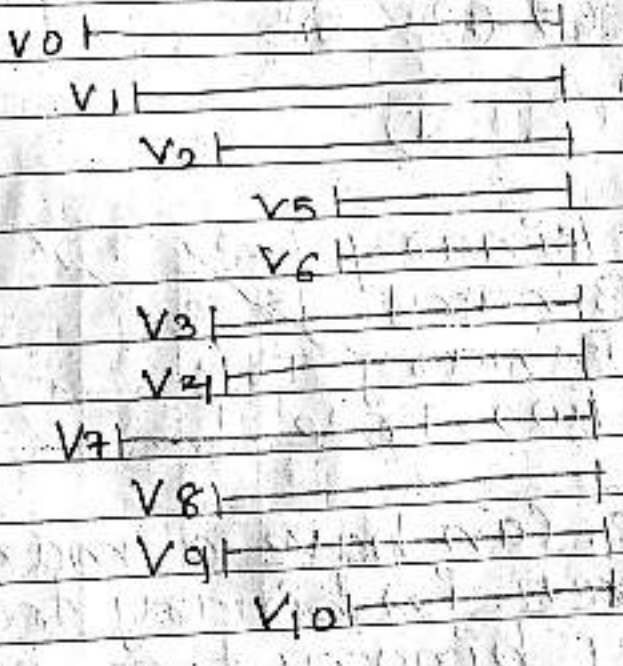


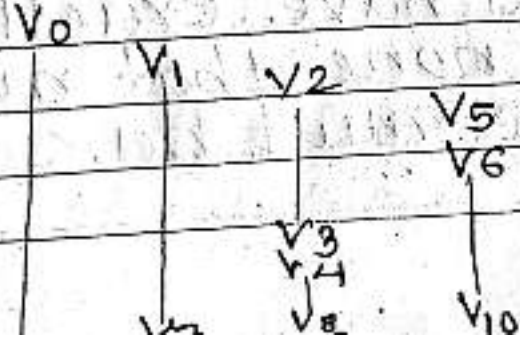
(V7(V9(V10))(V8))

Date:
MON TUE WED THU FRI SAT

① 

② (V0 (V1 (V2 (V5)(V6)) (V3)(V4))
(V7 (V9(V10)) (V8)))

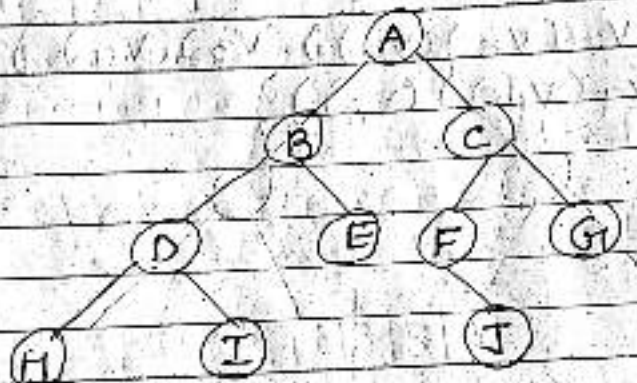
③ 

④ 

Date:
MON TUE WED THU FRI SAT

Binary tree

In normal tree / simple tree every node can have any no. of children whereas in binary tree is a special type of data structure in which every node can have max. of two children one is known as left child & other is known as right child.



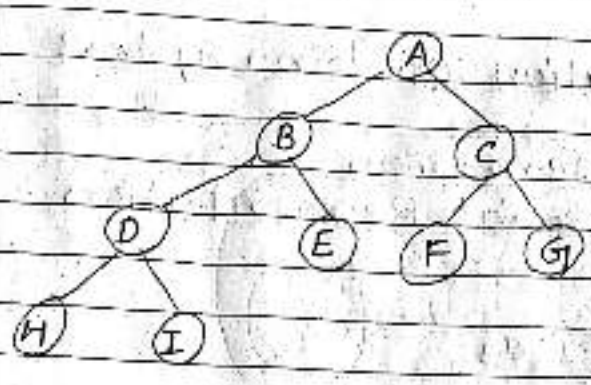
Types of Binary Tree

1) Strictly Binary Tree
2) Complete Binary Tree
3) Extended / Two Tree

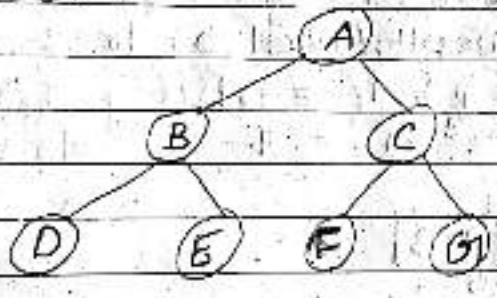
1) ⇒ Every node can have a max. of two children in binary tree but in strictly binary tree every node should have exactly two children or none that means internal node must have exactly 2 children.

2) ⇒

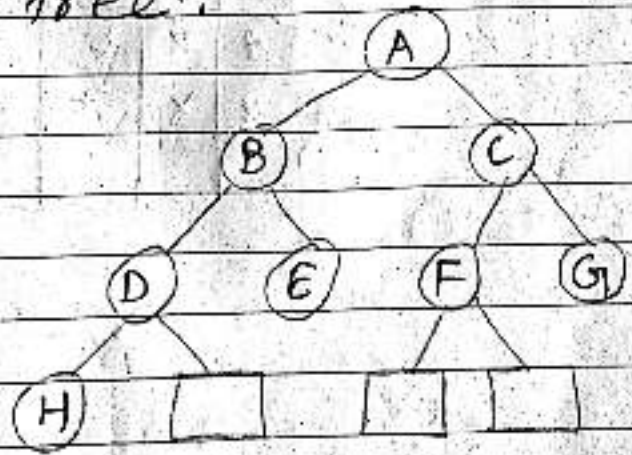
3) ⇒



2) ⇒ A binary tree in which every internal node has exactly two children & all leaf node are at same level. is called Complete/perfect binary tree.



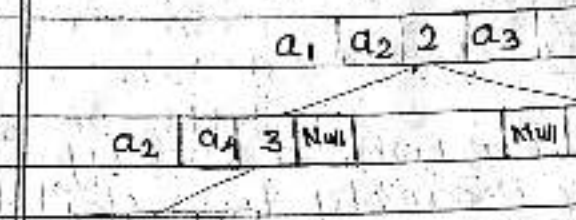
3) ⇒ A Full binary tree adding dummy nodes to a binary tree is called as extended binary tree or 2 tree.



Representation of binary tree

1) Linked Representation
2) Linear / Sequential Representation

1)



Maximum number of nodes allowed in binary tree = $2^{D+1} - 1$

D = Depth/level of tree

for ex: D = 4 = $2^5 - 1 = 31$

D = 2 = $2^3 - 1 = 7$

2)

2	3	5	7	B	B	B
0	1	2	3	4	5	6

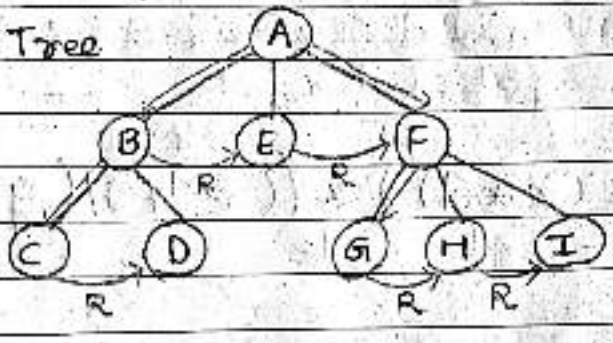
1	X		
2			
3	3	2	5
4	7	3	X
5	X	5	X
6	X	7	X
7			

Date:
MON TUE WED THU FRI SAT

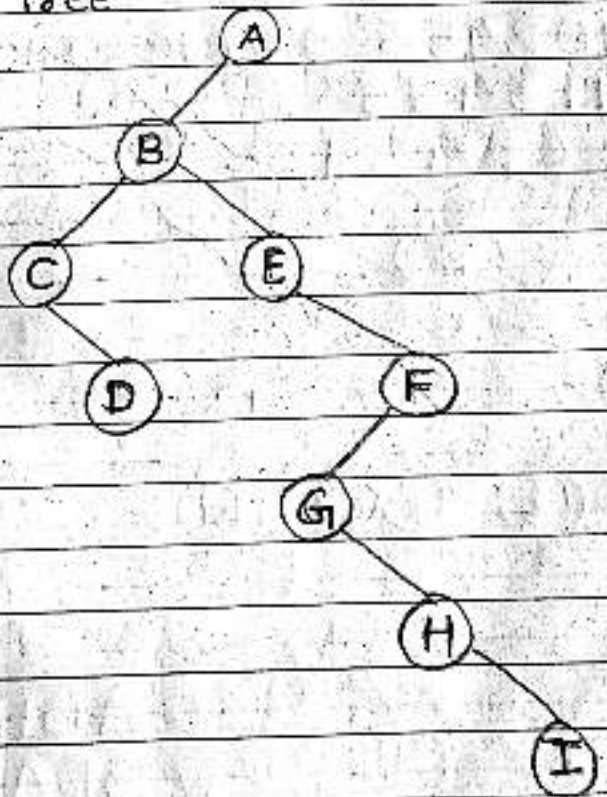
How to Convert general tree into binary tree.

1] All nodes of general tree will be nodes of binary tree.
2] Root of general tree is root of binary tree
3] Find branch parent to left most child.
4] Connect Siblings of each node from left to right child
5] Delete all link from node to it's children

General Tree



Binary Tree

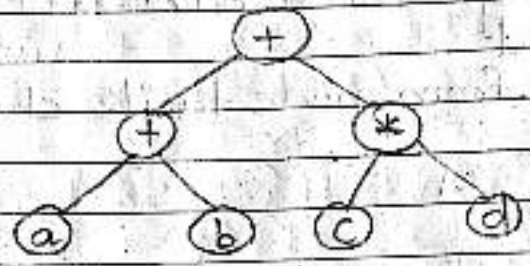


Date:
MON TUE WED THU FRI SAT

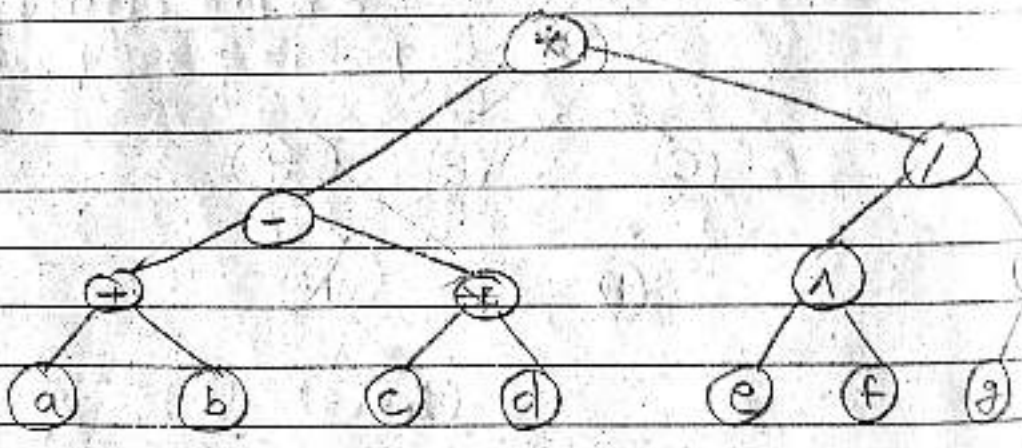
Expression Tree

The tree which stores algebraic expression is called expression tree. where all operands will be the leaf node & operators will be internal node.

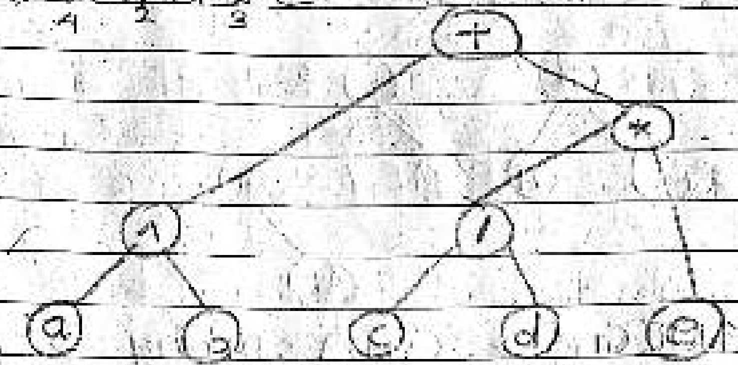
(a+b) * (c*d)



(a+b) - (c*d) * ((e^f)/(g-h))



$a \uparrow b + c / d * e$



1) Preorder → Root Left Right
2) Inorder → Left Root Right
3) Postorder → Left Right Root

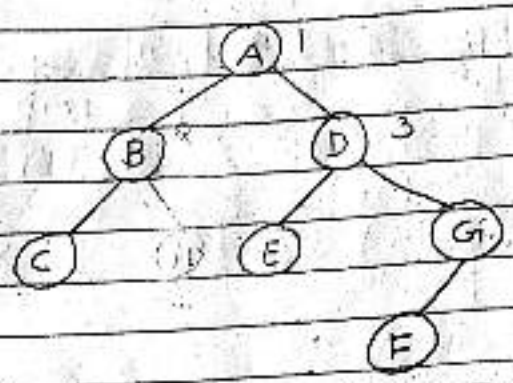
One of the most common operation performed on tree structure is there of traverse this is a procedure by which each node is processed exactly once in a systematic manner. There are two ways to define order

i) Recursive
ii) Iterative 3

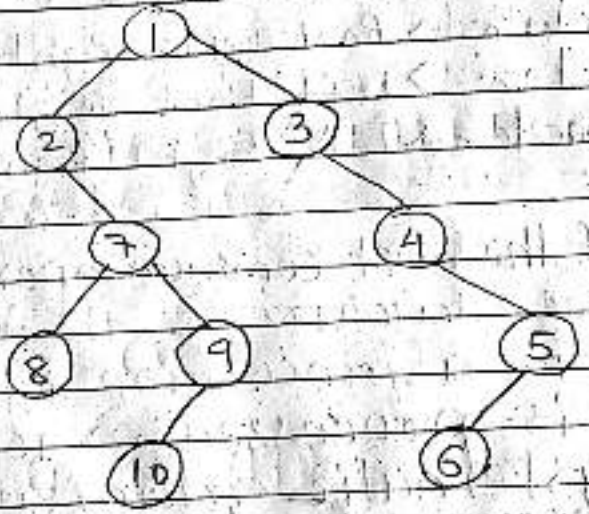
There are two ways in order for traversing a tree

1) → Process the root node
Traverse the left sub tree
Traverse the right sub tree

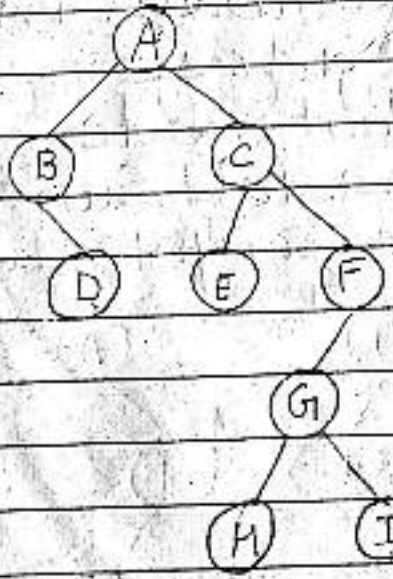
Date:
MON TUE WED THU FRI SAT



⇒ ABCHDEGF ⇒ CBAEDFG ⇒ CBDAEFG



Pre ⇒ 1278910 3456
In ⇒ 2871091 3465
⇒ 8109726543 1



⇒ ABDCEFGHI
⇒ BDAECHGIFF

Date:
MON TUE WED THU FRI SAT

Recursive PREORDER

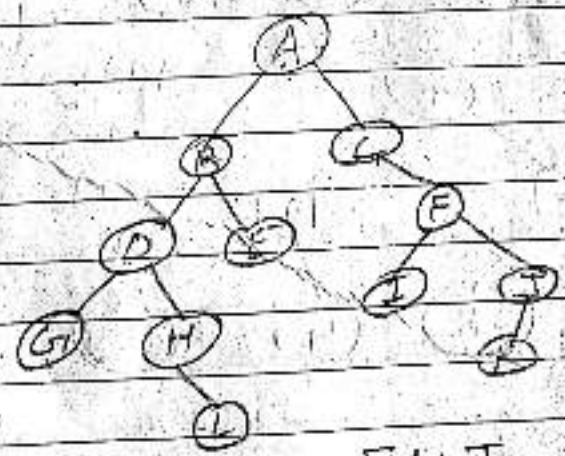
RPRORDER (T) → Address of root node

1) [Process the root node]
   if T ≠ NULL
   then write (DATA(T))
   else Return

2) [Process the left Subtree]
   if LPTR(T) ≠ NULL
   then call (RPREORDER (LPTR(T))

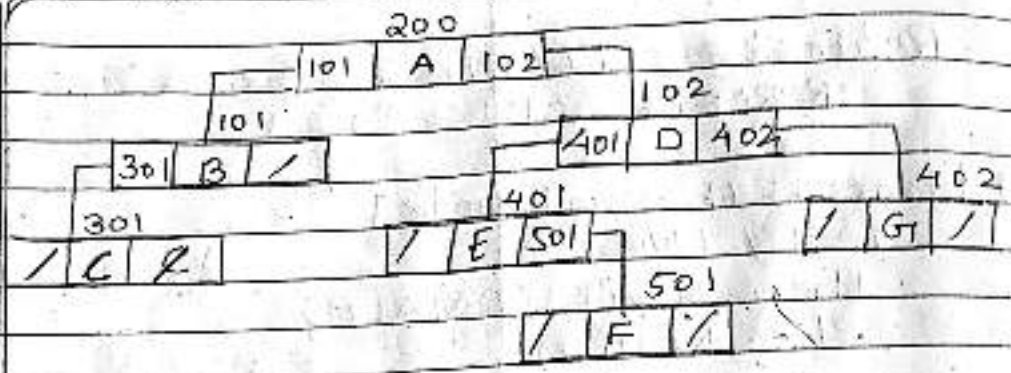
3) [Process the right Subtree] if RPTR(T) ≠ NULL
   then call (RPREORDER (RPTR(T))

4) [Finished]
   Return



In ⇒ GDBHLBEACIFKJ

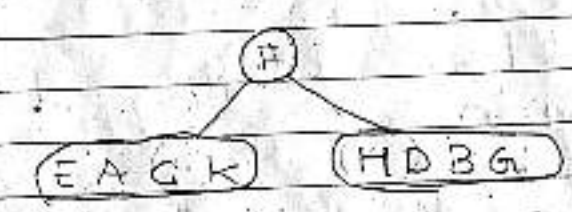
Pr ⇒ ABDGHLECFIJK

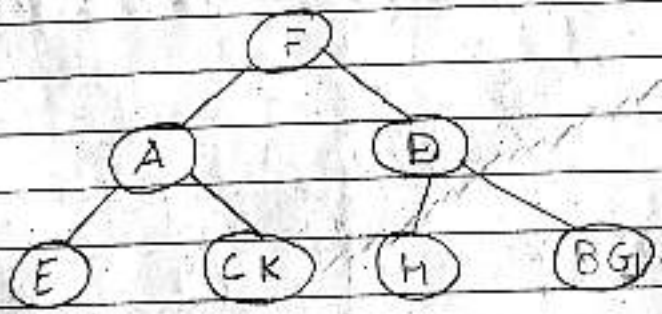


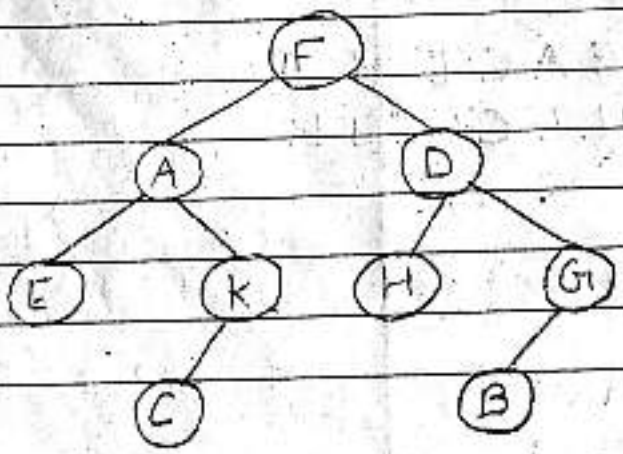
Draw the tree where inorder & preorder are given

Inorder: EACKFHDBG

Preorder: FAEKCDHGB



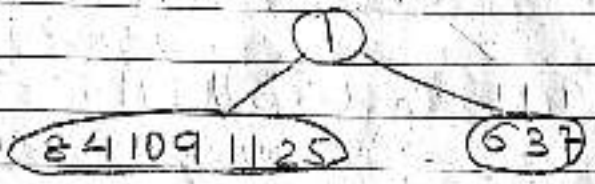




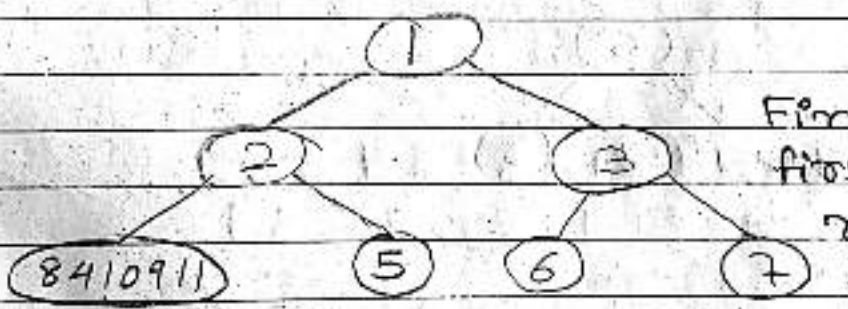
→ Step-1 Find the root node from Preorder
→ Step-2 Find the child (i.e left child & right child) from Inorder.
→ Step-3 Find the nearest left node from root node & made it left node.

RLG
left right root

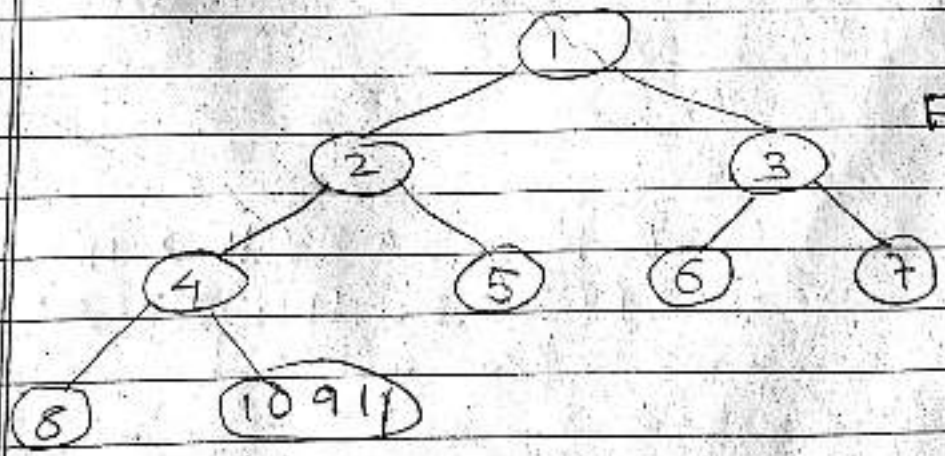
Preorder : 1 2 4 8 9 10 11 5 3 6 7
Inorder : 8 4 10 9 11 2 5 1 6 3 7



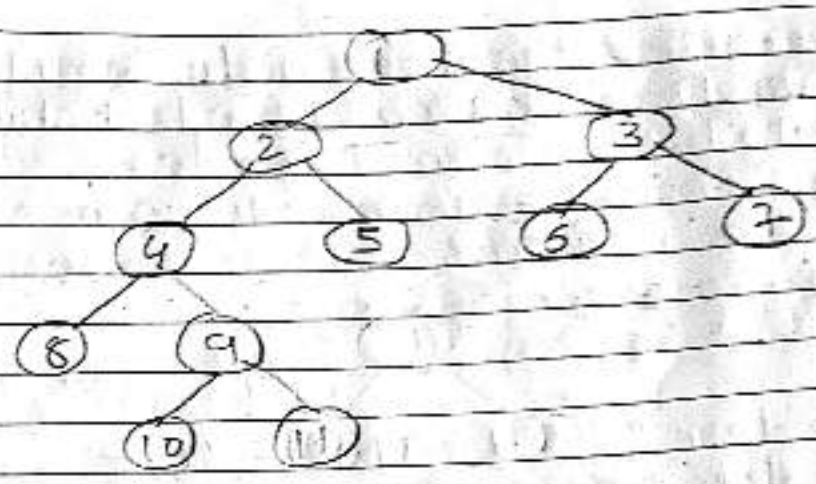
Find the root node.



Find the first left & right child of root



Find the



7

8 4 10 9 11 2 5 1 6 3

7

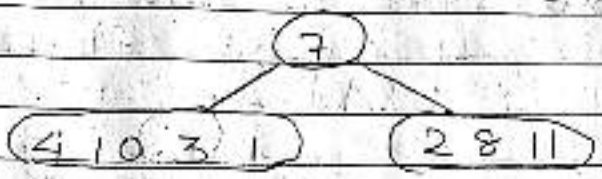
6

8 4 1 10 9 11 2 5

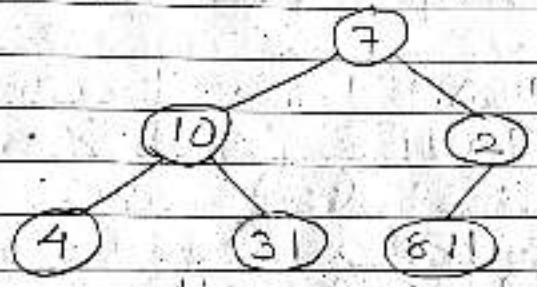
3

Date:
MON TUE WED THU FRI SAT

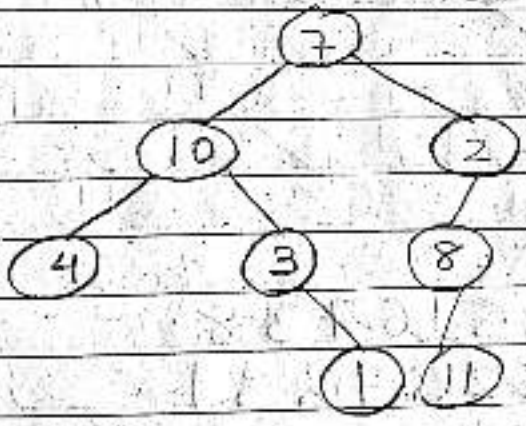
Preorder 7 10 4 3 1 2 8 11
Inorder 4 10 3 1 7 11 8 2



Root node from preorder



Left child & right child of root & then their sucessor

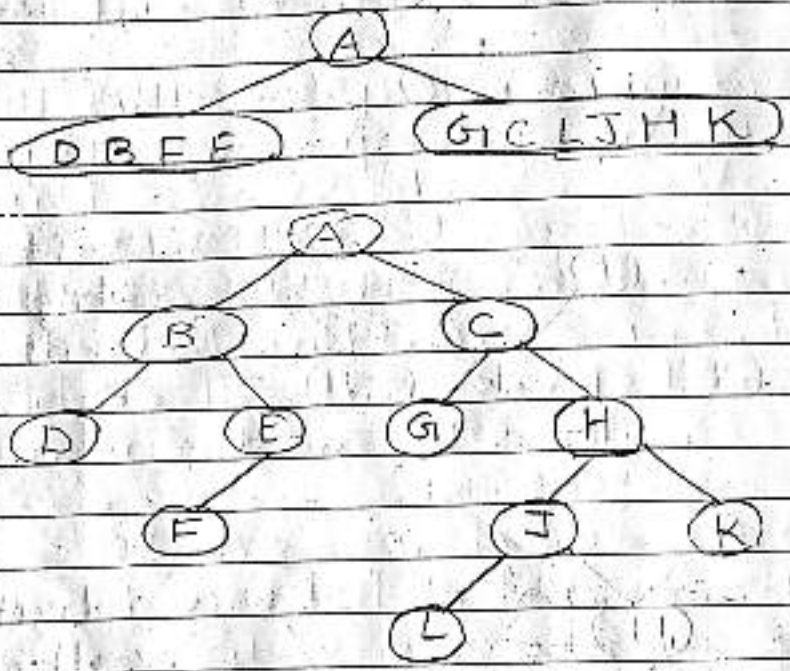


Leaf node of the tree

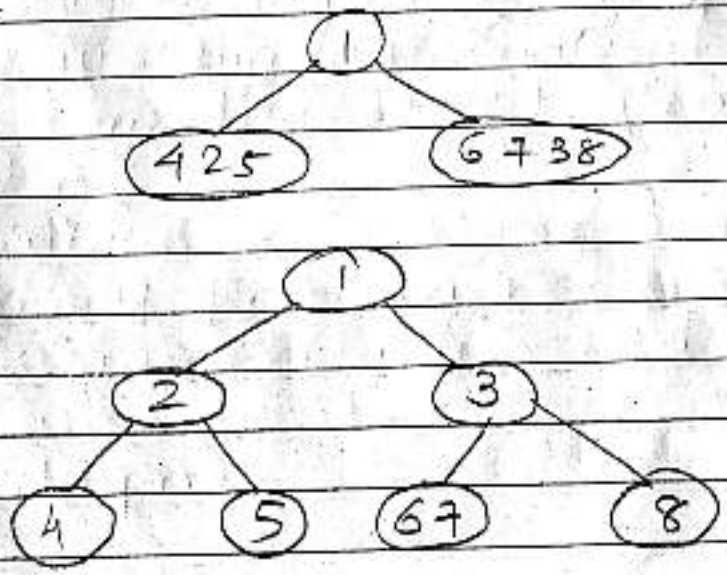
Date:
MON TUE WED THU FRI SAT

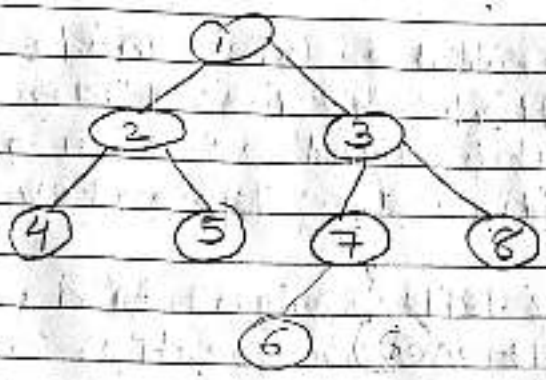
Left right root

Postorder : D F E B G L J K H C A
Inorder : D B F E A G C L J H K

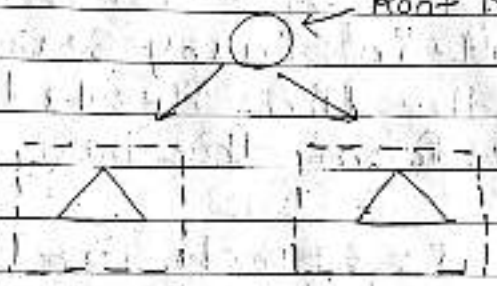


Inorder : 4 2 5 1 6 7 3 8
Postorder : 4 5 2 6 7 8 3 1





BST (Binary Search Tree)



- A BST is also known as ordered binary tree. in which the nodes are arranged in an order. In BST all the nodes in the left subtree have a values less than that of the root node Correspondingly all the nodes in the right of the tree have the value either equal or greater than the root node.

Here the sub tree should satisfy the foll. constraints

Date:
MON TUE WED THU FRI SAT

1. The left Subtree of a node N contains Values that are less than N's Value.
2. The right Subtree of a node N Contains Values that are greater than N's Value
3. Both the left & right binary tree also Satisfy these properties and thus are binary tree.

Advantages

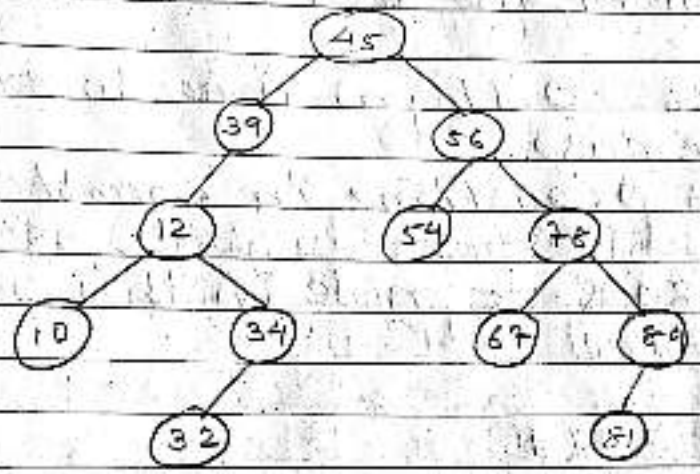
1. Since the node in binary Search tree are ordered so the time needed to search an element from the tree is greatly reduced.
2. whenever we search for an element we do not needed to traverse the entire tree at every node we get a hint regarding which Sub tree to Search in.
2. Average Running time of Search operation is O(log n) at every step that means we eliminate half of the Sub tree from the Search process.

Applications

→ Dictionary problems
→ Time Complexity for Searching an element in BST.
O(log n)

Date:
MON TUE WED THU FRI SAT

Create a BST for the foll. data
45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81



ALGO For Searching an element in BST

1. If tree → data = Val
   or tree = NULL then
   Return tree
   If Val < tree → data
   Return Search element (tree → left, Val)
   else
   Return Search element (tree → right, Val)
2. End

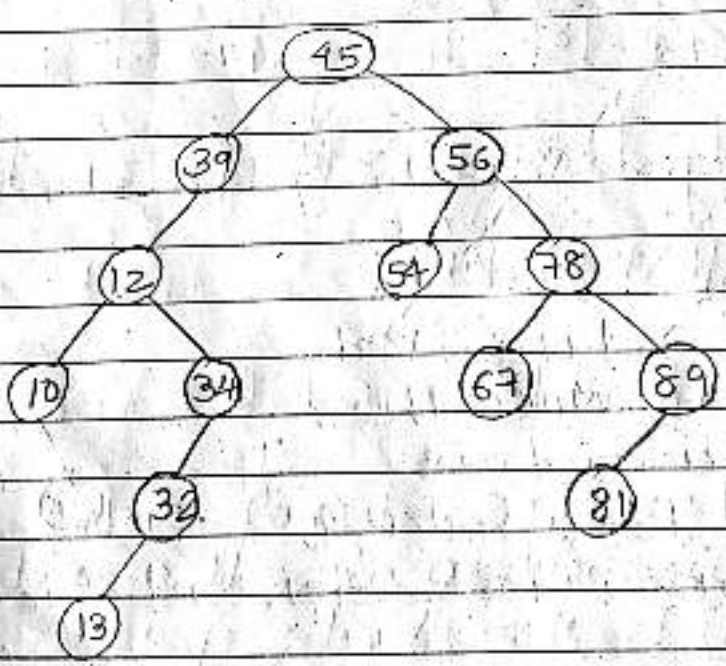
Date:
MON TUE WED THU FRI SAT

Insertion

Insertion required time proportional to hight of the tree in worst case.

1. It takes O (log n) time to execute in average case.
2. It takes O (n) time in worst case.

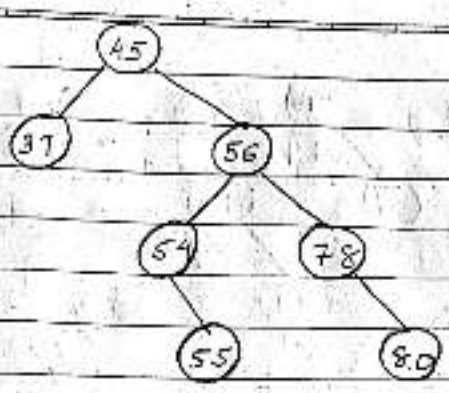
Insert 130 & 55 node in this particular given tree



Deletion

- There are 3 case for Deleting a node from BST (Binary search tree).

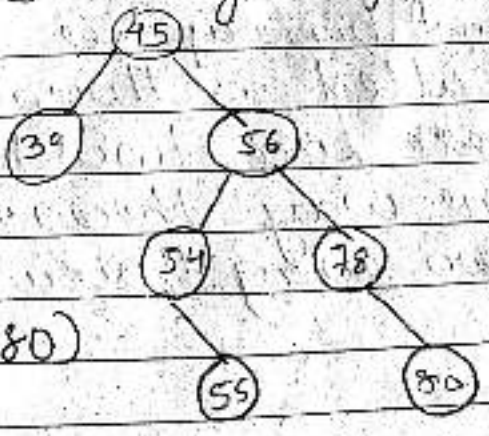
Date:
MON TUE WED THU FRI SAT



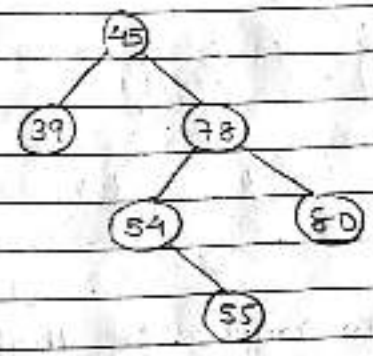
Case: 1 Delete node 55 that means deleting a node that has no child or children

Case: 2 Deleting a node with one child
- To handle this case the node child's is said to be the child of node's parent in other words replace the node with it's child.
- Now, if the node was the left child of it's parent the node's child becomes the left child of node's parent.

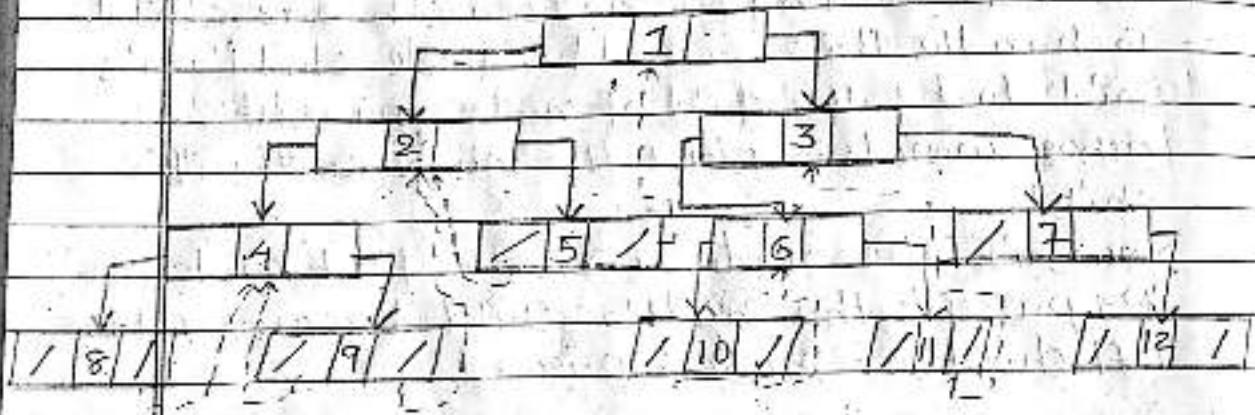
Case: 3 Deleting a node with two children
- To handle this case replace the node's value with it's inorder predessor or inorder sucessor.
- The in-order predecessor or the sucessor can then be deleted using any of the above cases.



Inorder:
→ (39, 45, 54, 55, 56, 78, 80)



Threaded Binary Tree



8 4 9 2 5 1 10 6 11 3 7 12

- Null entries can be replaced to store a pointer to the in-order successor or the in-order predecessor of the tree.
- This special pointers is called as threads and binary tree containing threads are called as threaded binary tree.
- There are two types of TBT:
  i) One way threading
  ii) Two way threading / Double / full

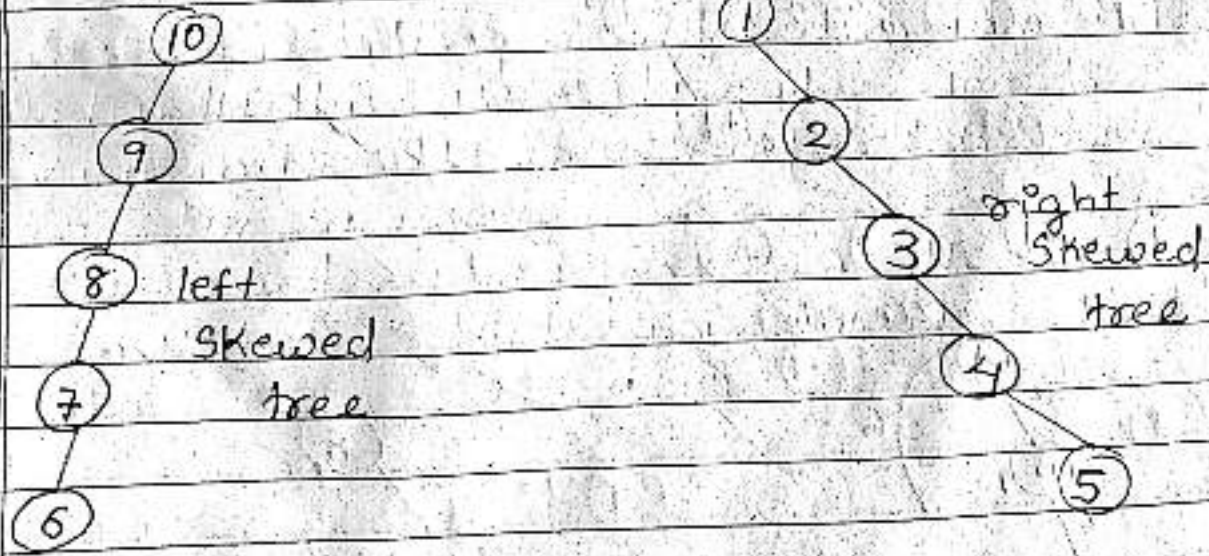
Date:
MON TUE WED THU FRI SAT

i) ⇒ If the thread appears in the left field then left field will be made to point to the in-order predecessor of that node such a one way threaded tree is called as left threaded binary tree

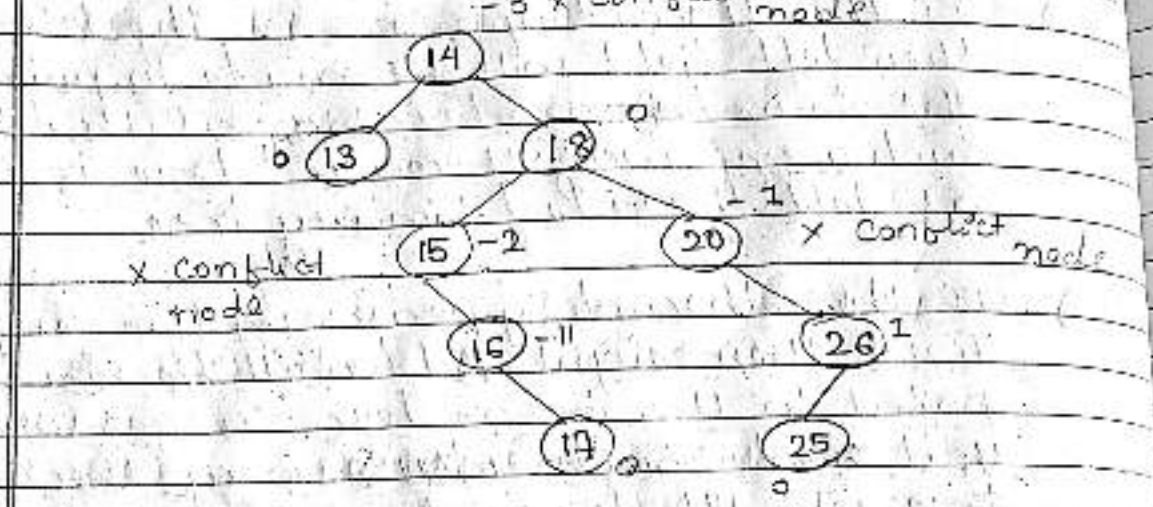
ii) ⇒ If the thread appears in the right field then right field will be made to point to the in-order successor of that node such a one way threaded tree is called as right TBT.

iii) ⇒ In a two way threaded tree thread will appear in both the left & right field of the node

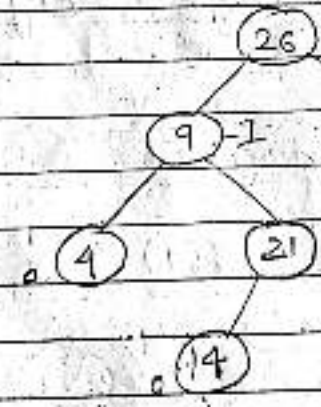
AVL Tree



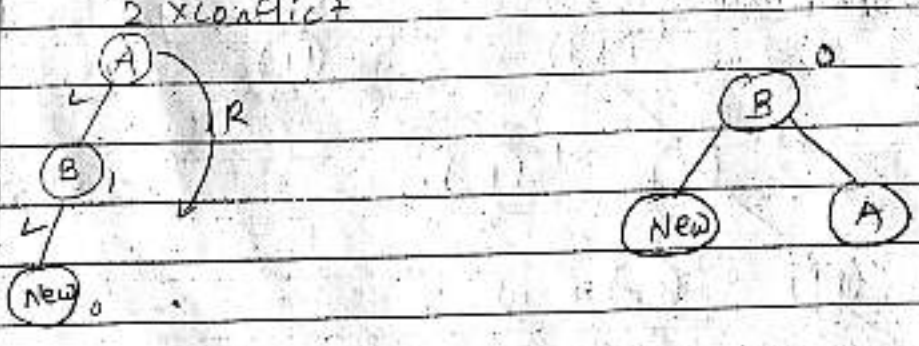
Date:
MON TUE WED THU FRI SAT



Balance factor = height of LS - height of RS



i) LL



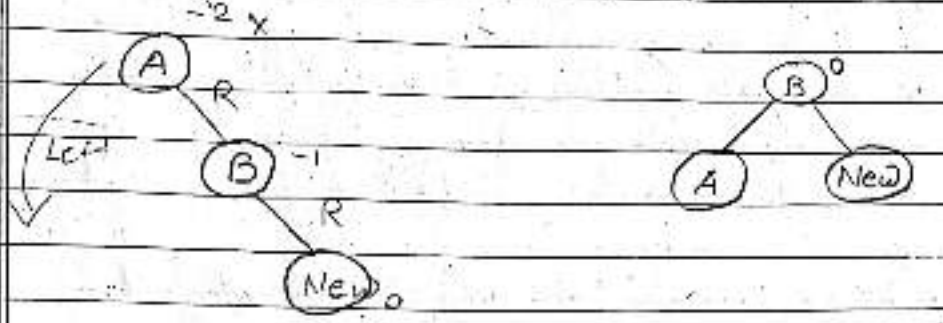
ii) RR

iii) LR

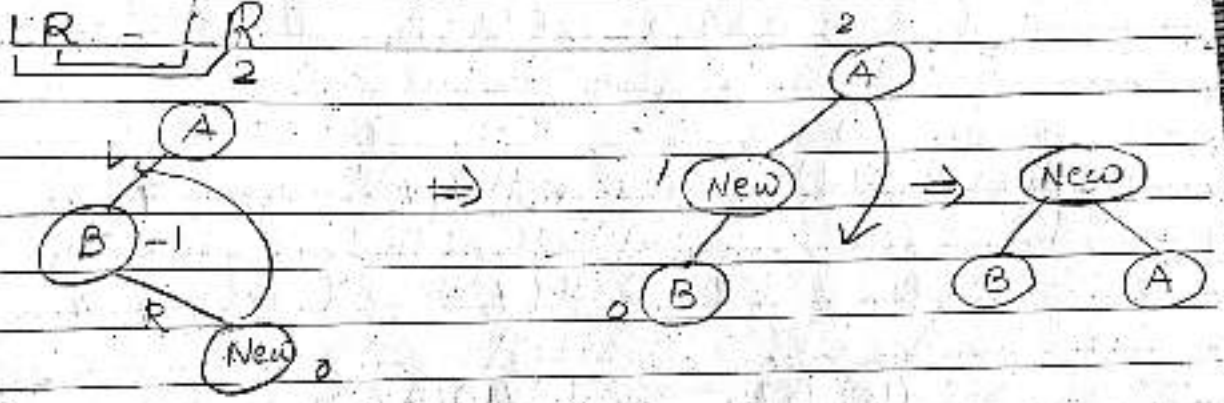
iv)

Date:
MON TUE WED THU FRI SAT

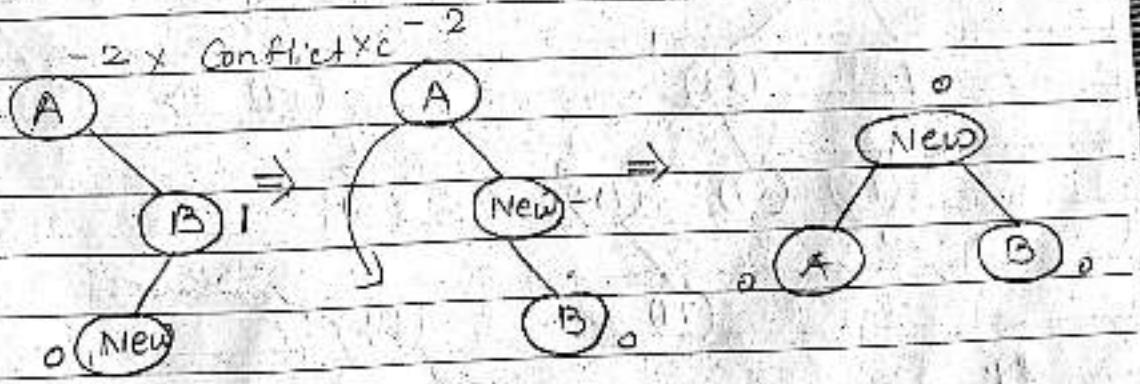
ii) RR



iii) LR - LR

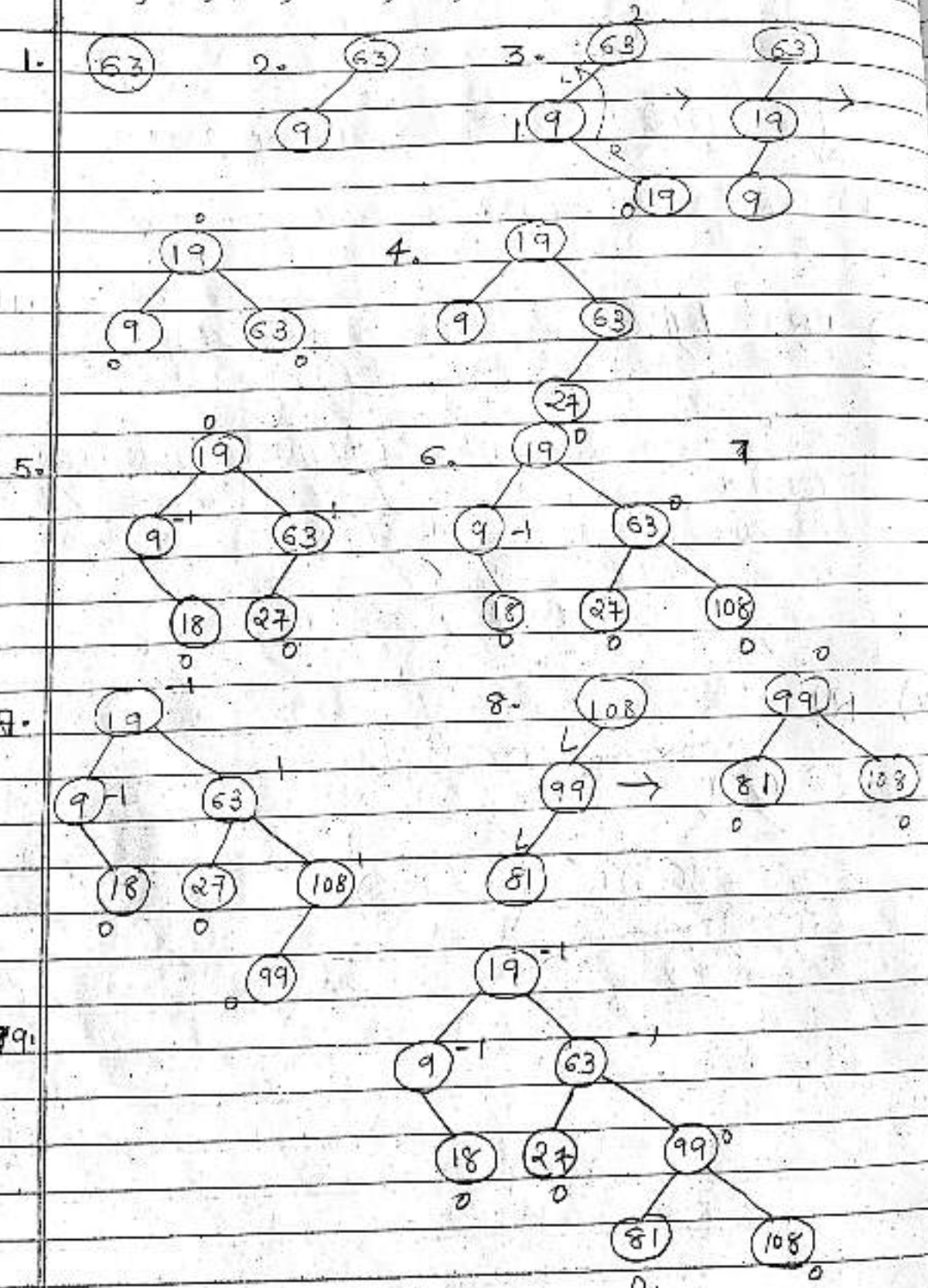


iv) RL



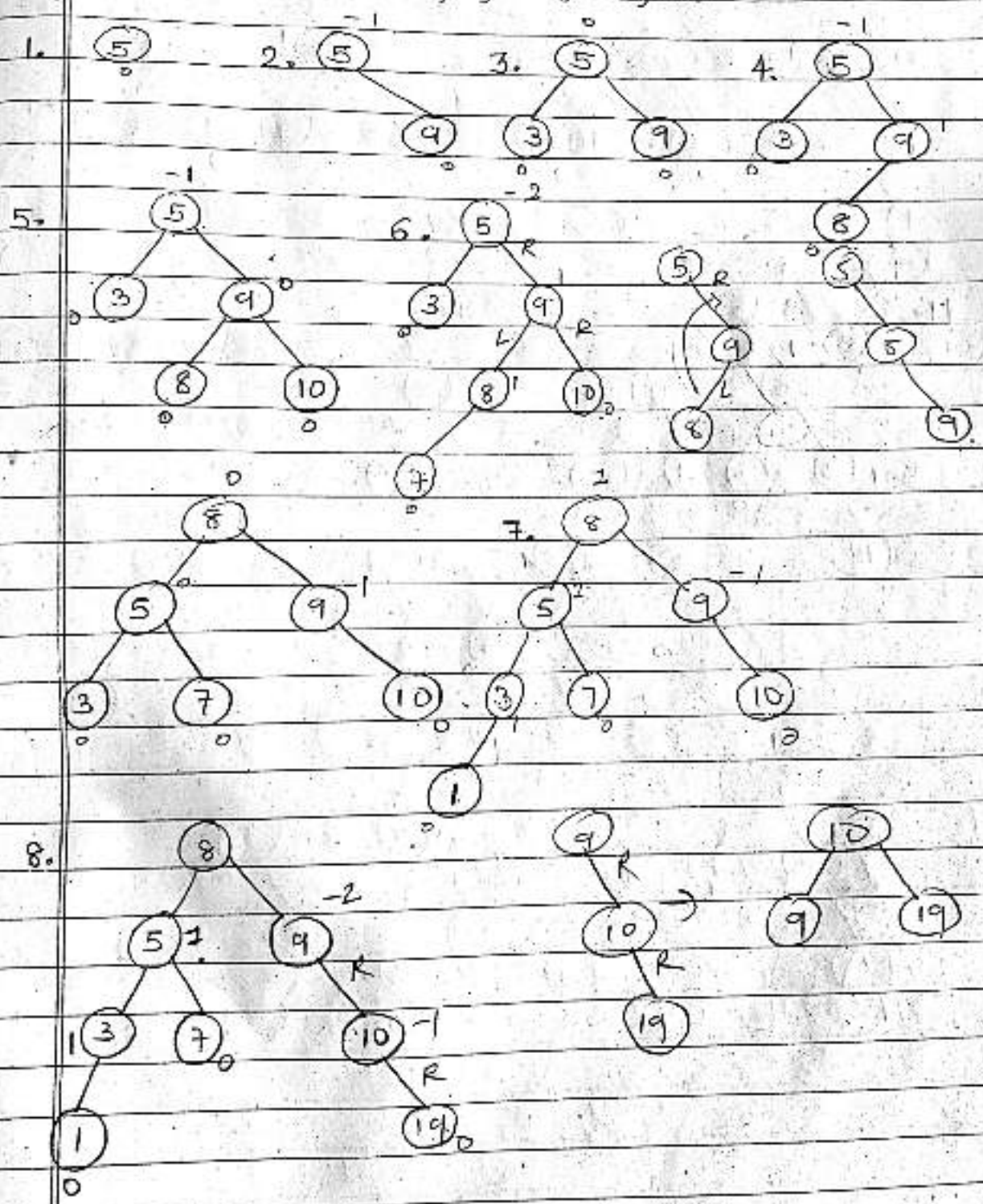
Date:: MON TUE WED THU FRI SAT

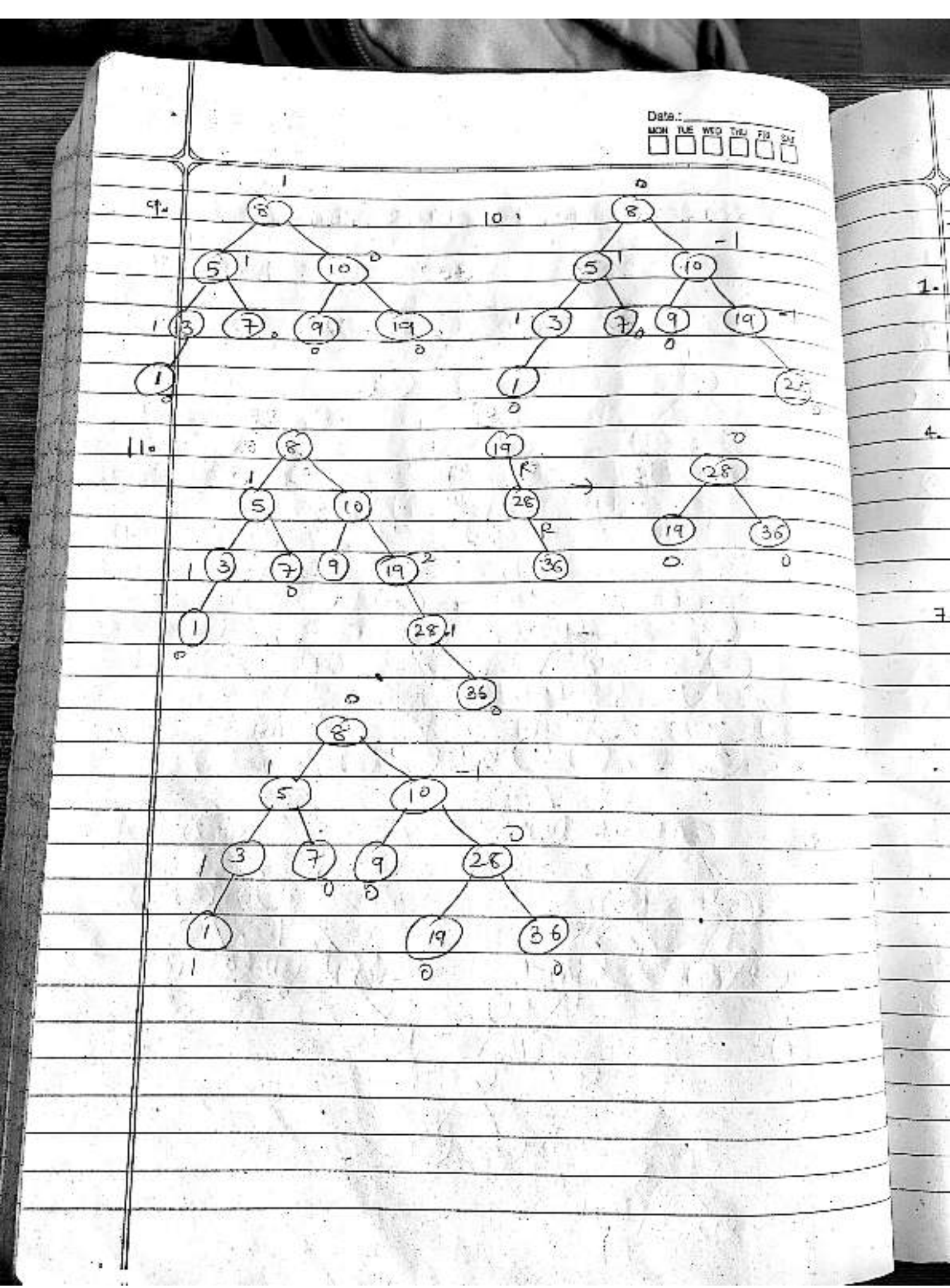
63, 9, 19, 27, 18, 108, 99, 81



Date:
MON TUE WED THU FRI SAT

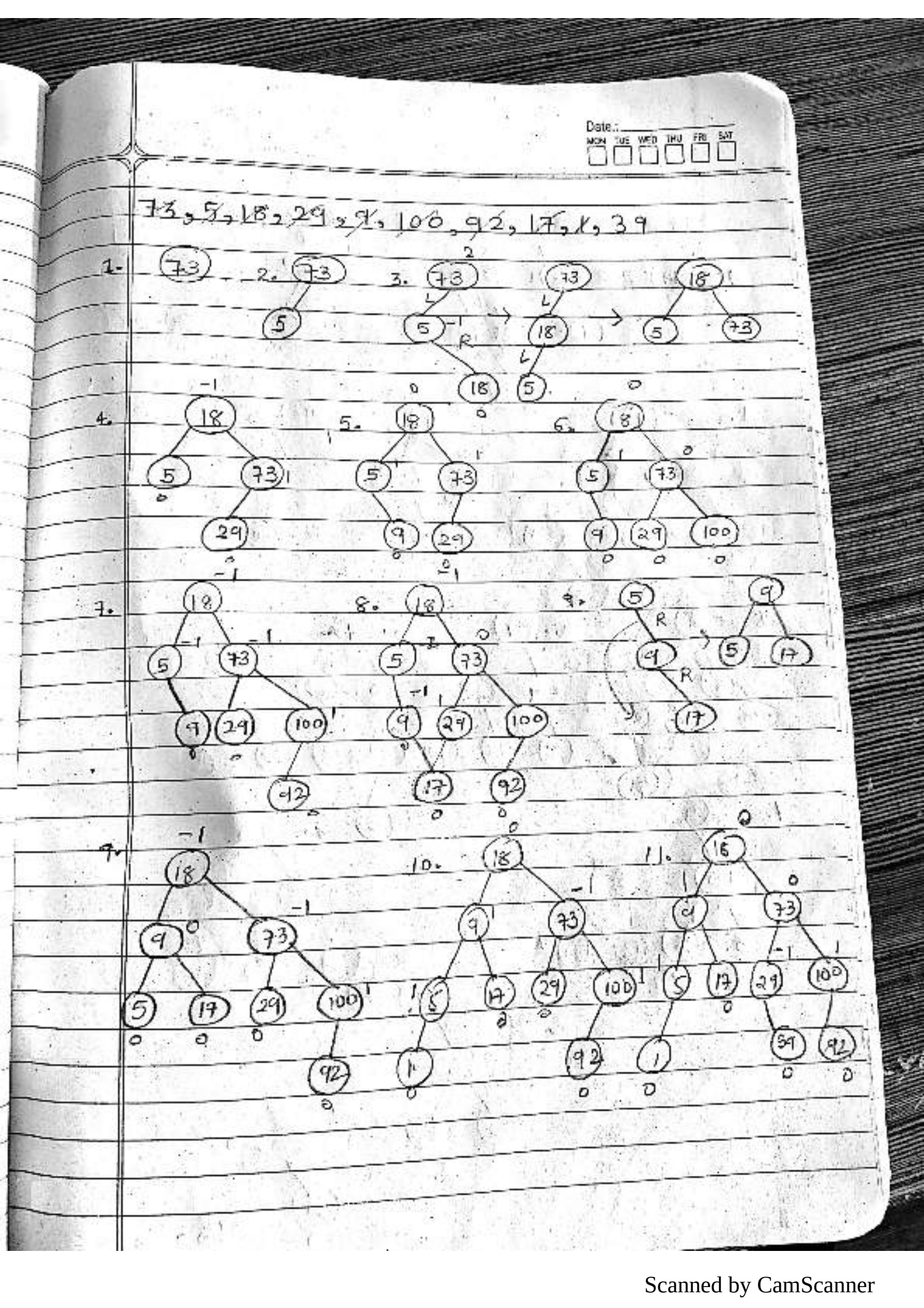
5, 9, 3, 8, 10, 7, 1, 19, 28, 36





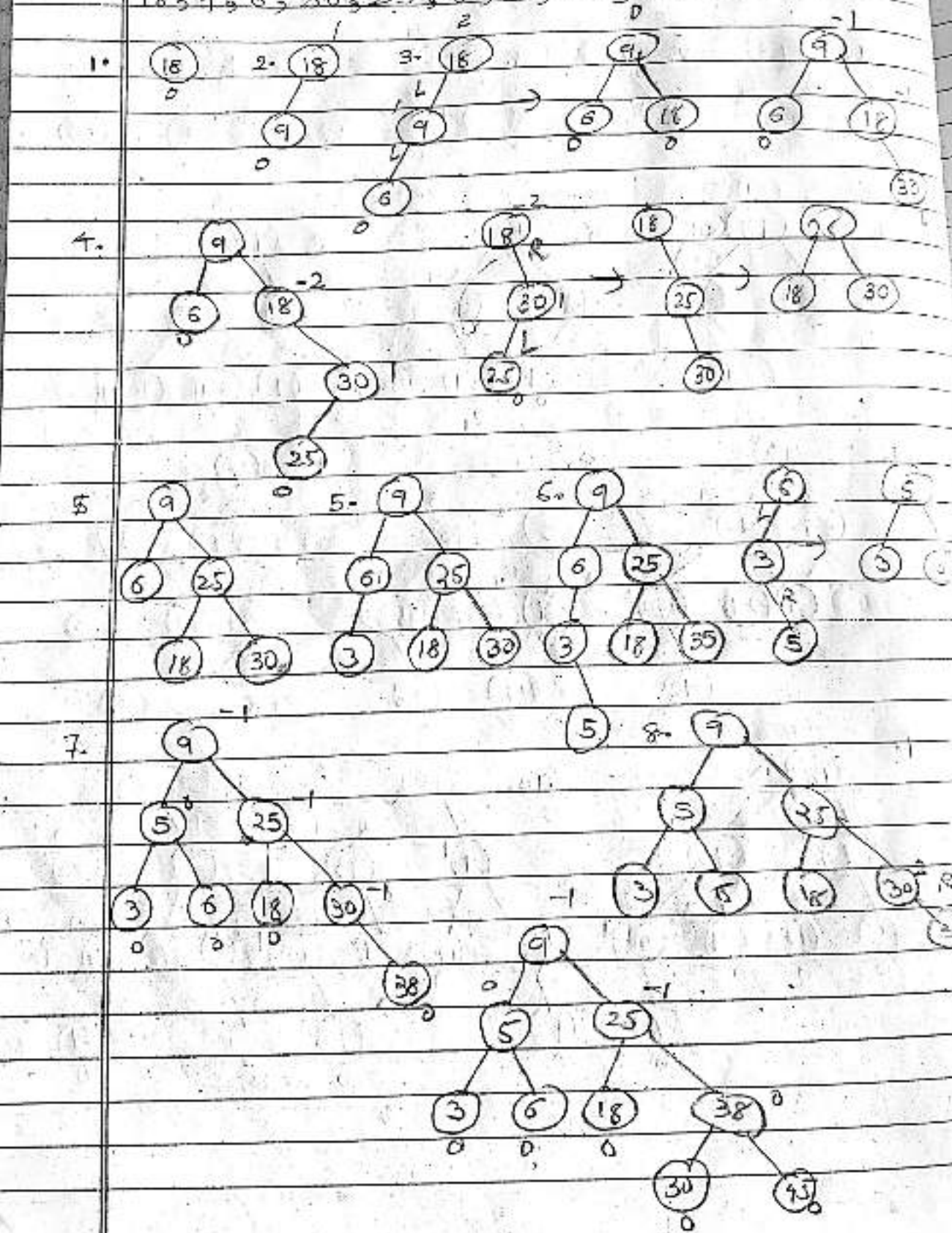
Date:
MON TUE WED THU FRI SAT

73, 5, 18, 29, 9, 100, 92, 17, 1, 39



Date:
MON TUE WED THU FR SU

18, 9, 6, 30, 25, 3, 5, 38, 45



Date:
MON TUE WED THU FRI SAT

Delete (AVL)

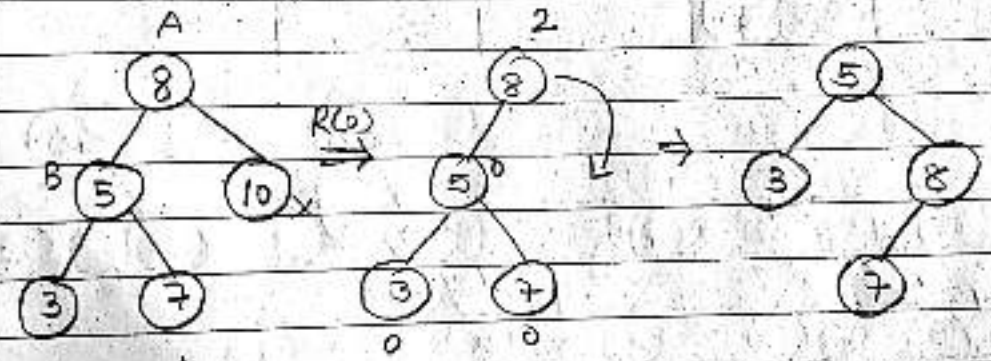
Type (R)
(a) R(0)
(b) R(1)
(c) R(-1)

Type (L)
(a) L(0)
(b) L(1)
(c) L(-1)

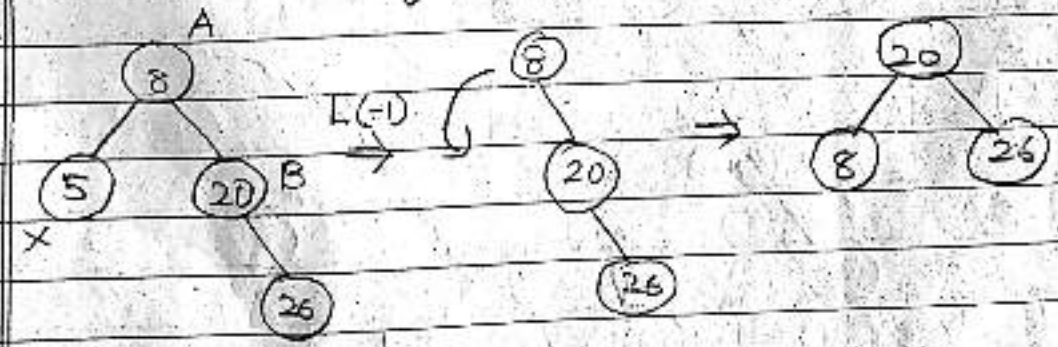
B = Sibling of X

R			L		
B.F of B	Rotation	Similar Rotation	B.F of B	Rotation	Similar Rotation
0	R(0)	LL	0	L(0)	RR
1	R(1)	LL	1	L(1)	RL
-1	R(-1)	LR	-1	L(-1)	RR

1.

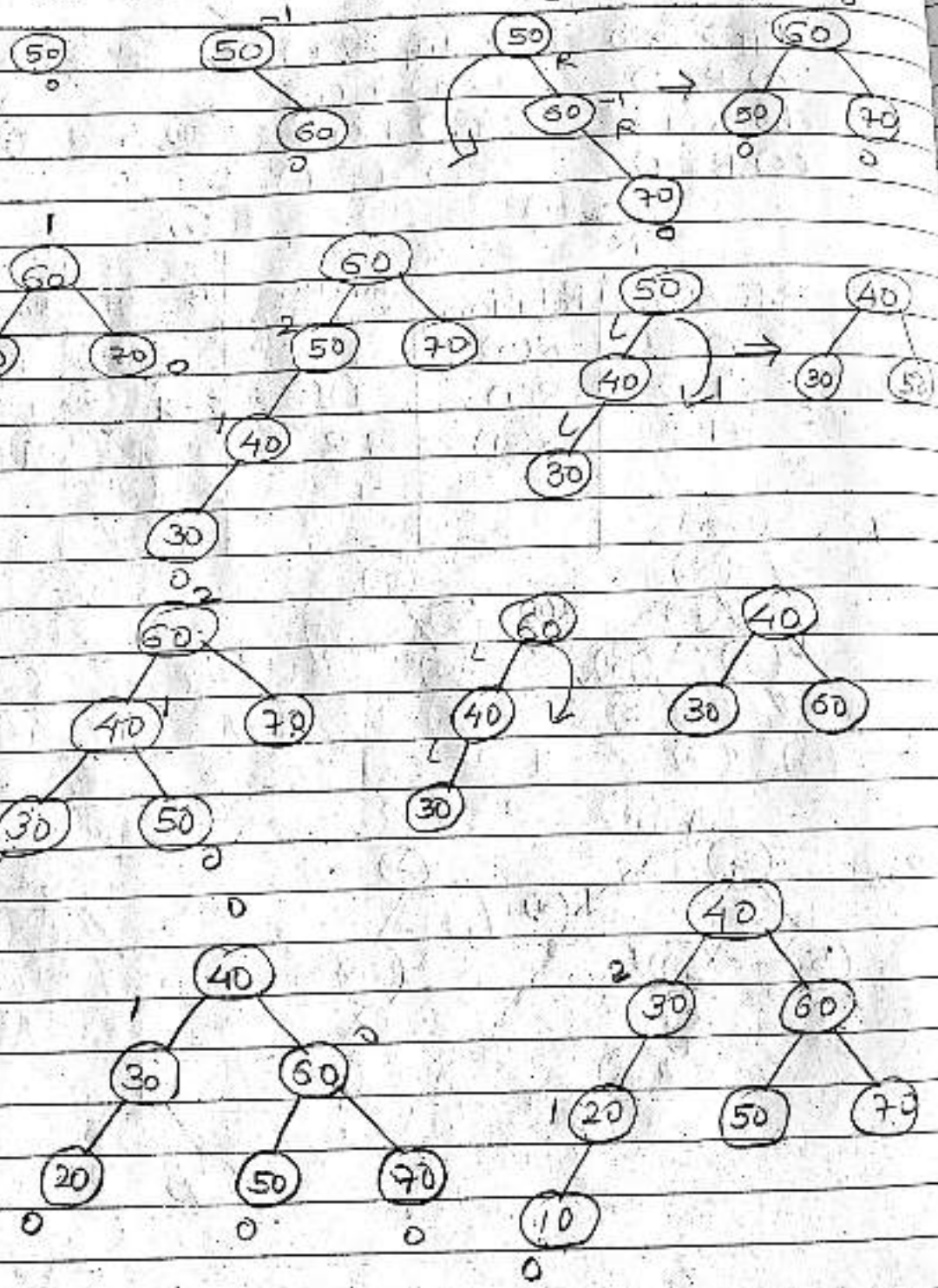


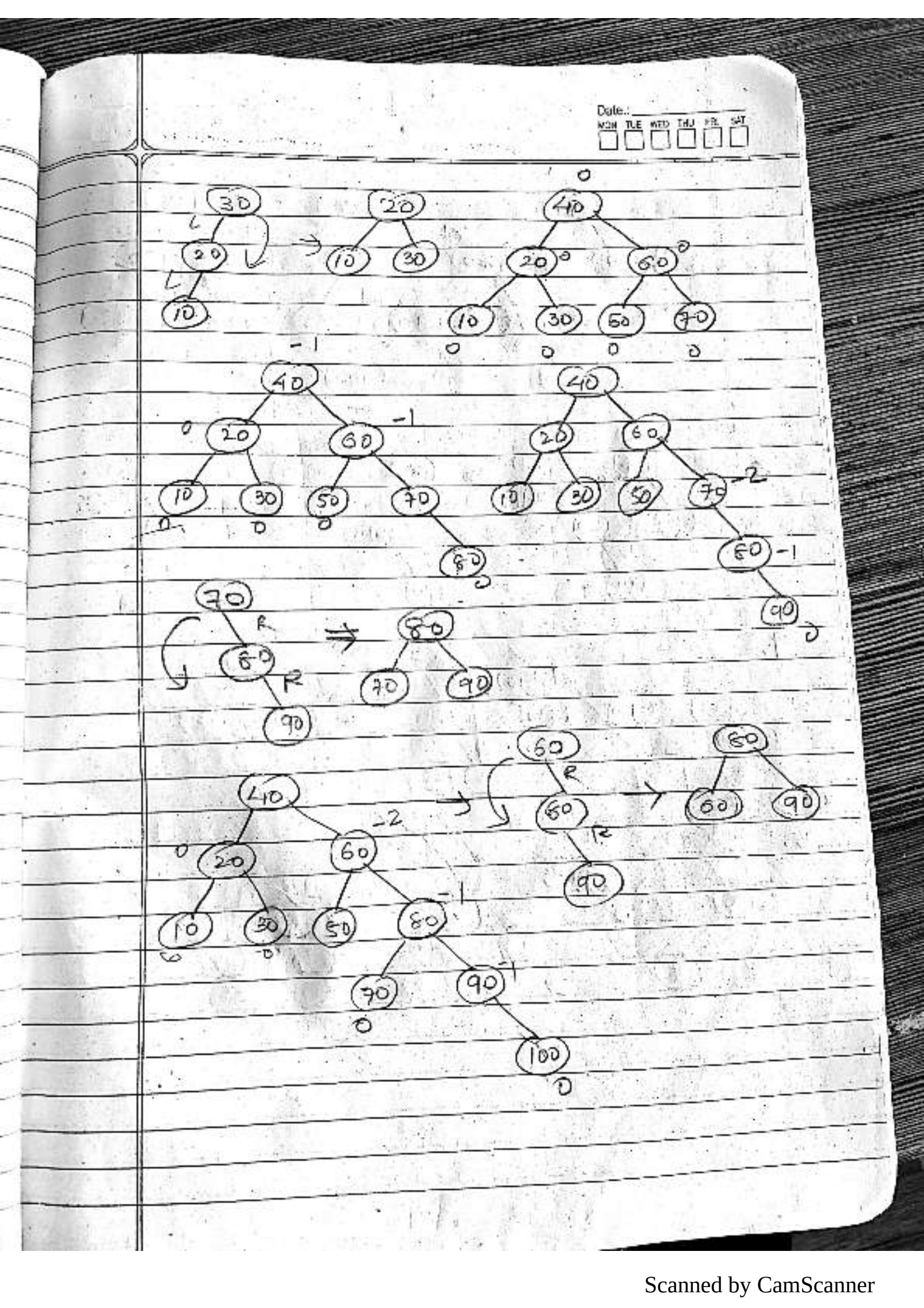
2.

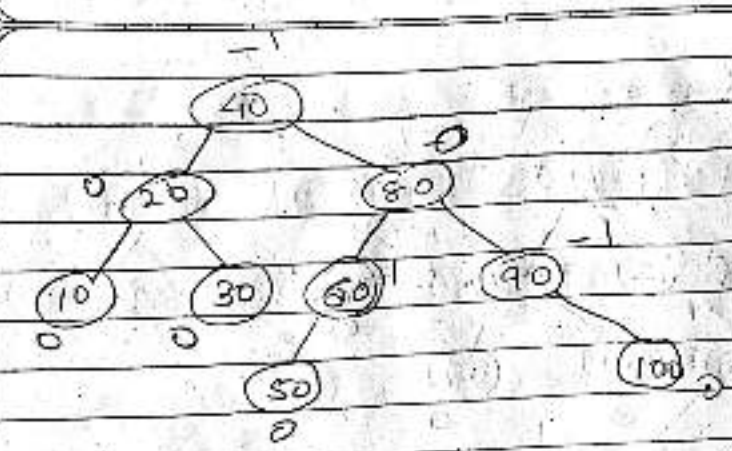


Date:
MON TUE WED THU FRI SAT

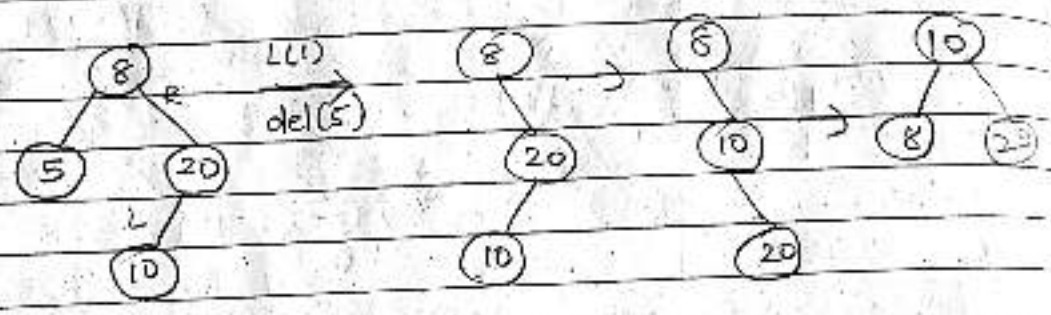
50, 60, 70, 40, 30, 20, 10, 80, 90, 100



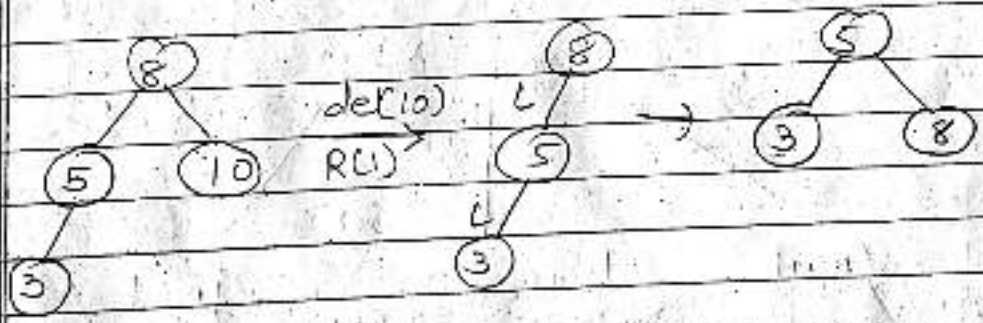




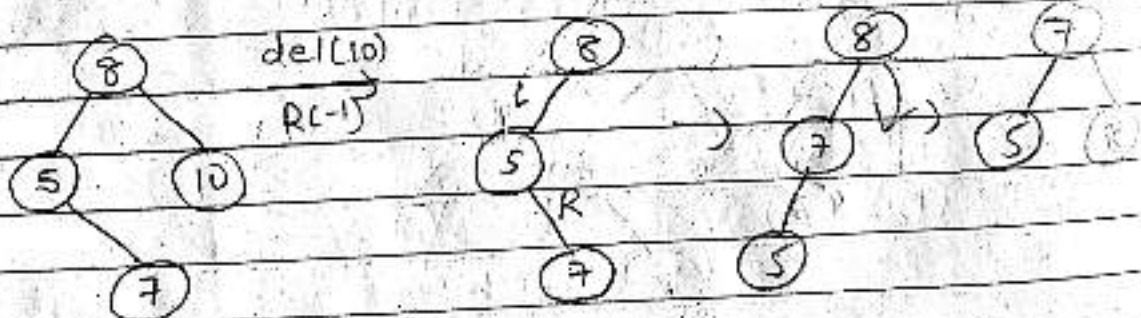
3.



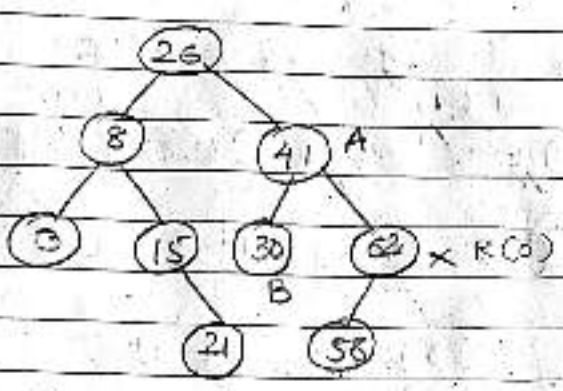
4.



5.



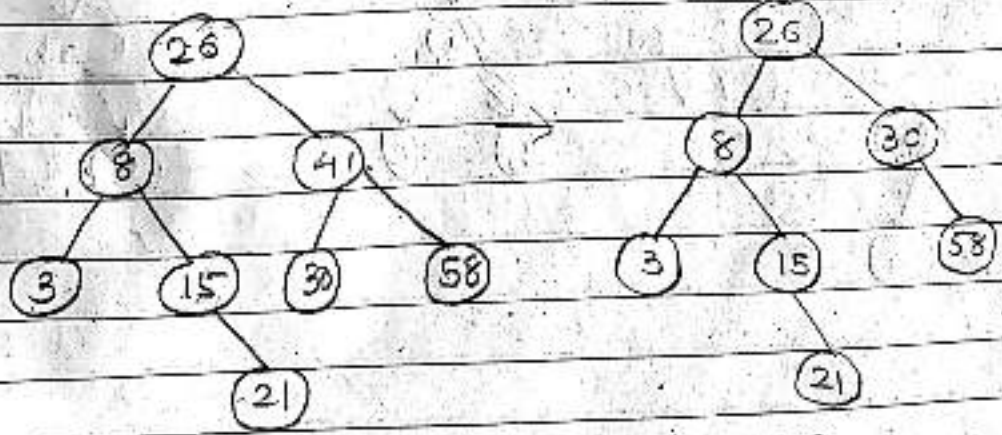
Date:
MON TUE WED THU FRI SAT



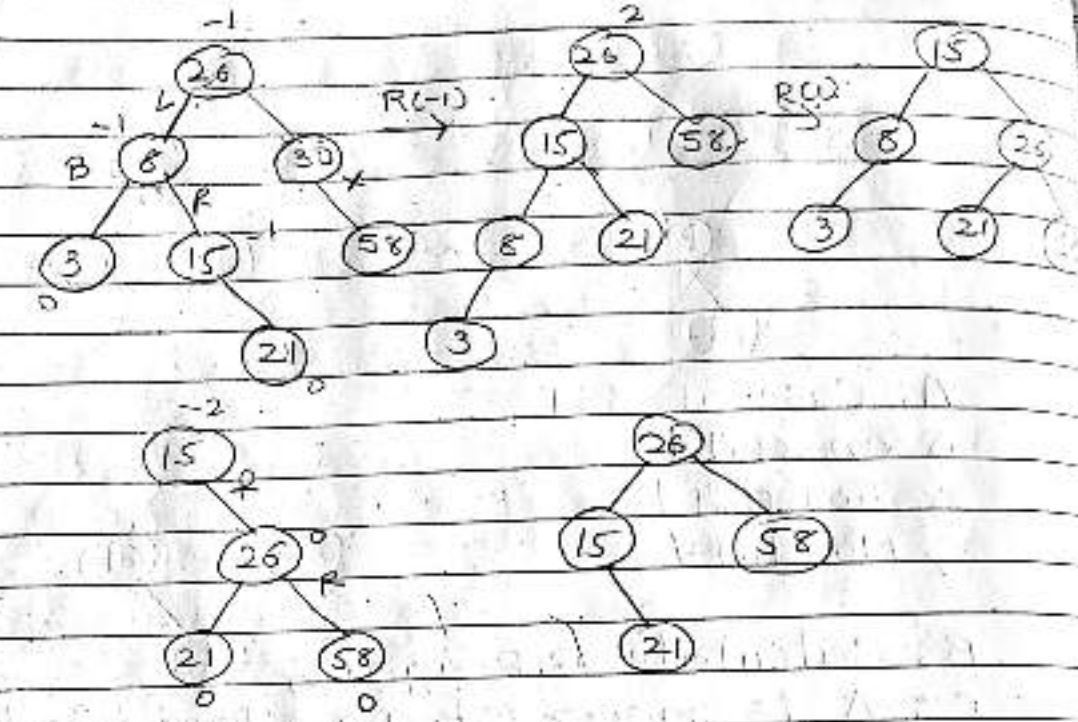
A: Case of BST
1) one child
2) Two child
3) No child

B: Calculate B.F again
C: A is nearest ancestor of deleted node.
D: If deleted node are from left sub tree of A then it is called Type L. Delete. otherwise it is called Type R Delete.

62, 41, 30, 3, 8

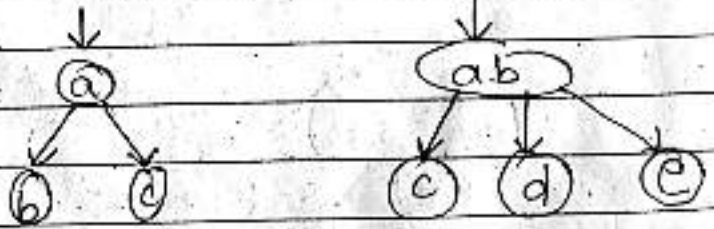


3, 8, 15, 21, 26, 30, 41, 58



2-3 Tree

- leaf node
- one node two childs
- two node three child

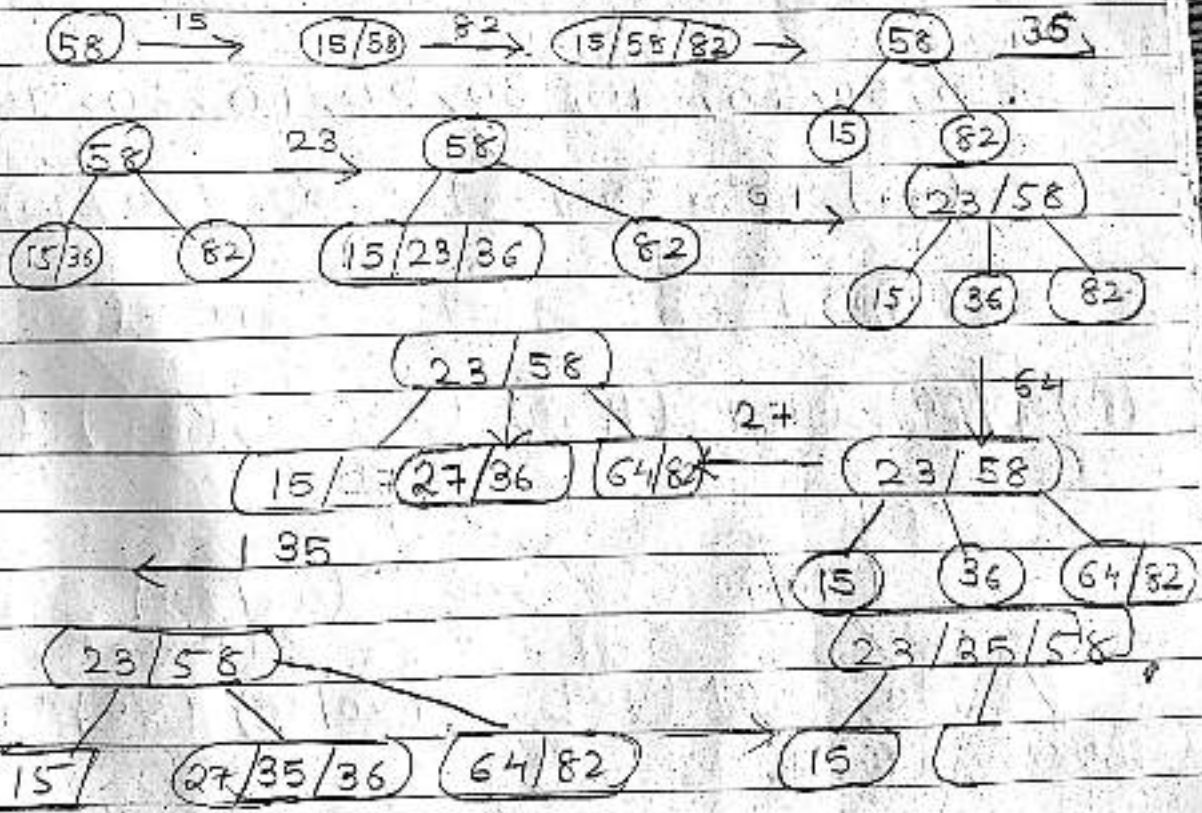


Date:
MON TUE WED THU FRI SAT

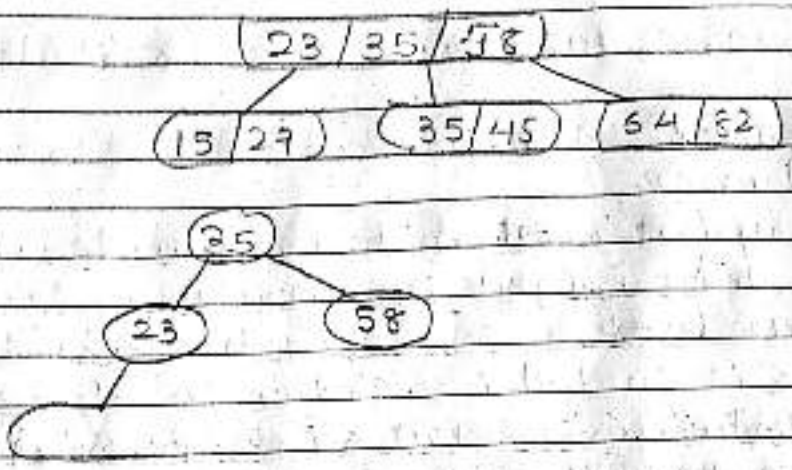
50, 30, 10, 70, 60

Property
1. Balance
2. Every internal node is 2 node or 3 node so tree might be shorter
3. Same level
4. Data is sorted in order to store.
5. Maintaining of 2-3 Tree is simple than maintaining a BST.

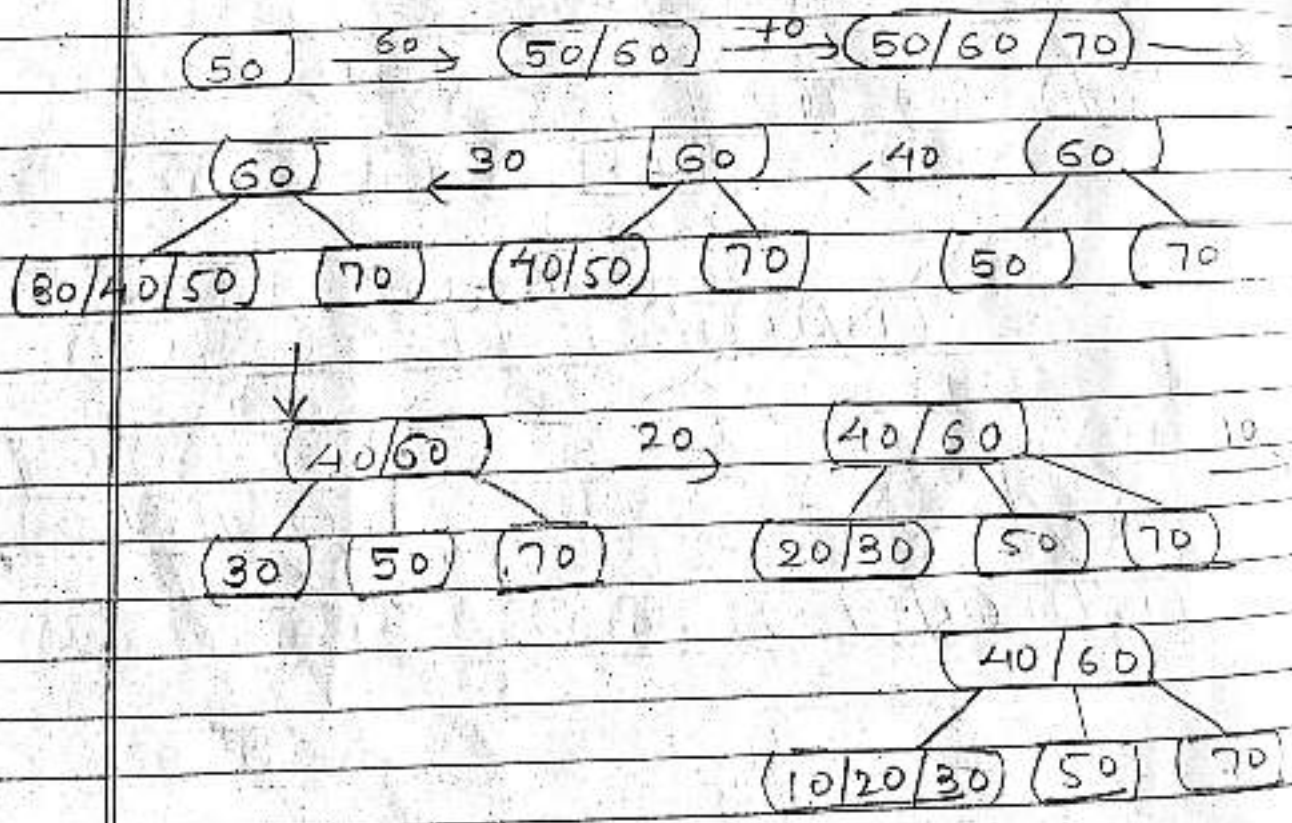
58, 15, 82, 36, 23, 64, 27, 35, 45

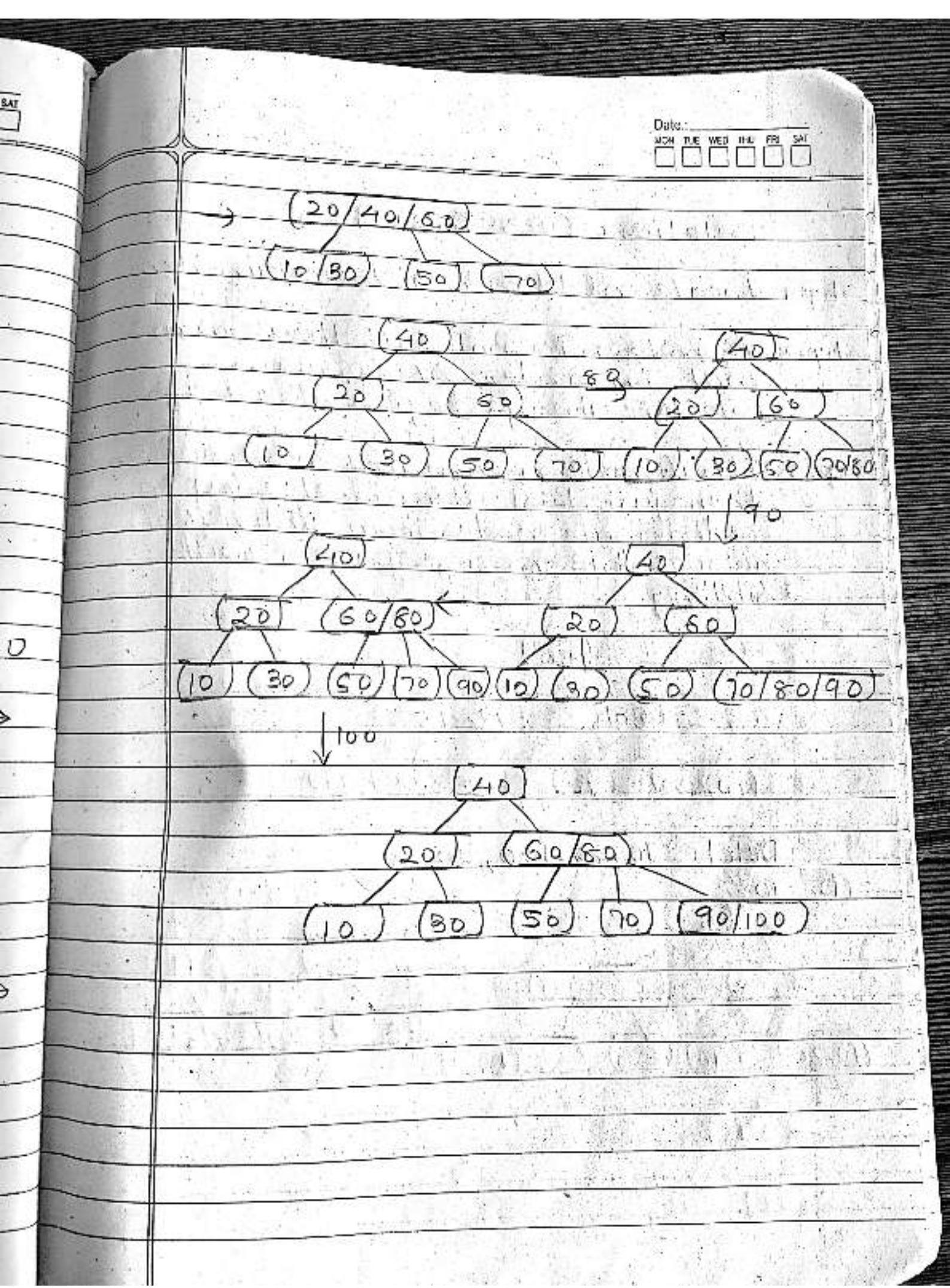


Date.:
MON TUE WED THU FRI SAT



50, 60, 70, 40, 30, 20, 10, 80, 90, 10





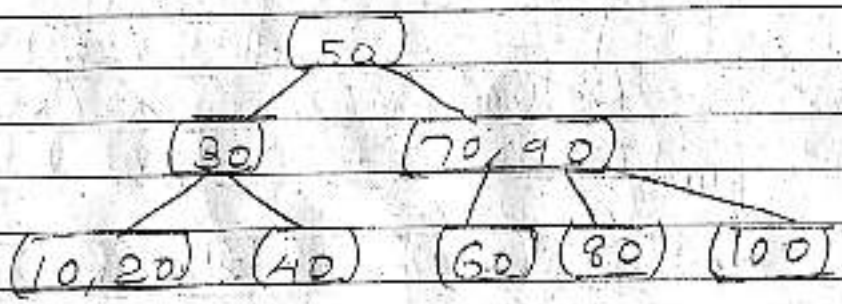
Date:
MON TUE WED THU FRI SAT

Deletion (2-3 Tree)

Step-1 Locate node n which contain item I

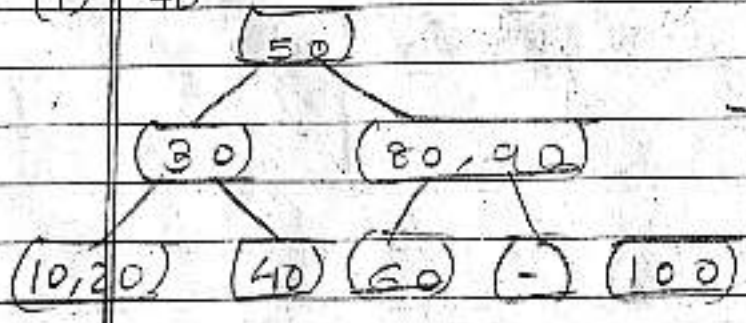
Step-2 If node n is not leaf then Swap I with in order successor. Deletion always begins at a leaf.

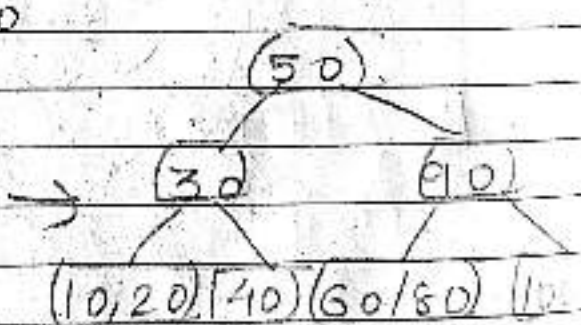
step-3 If leaf node n contains another item then just delete item I. else try to redistribute nodes from siblings. if not possible merge parent with sibling.



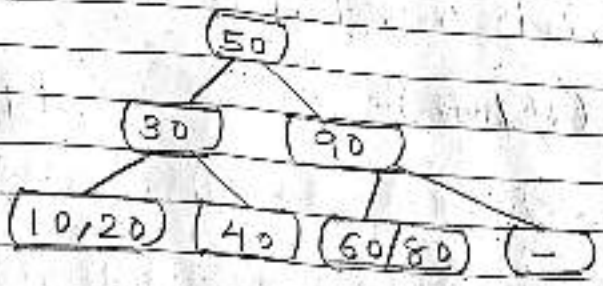
Delete 70, 80, 100, 80

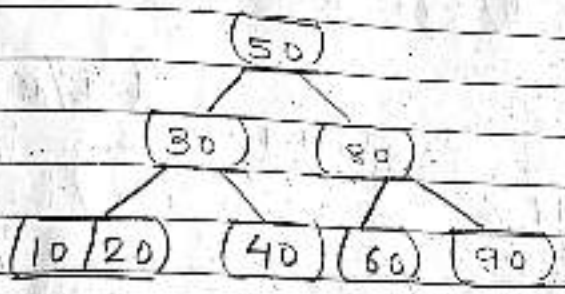
(1) 70



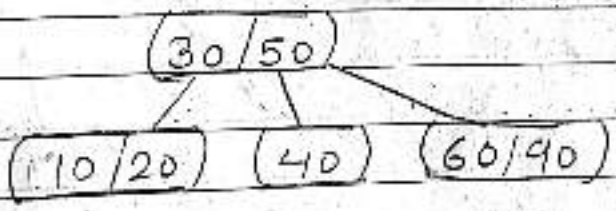


(ii) 100



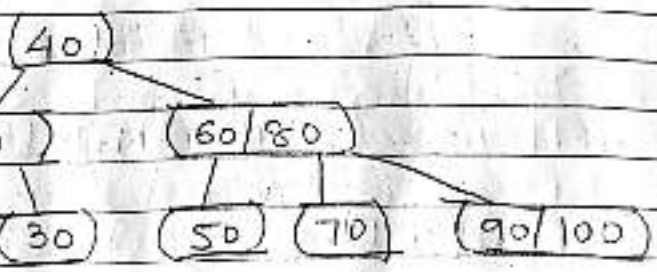


(iii) 80

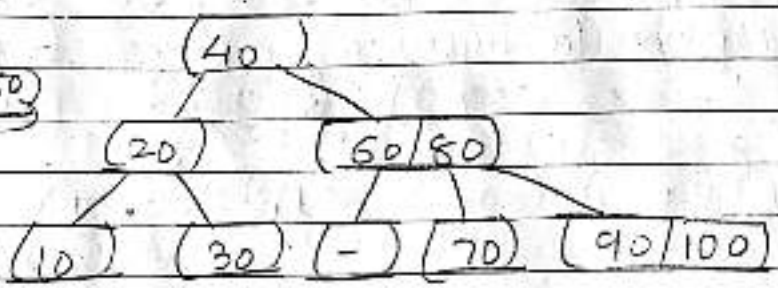


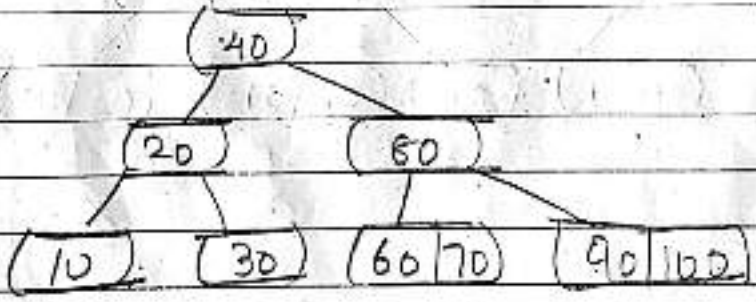
Date:
MON TUE WED THU FRI SAT

Delete 50, 20, 10.

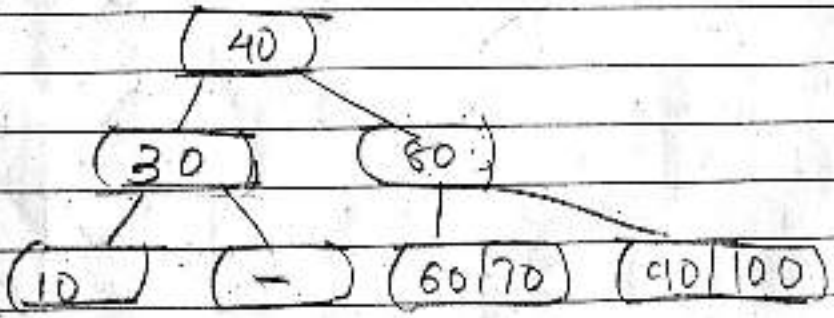


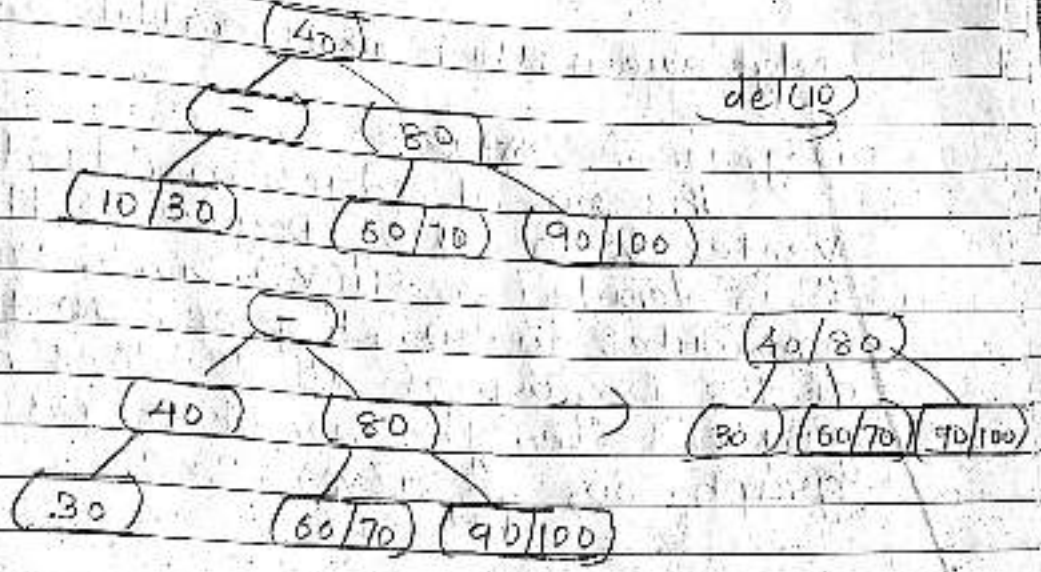
del(50)





del(20)





Graph

Adjacent Vertex

If two vertex join by same edge then it is called adjacent vertex.

Adjacent Edges

If two edges are incident on same vertex then it is called as adjacent edge.

Self loop

Edge having same vertex (V_i, V_j) as it's end vertices then it is called self loop.

Parallel loop

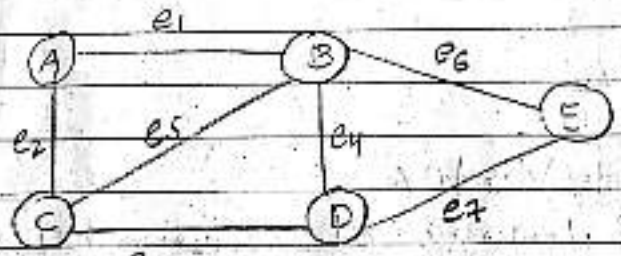
When more than one edge associated with a given pair of vertices such

Date:
MON TUE WED THU FRI SAT

edge are called as parallel edge.

Degree of Vertex

The no. of edges incident on a vertex V_i is called Degree of that vertex. it is denoted as $D(V_i)$. Since, each edge contributes to two degrees so the sum of the Degree of all vertices in a graph is two the no. of edges of that graph. So, $\sum_{i=1}^{n} d(v_i) = 2 \times e$



	Self loop	Parallel edge
Simple g.	X	X
Multi g.	X	✓
Pseudo g.	✓	X
graph	✓	✓

Date:
MON TUE WED THU FRI SAT

Finite graph

A graph with finite no. of vertices as well as finite no of edges is called as finite graph.

Null Graph

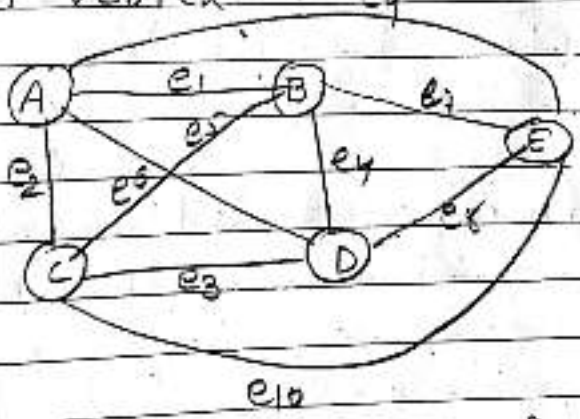
A graph having no edge is called as Null graph.

Trivial Graph

A graph with only vertex is called Trivial Graph

Complete Graph

There is an edge between every pair of Vertex



The maximum number of edges in Simple graph with vertex n is $\frac{n \times n-1}{2}$

For undirected graph.

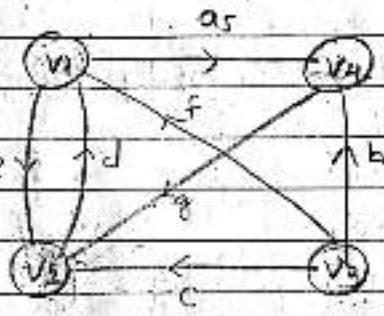
For directed graph $n \times (n-1)$.

Date:
MON TUE WED THU FRI SAT

Matrix Representation of Graph

i) Sequential Representation
By using adjacency matrix

ii) Link Representation
By using an adjacency list that stores the neighbours of a node using a link list.



A^2 =

	V_1	V_2	V_3	V_4
V_1	1	1	0	0
V_2	0	1	0	1
V_3	1	2	0	1
V_4	1	0	0	0

A^3 =

	V_1	V_2	V_3	V_4
V_1	1	1	0	1
V_2	1	1	0	0
V_3	2	2	0	1
V_4	0	1	0	1

A^4 =

	V_1	V_2	V_3	V_4
V_1	1	2	0	1
V_2	1	1	0	1
V_3	2	3	0	2
V_4	1	1	0	0

Path Matrix

Path matrix is used to show whether to exist a simple path node V_i to V_j or not.

Date:
MON TUE WED THU FRI SAT

Path Matrix

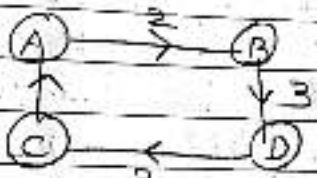
$B^2 = A^1 + A^2 + A^3 + \cdots + B^\sigma$

$B^4 = \begin{bmatrix} 3 & 6 & 6 & 5 \\ 3 & 5 & 6 & 7 \\ 2 & 3 & 3 & 5 \\ 6 & 8 & 7 & 9 \end{bmatrix}$

$V_{ij} = 0$ if the path is not exist

$V_{ij} = 1$ if the path is exist

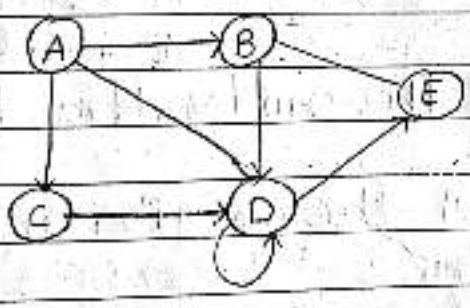
Weighted Matrix



Edge containing weight

A^3	A	B	C	D
A	0	0	8	
B				
C				
D				

Adjacency



A → B → C /

B → D → E /

C → D /

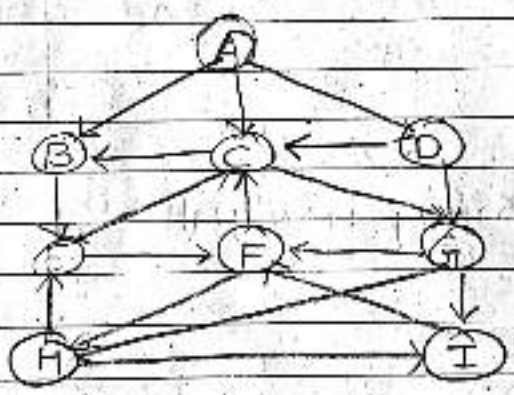
D → A → D → E /

E /

Date:
MON TUE WED THU FRI SAT

i. BREADTH FIRST SEARCH
ii. DEPTH FIRST SEARCH

i⇒ Uses a queue data structure. In BFS begins with any node (as a starting node) and explore all the neighbouring nodes then for each of those nearest node the algorithm explore their unexplored neighbouring nodes and so on until it finds the goal.



A - B C D
B - E
C - B G
D - C G
E - F C
F - C H
G - F H I
H - F I E
I - F

Q = Used to hold the nodes that have been processed
O = keep track of the origin of each edge.

Q = A
O = \0

Q = A B C D
O = \0 A A A

ii⇒ D

Q = A B C D E Gr E (H) I → Destination
\0 A A A B C F Gr Gr'

[A - C - Gr - H] → Path from source to dest.

ii ⇒ DFS uses a stack data structure. The DFS algorithm progresses by expanding the starting node of Graph Gr & thus going deeper & deeper until a gold node is found. when a dead end is reached the algorithm back trace returning to most recent node that has not been completely explored.

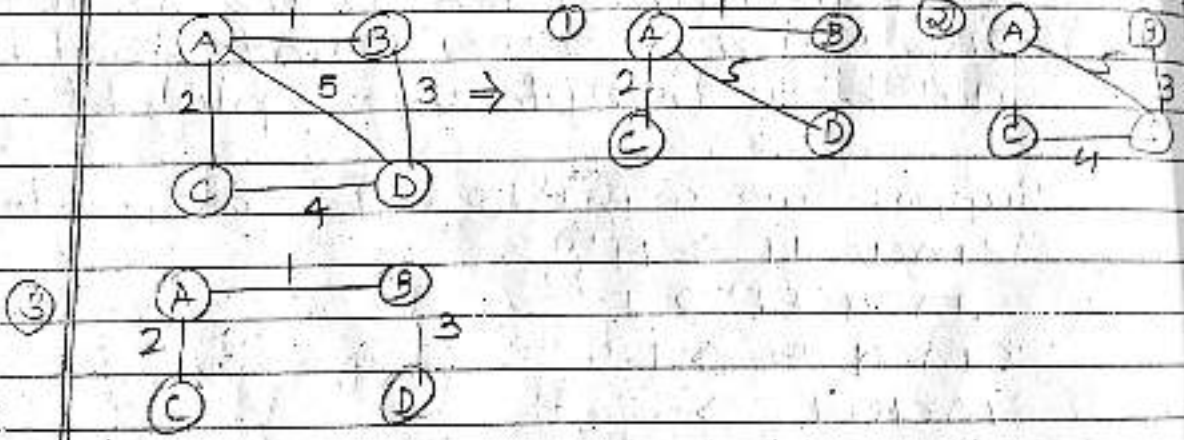
Suppose we want to search from node H

STACK H → H
STACK EI → I
STACK EF → F
STACK EC → C
STACK EBG → G
STACK EB → B
STACK E → E

Minimum Spanning Tree (MST)

1. Prim's Algo
2. Kruskal's Algo

- Minimum spanning Tree is define for weighted graph.
- Spanning tree of a graph G consist of all vertices & some of the edges, so that the graph doesn't contain a cycle.

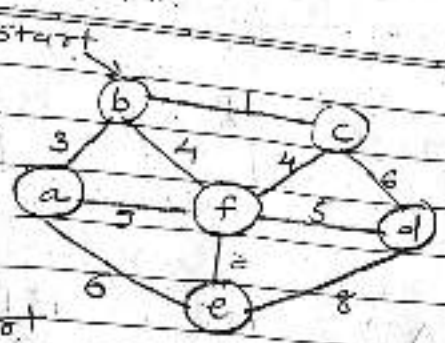


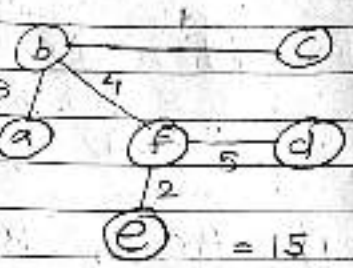
i) Prim's Alg.

1. Select one any connected vertices with minimum weight.
2. Select unvisited vertex which is adjacent of visited vertices with minimum weight
3. Repeat Step-2 until all vertices are visited
4. Uses Vertex,

ii)

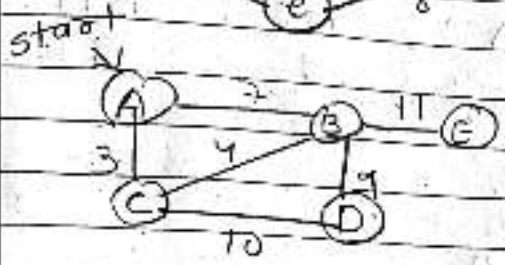
(1)

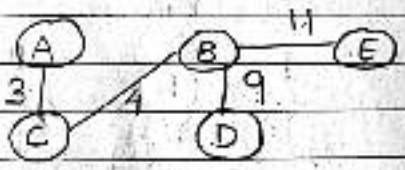




= 15

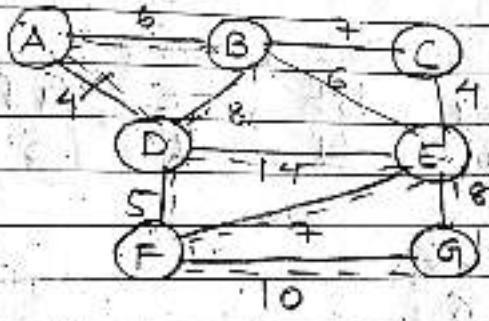
(2)

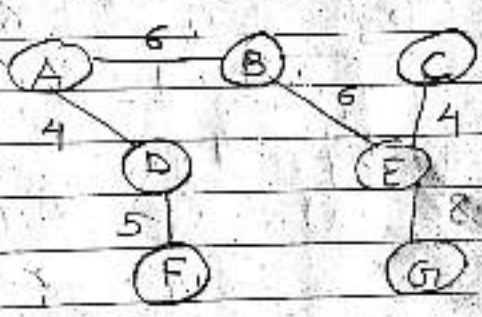




= 27

(3)



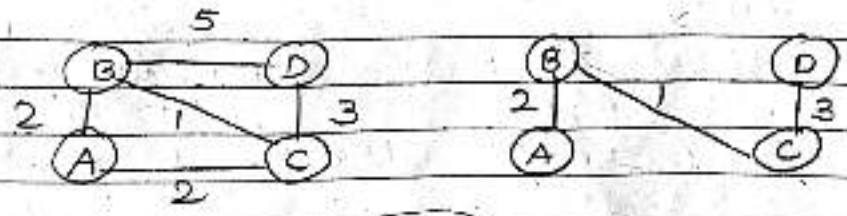


= 6+6+4+5+4+8

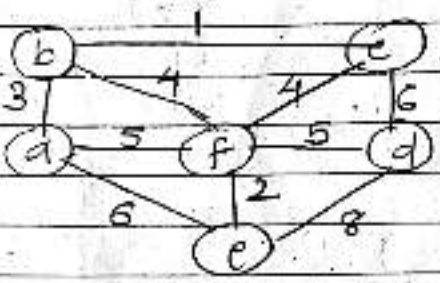
= 33

(ii) builds MST of a graph by adding edges to the spanning tree spanby it uses edges.

Date:
MON TUE WED THU FRI SAT

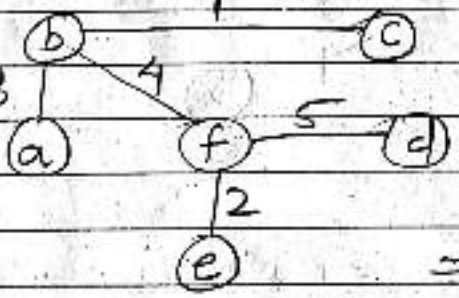


1) BC, BA, AC, DC, BD
1 2 2 3 5

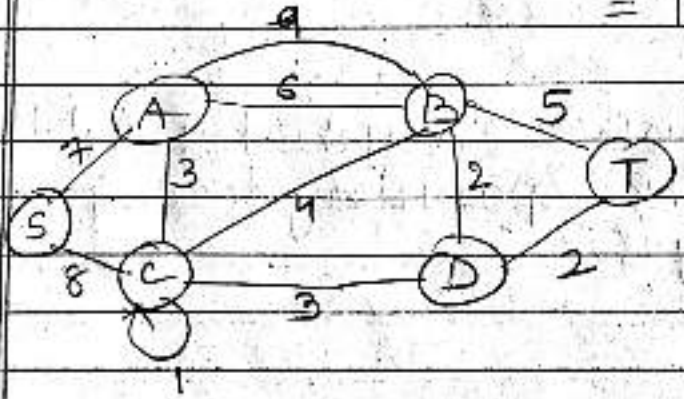


bc, fe, ba, bf, fc, af, fd,
1 2 3 4 4 5 5

cd, ae, de
6 6 8



= 1+4+5+3+2
= 15



Date:
MON TUE WED THU FR SAT

BD → DT → CA → CD → CB → BT → AB →
2 2 3 3 4 5 6

AS → SC
7 8

= 17