Department of Computer Science

Gujarat University

Certificate

Roll No: 30 Seat No: _____

This is to certify that Mr./~~Ms.~~ Rathod Ajinkya Sreekant student of MCA Semester – III has duly completed his/~~her~~ term work for the semester ending in December 2020, in the subject of _____ **Data Structures (DS)** towards partial fulfillment of his/~~her~~ Degree of Masters in Computer Applications.

Date of Submission
11 - December - 2020

Internal Faculty

Head of Department

Department Of Computer Science
Rollwala Computer Centre
Gujarat University

MCA –III

Subject: - DATA STRUCTURES

Name :- Ajinkya Rathod

Roll No.: - 30 **Exam Seat No.: -_____**

Sr. No.	Contents	Date	Signature
1.	Infix to Postfix	11/12/2020	
2	Binary Search Tree	11/12/2020	
3	Expression Tree	11/12/2020	
4	AVL Tree	11/12/2020	
5	UnWeighted Graph Shortest Path	11/12/2020	
6	Weighted Graph Shortest Path	11/12/2020	
7	Kruskal's Mininmum Spanning Tree	11/12/2020	
8	S-paly	11/12/2020	
9	Delete MIN	11/12/2020	
10	B Tree	11/12/2020	
11	B+ Tree	11/12/2020	
12	B* Tree	11/12/2020	

Date __/__/____
Page _____

Name :- Ajinkya Rathod.

class :- MCA -3.

Subject :- Data structures

Roll no :- 30.

Date / /
Page 1

Assignment - 1.

Infix, prefix, & Post fix noration :->

① $(x + y + z) * (A + B - c)$

-> Post fix :-

$((xy+)+z) * ((AB+)-c)$
$((xy+)z+) * ((AB+)C-)$
$xy+z+AB+C-*$

-> Prefix :-

$((x+y)+z) * ((A+B)-c)$
$(+(+xy)z) * (+(+AB)-c)$

$++xyz*-+ABC$

② $(A/B - c) + (D * E)$

-> Post fix :-

$((A/B)-c) + (D*E)$
$(AB/-c) + (DE*)$
$AB/c- DE*+$

→ prefix

((A/B)-C) + (D*E)
(/AB)-C) + (*DE)
(-/ABC) + (*DE)

+ -/ABC *DE

③. ((((A+B) + (C/D))) * (E-(F*G)))

→ Post fix:-

(((AB+)+(CD/)) * (E-(FG*)))
((AB+CD/+) * (EFG*-)

AB+CD/+EFG*-*

→ Prefix:-

((C + AB) + (/CD)) * (E-(*FG))
(++AB/CD) * (-E*FG)
* ++AB/CD-E*FG

(4). ((x * y) * (z * w) - (A + B * C - D)

→ Post fix :-

(xy * * z * w) - (A + B * C - D)

xy * z * w * - (A + BC * - D)

xy * z * w * - ABC * + D

xy * z * w * ABC * + D.

→ Pre fix :-

* * * xyzw - (A + * BC - D)

* * * xyzw - - + A * BCD

- * * * xyzw - + A * BCD

(5). A / B / c + D * E - F

→ Post fix :-

A B / c / + D E * - F

A B / c / D E * + F -

→ Prefix :-

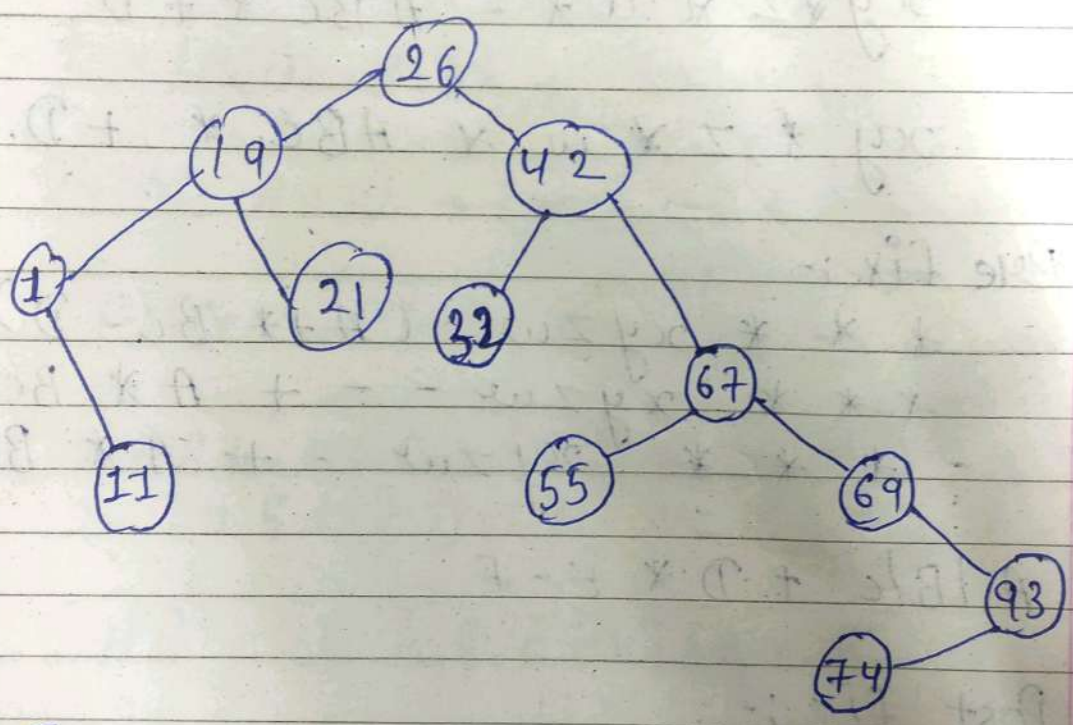
/ / A B C + P * E - F

/ / A B C + * D E - F

- + / / A B C * D E F

Q.2. Create BST for following data & perform all 3 traversals on resultant tree.

(1). 26, 19, 42, 67, 21, 1, 69, 55, 93, 74, 32, 11

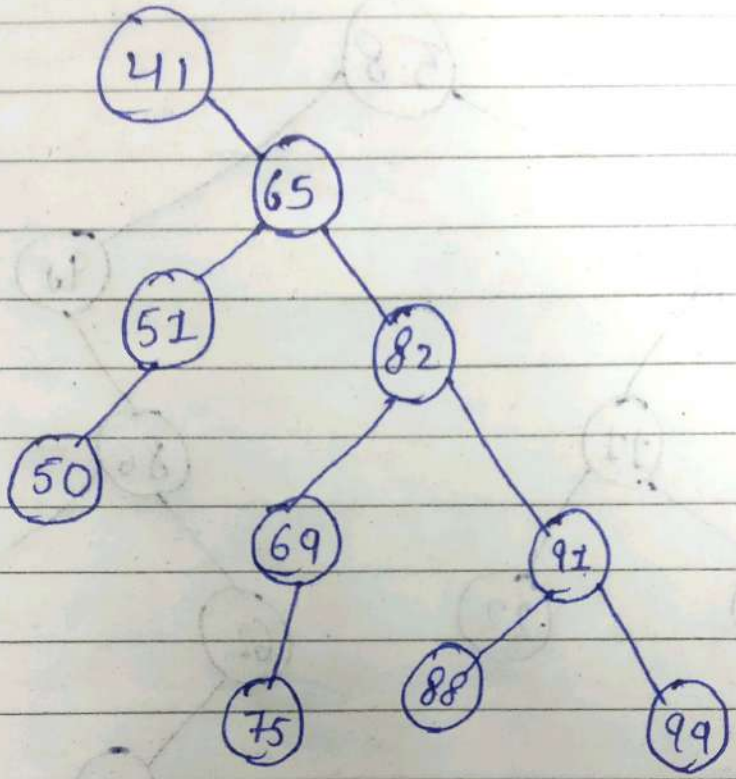


→ In order Traversals:- (L Root R)
1, 11, 19, 21, 26, 32, 42, 55, 67, 69, 74, 93.

→ Pre order Traversals:- (Root LR)
26, 19, 1, 11, 21, 42, 32, 67, 55, 69, 93, 74.

→ Post order Traversals:- (L R Root)
11, 1, 21, 19, 32, 55, 74, 93, 69, 67, 42, 26

(2). 41, 65, 82, 91, 51, 50, 69, 75, 88, 99.

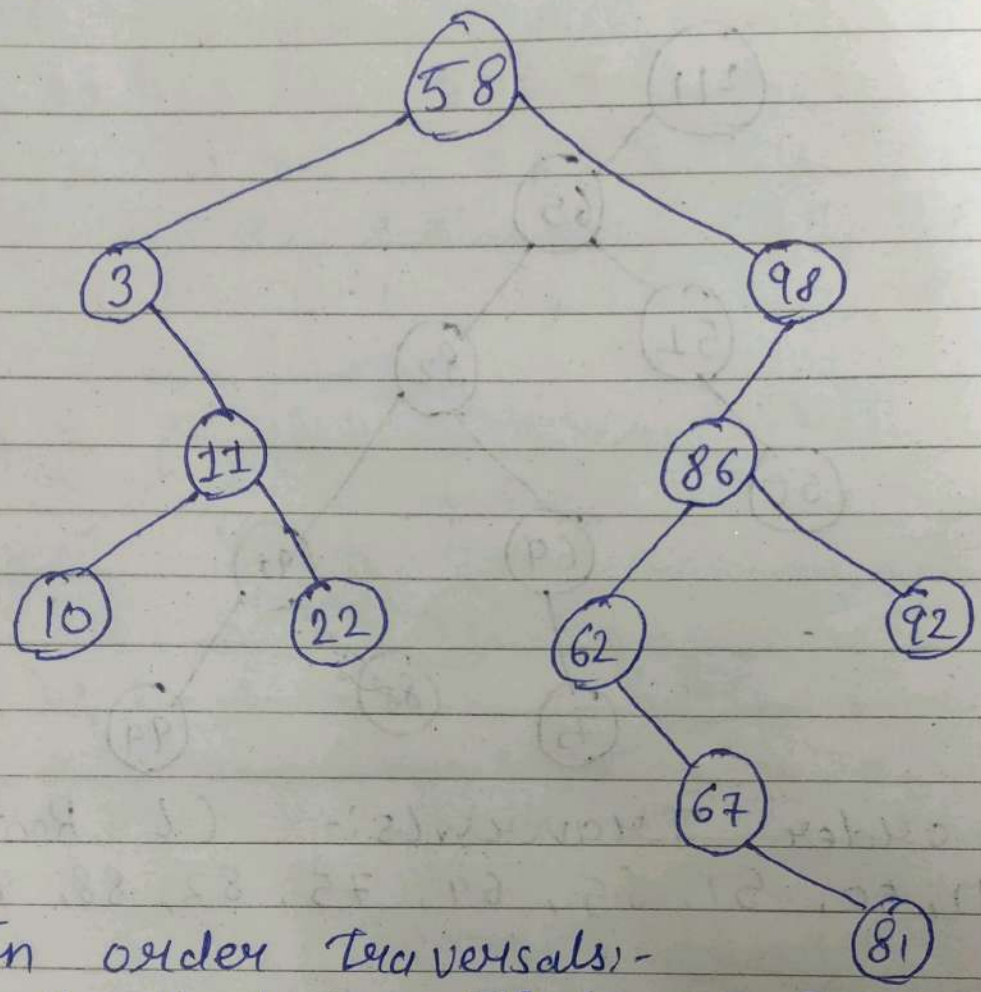


→ In order Traversals :- (L Root R)
41, 50, 51, 65, 69, 75, 82, 88, 91, 99.

→ Pre order Traversals :- (Root LR)
41, 65, 51, 50, 82, 69, 75, 91, 88, 99.

→ Post order Traversals :- (L R Root)
50, 51, 75, 69, 88, 99, 91, 82, 65, 41

③ 58, 3, 98, 11, 86, 22, 62, 67, 81, 92,

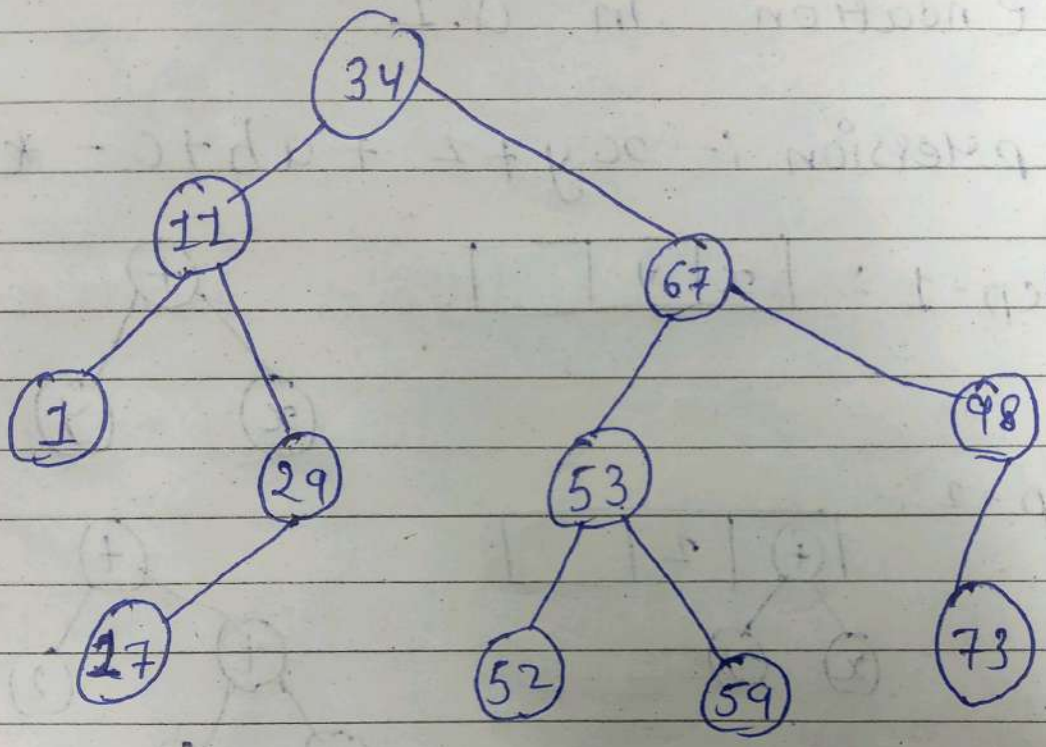


→ In order Traversals:-
3, 10, 11, 22, 58, 62, 67, 81, 86, 92, 98

→ Pre order Traversals:-
58, 3, 11, 10, 22, 98, 86, 62, 67, 81, 92

→ Post order Traversals:-
10, 22, 11, 3, 81, 67, 62, 92, 86, 98, 58.

(4) 34, 11, 67, 98, 1, 73, 53, 29, 52, 17, 59.



→ In order Traversals:-
1, 11, 17, 29, 34, 52, 53, 59, 67, 73, 98

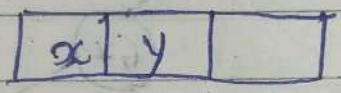
→ In pre order Traversals:-
34, 11, 1, 29, 17, 67, 53, 52, 59, 98, 73.

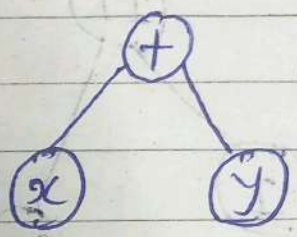
→ Post order Traversals:-
1, 17, 29, 11, 52, 59, 53, 73, 98, 67, 34

Q.3. Create Expression Tree from the equation in Q.1.

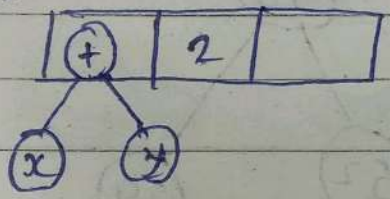
1) Expression :- x y + z + a b + c - *

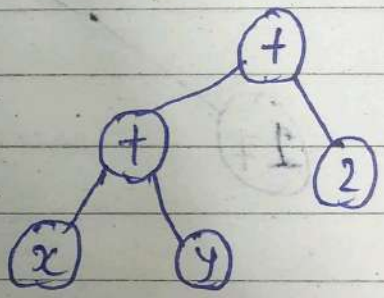
Step-1 :



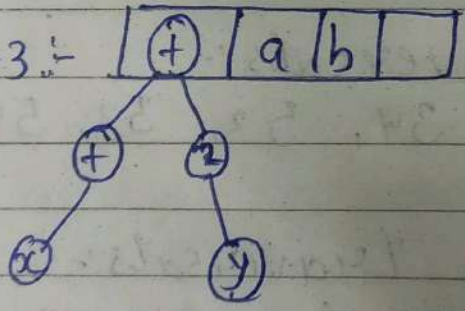


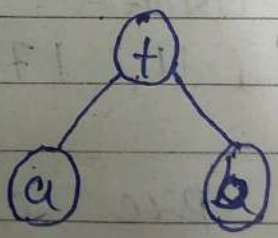
Step-2 :-



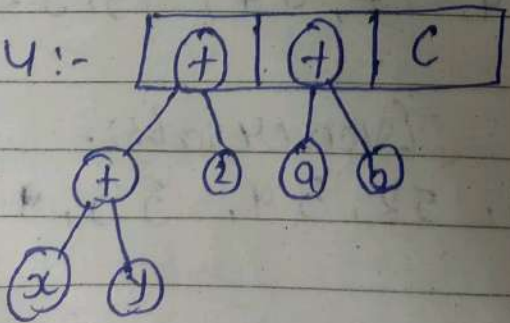


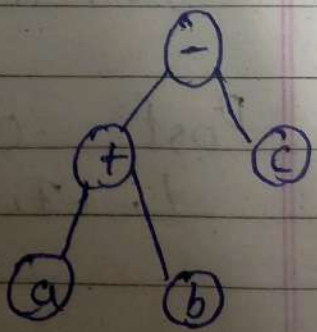
Step-3 :-



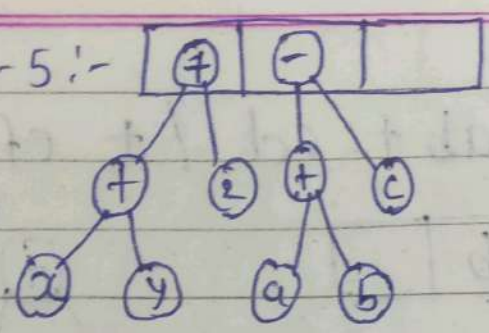


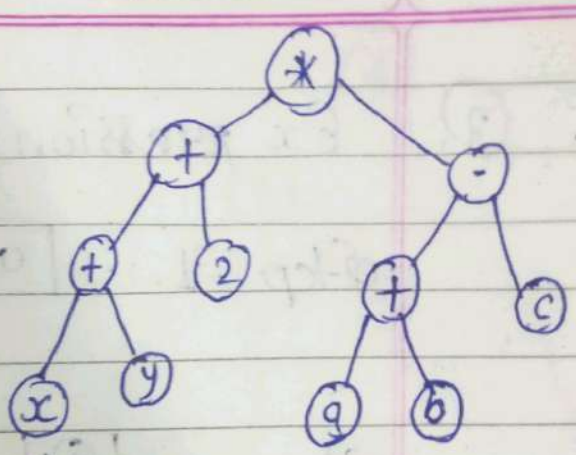
Step-4 :-





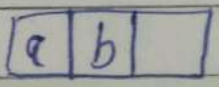
Step -5 :-

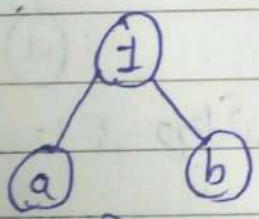




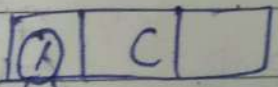
(2). Expression :- a b / c - d e * +

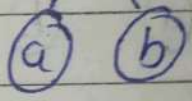
Step-1 :-

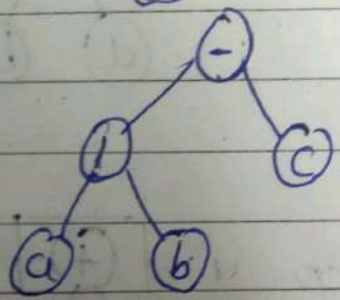




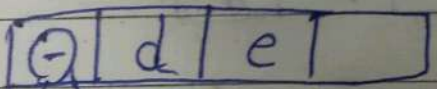
Step-2 :-

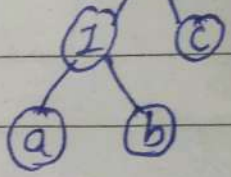


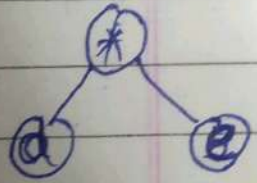




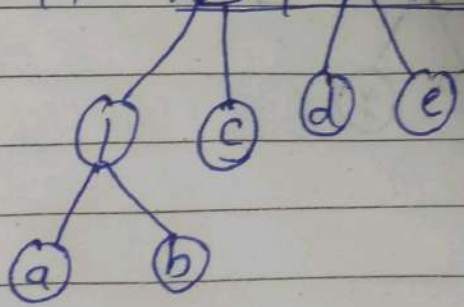
Step-3 :-

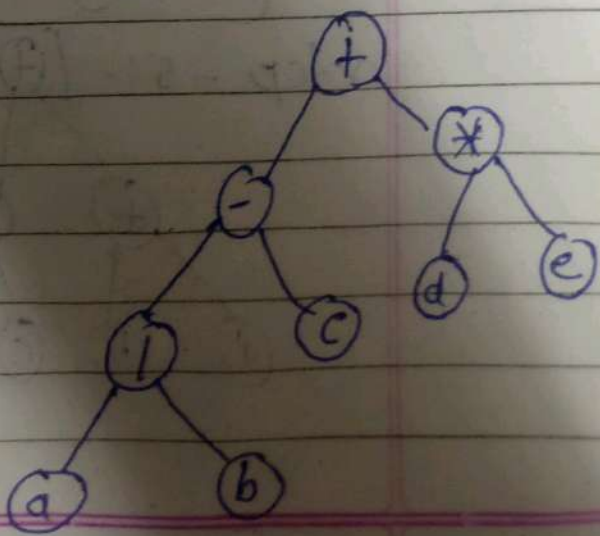






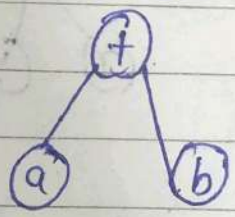
Step -4 :-



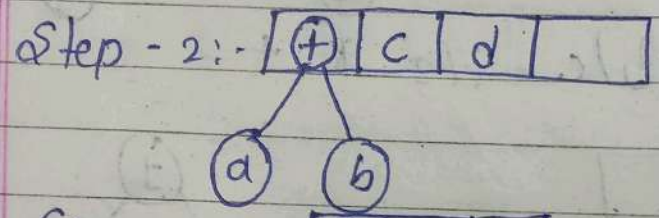


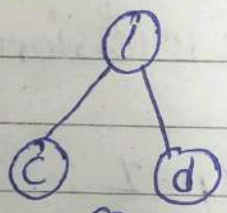
③ Expression:- ab + cd / + efg * - *

Step -1 :-

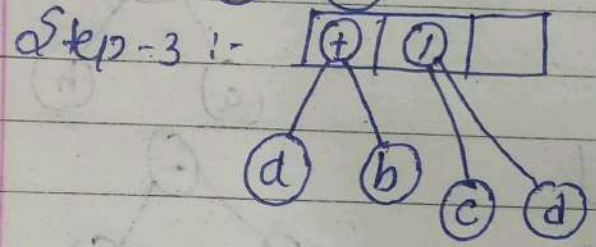


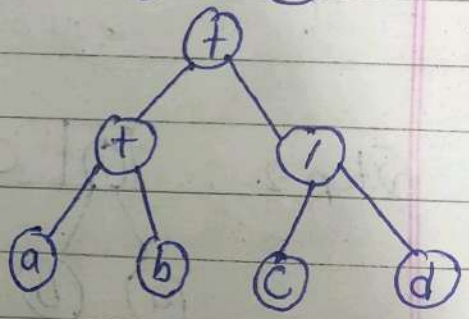
Step - 2:-



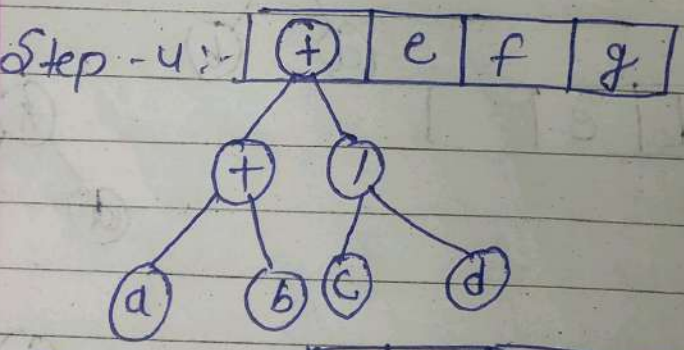


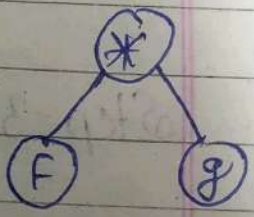
Step - 3 :-



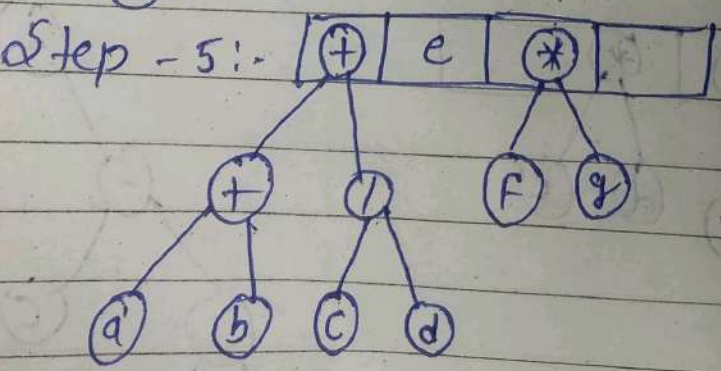


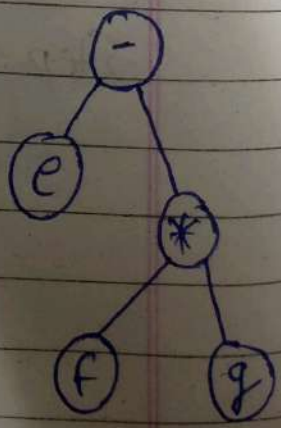
Step - 4 :-



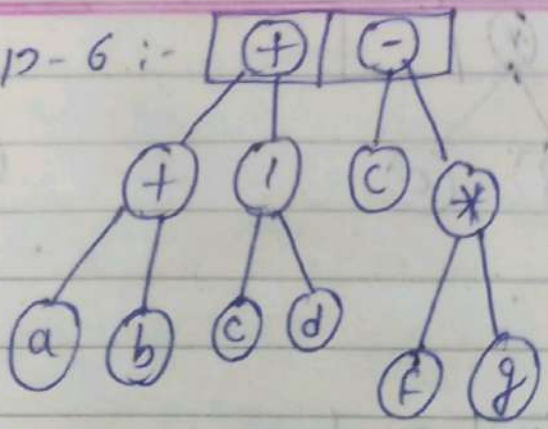


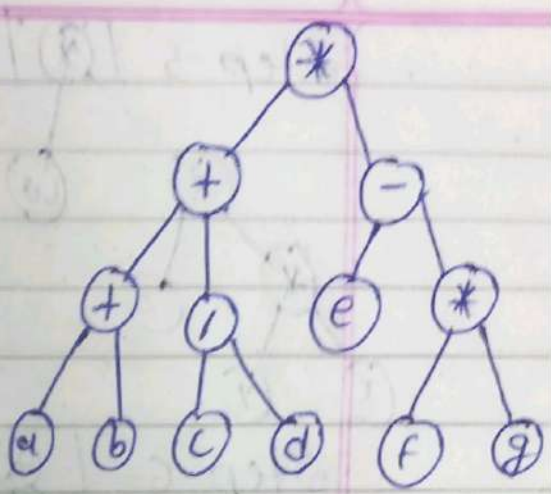
Step - 5 :-





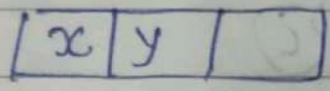
Step-6 :-

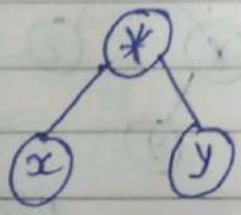




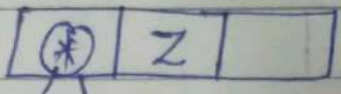
(4) Expression :- $xy * z * w * ABC * + D - -$

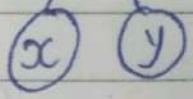
Step-1

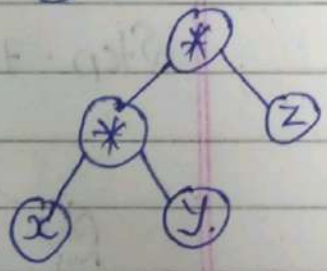




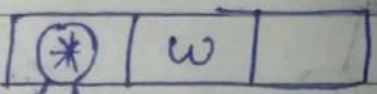
Step-2

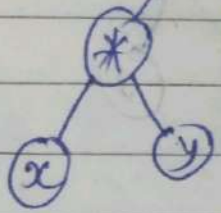


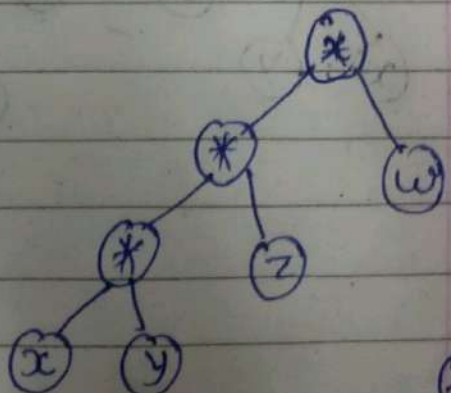




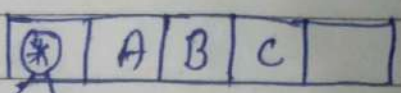
Step-3.

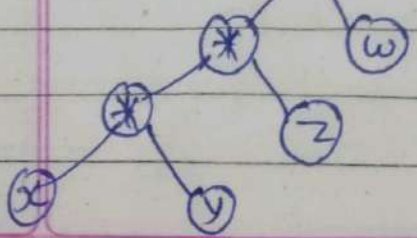


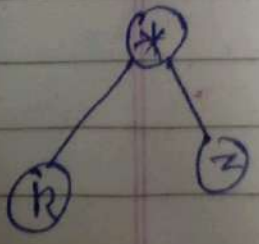




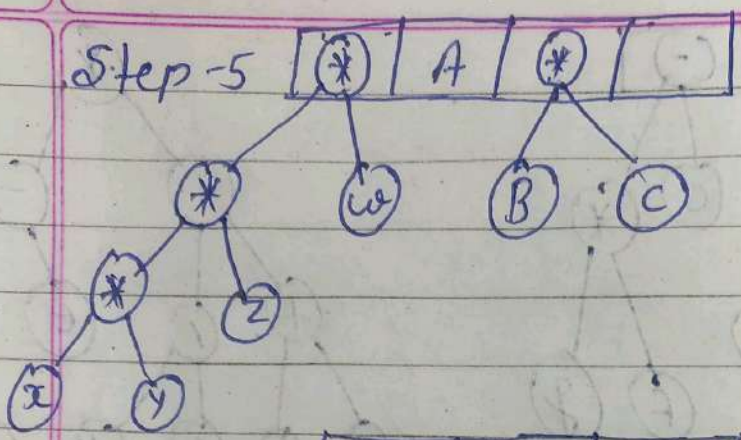
Step-4 :-

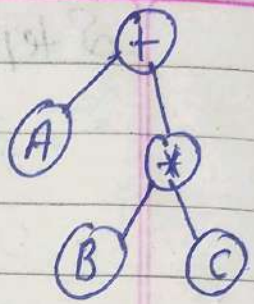




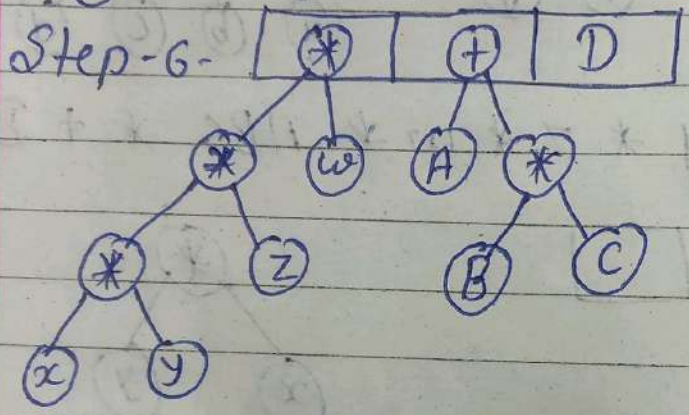


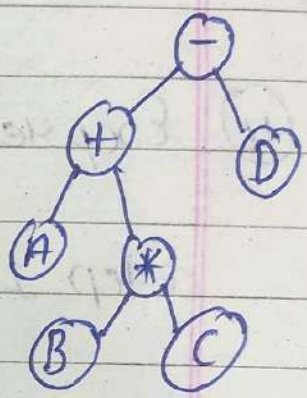
Step-5



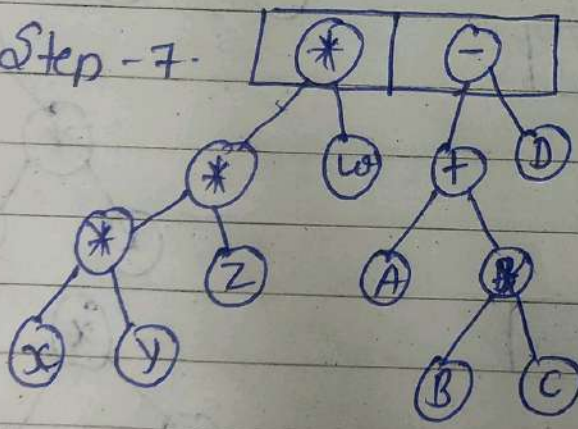


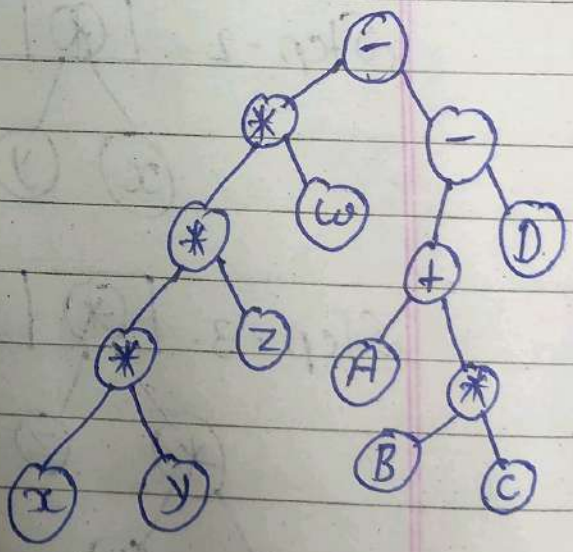
Step-6





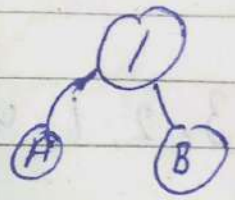
Step-7



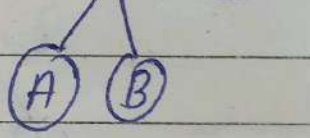


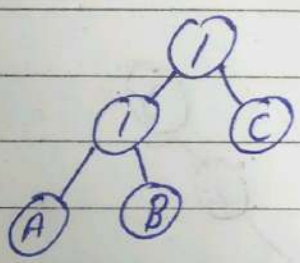
(5) Expression :- AB / C / DE * + F - .

Step-1 [A | B]

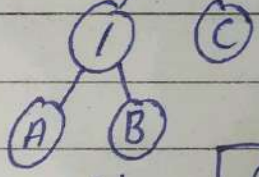


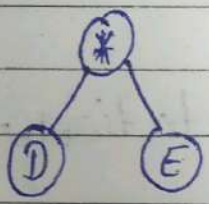
Step-2 [/ | C]



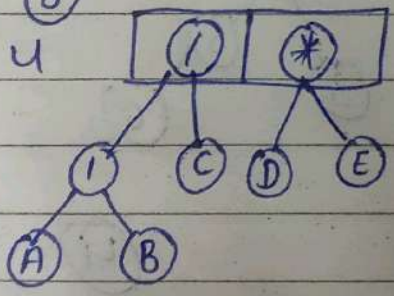


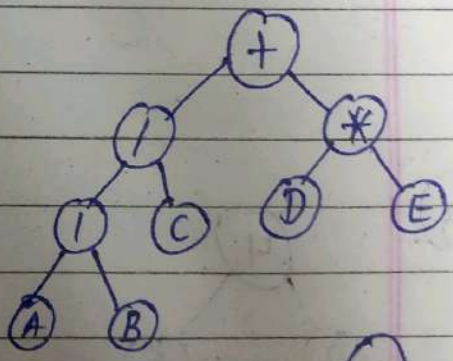
Step-3 [/ | D | E]



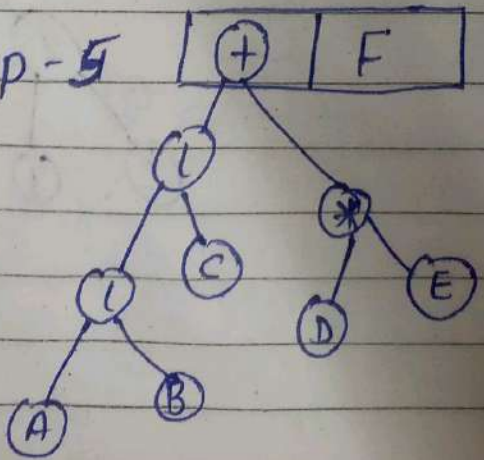


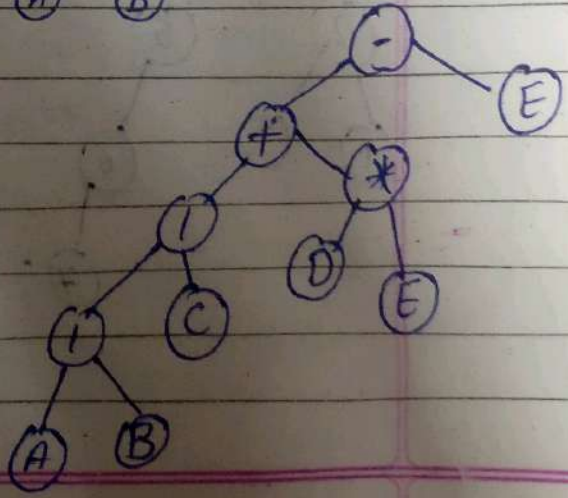
Step-4 [/ | *]





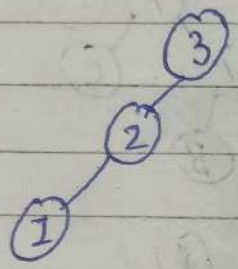
Step-5 [+ | F]



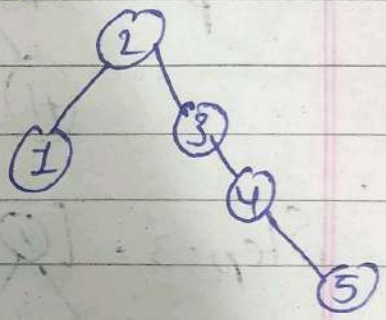


Q.4. AVL Tree (Rotation LL, RR, LR, RL)
(Balance factor $L^L - h^R$ (-1, 0, 1)

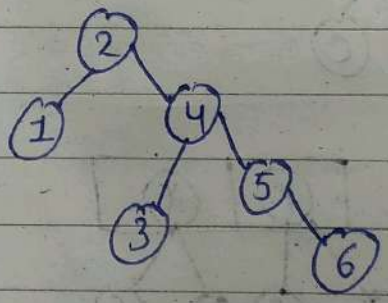
(1). 3, 2, 1, 4, 5, 6, 7, 16, 15, 14, 13, 12, 11, 10, 8, 9



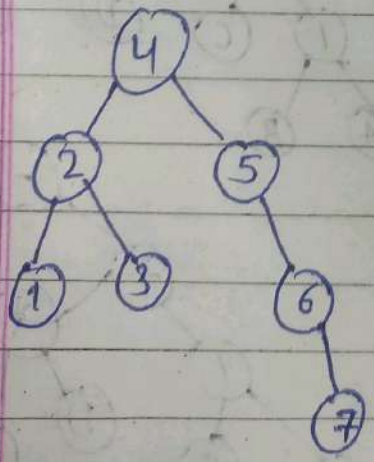
Rotation - 1.

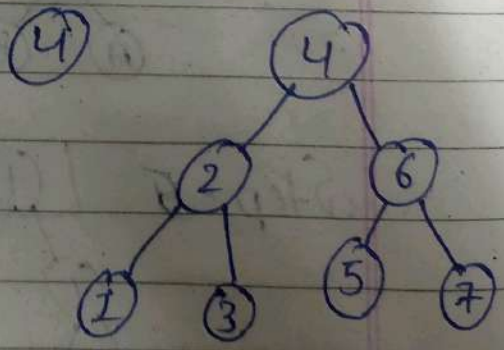


Rotation - 2



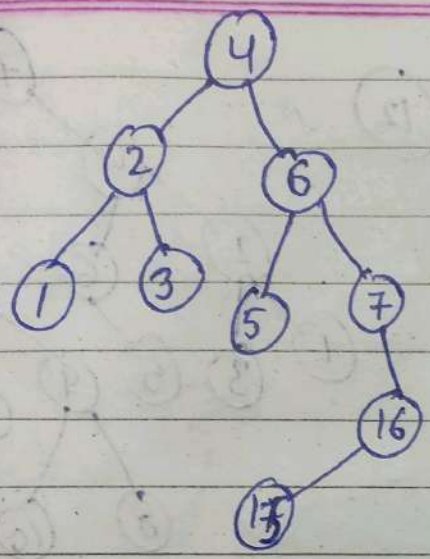
(3).



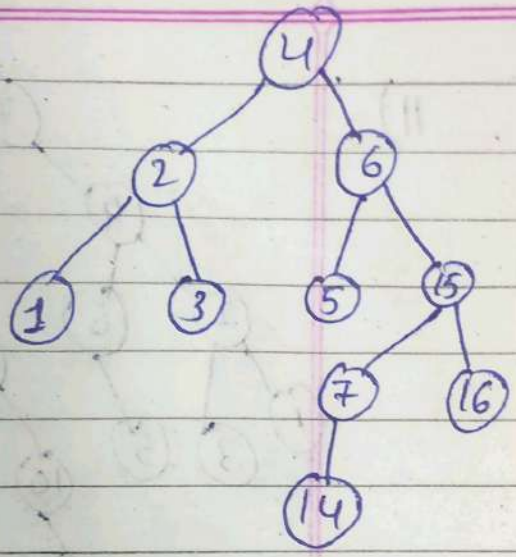


Date ___/___/___
Page 15

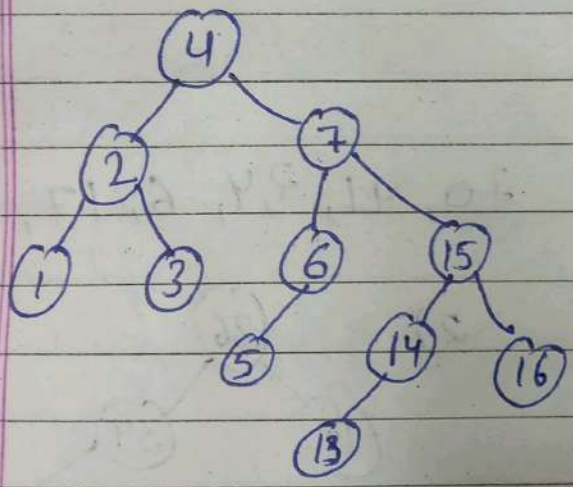
5.



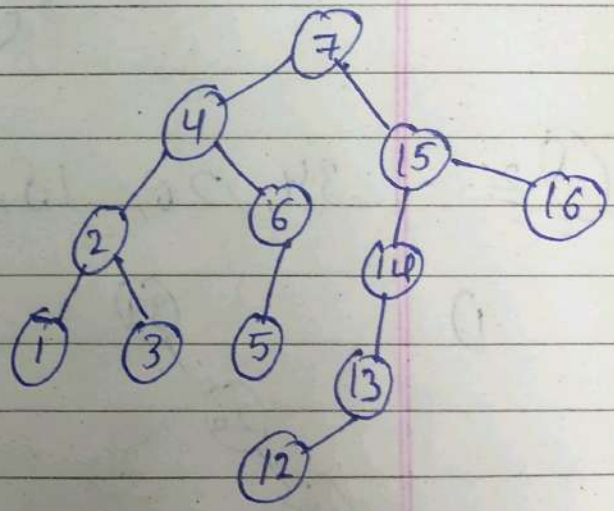
6.



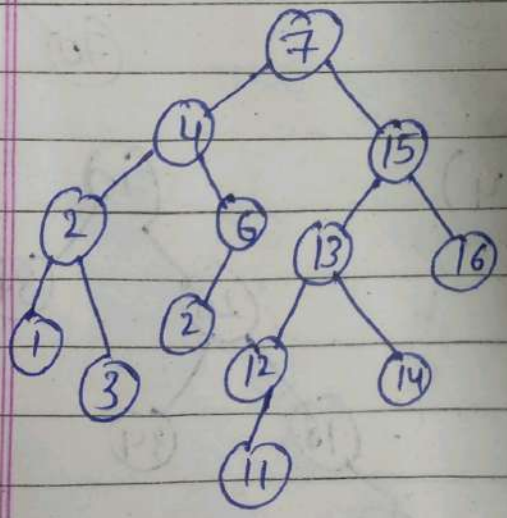
7.



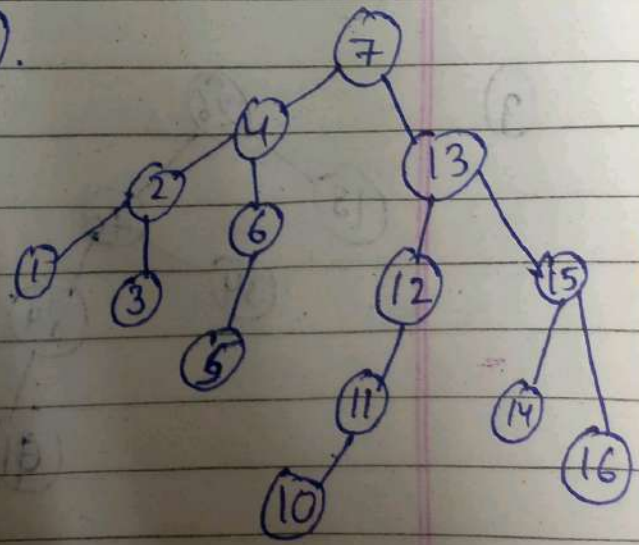
8.



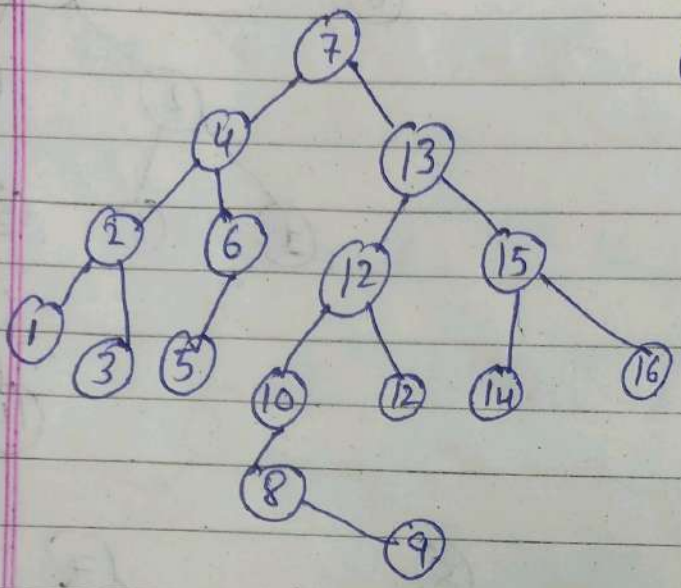
9.



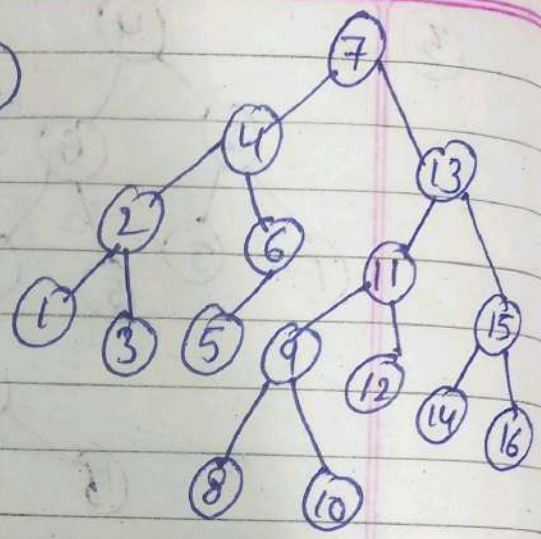
10.



11)

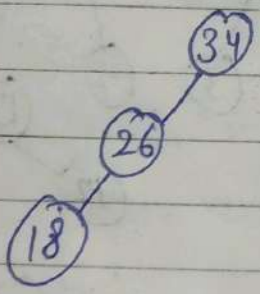


12

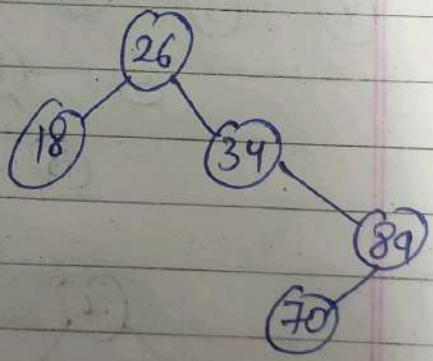


Q2. 34, 26, 18, 84, 70, 91, 24, 6, 17, 48, 55, 10

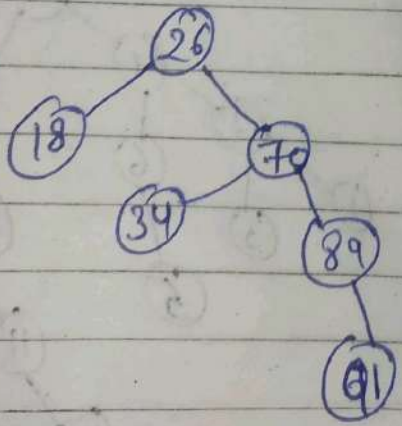
1)



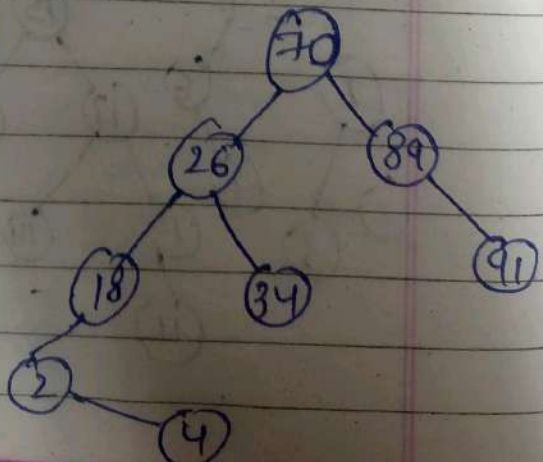
2)



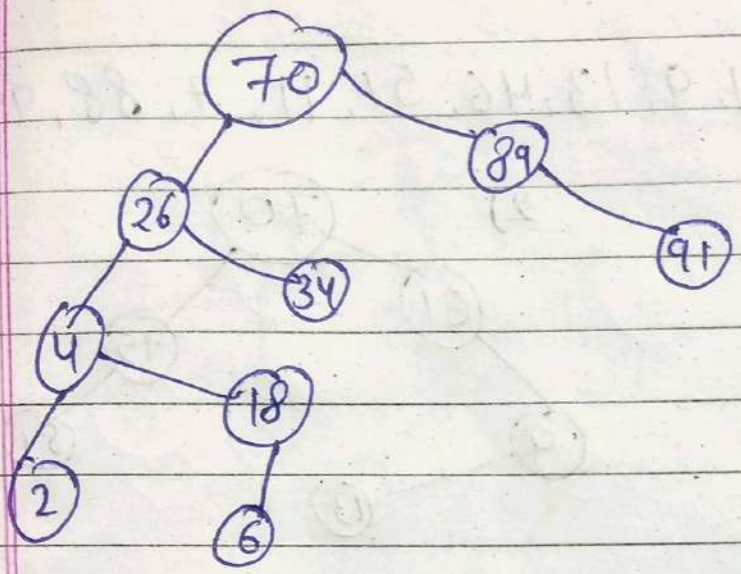
3)



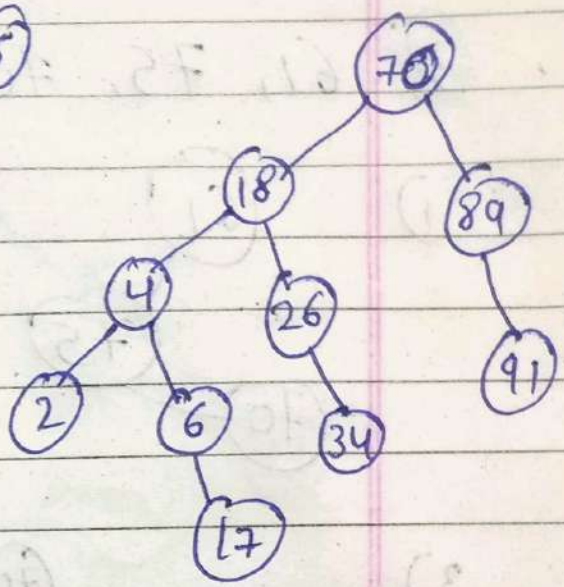
4)



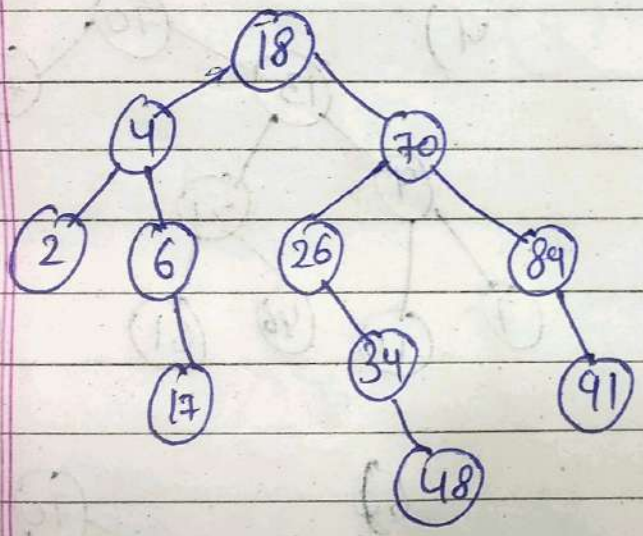
5)


70
26
89
4
34
91
18
2
6

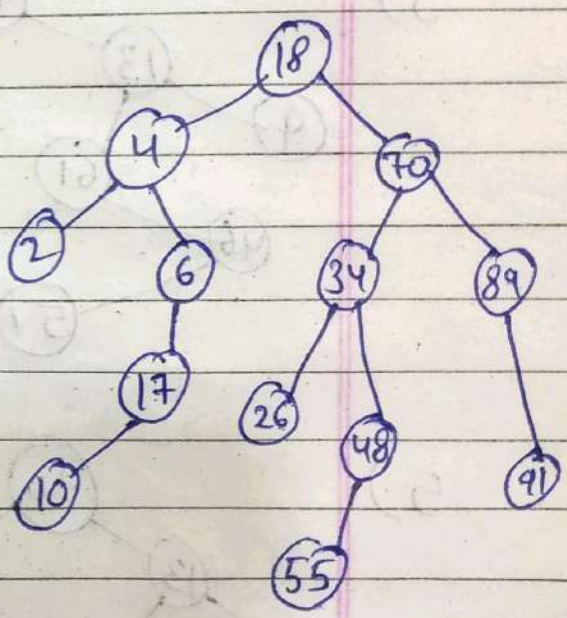
6)


70
18
89
4
26
91
2
6
34
17

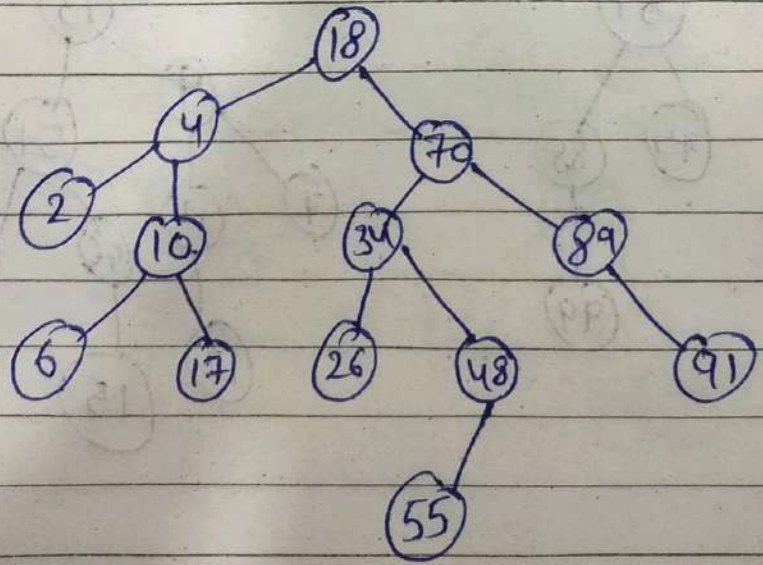
7)


18
4
70
2
6
26
89
17
34
91
48

8)

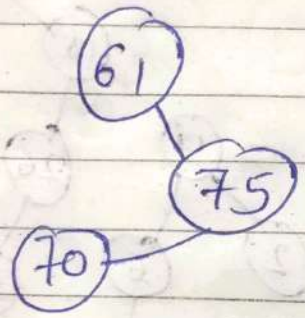

18
4
70
2
6
34
89
17
26
48
91
10
55

9)

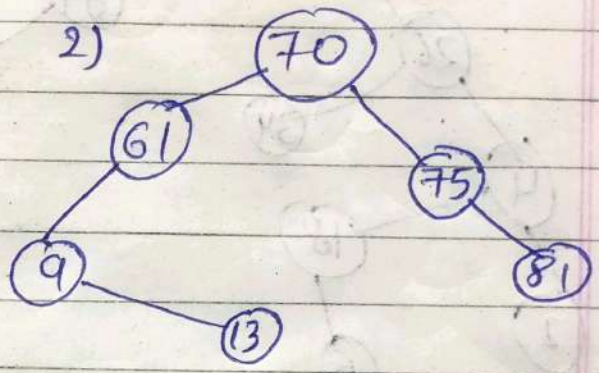

18
4
70
2
10
34
89
6
17
26
48
91
55

(3) 61, 75, 70, 81, 9, 13, 46, 51, 11, 7, 88, 99, 12, 15

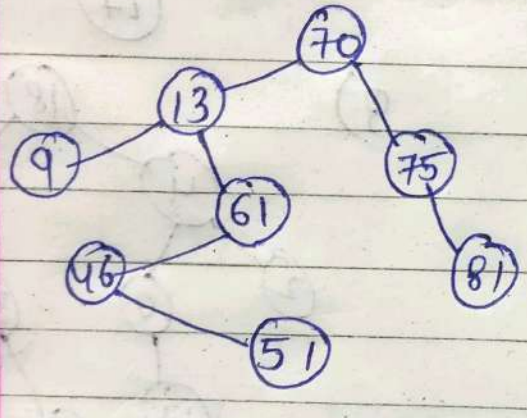
1)



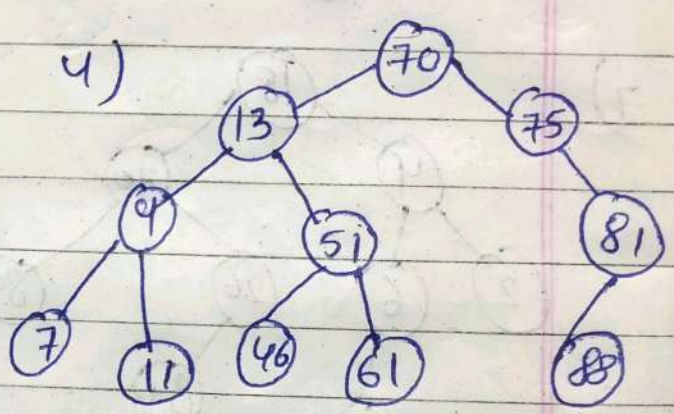
2)



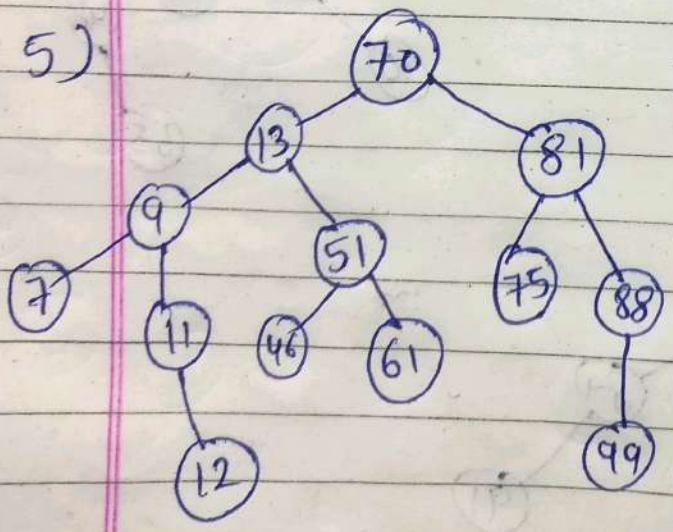
3)



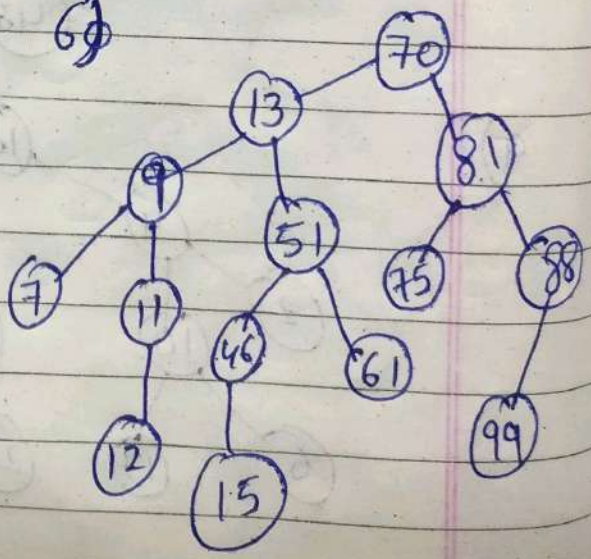
4)



5)



6)

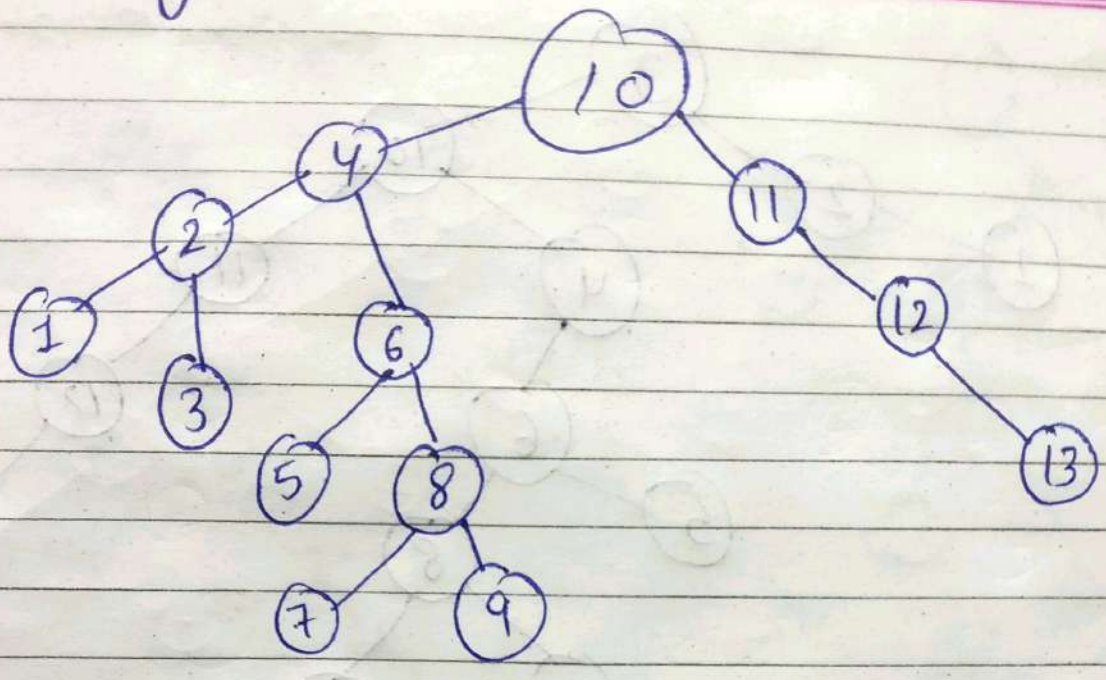


(1)

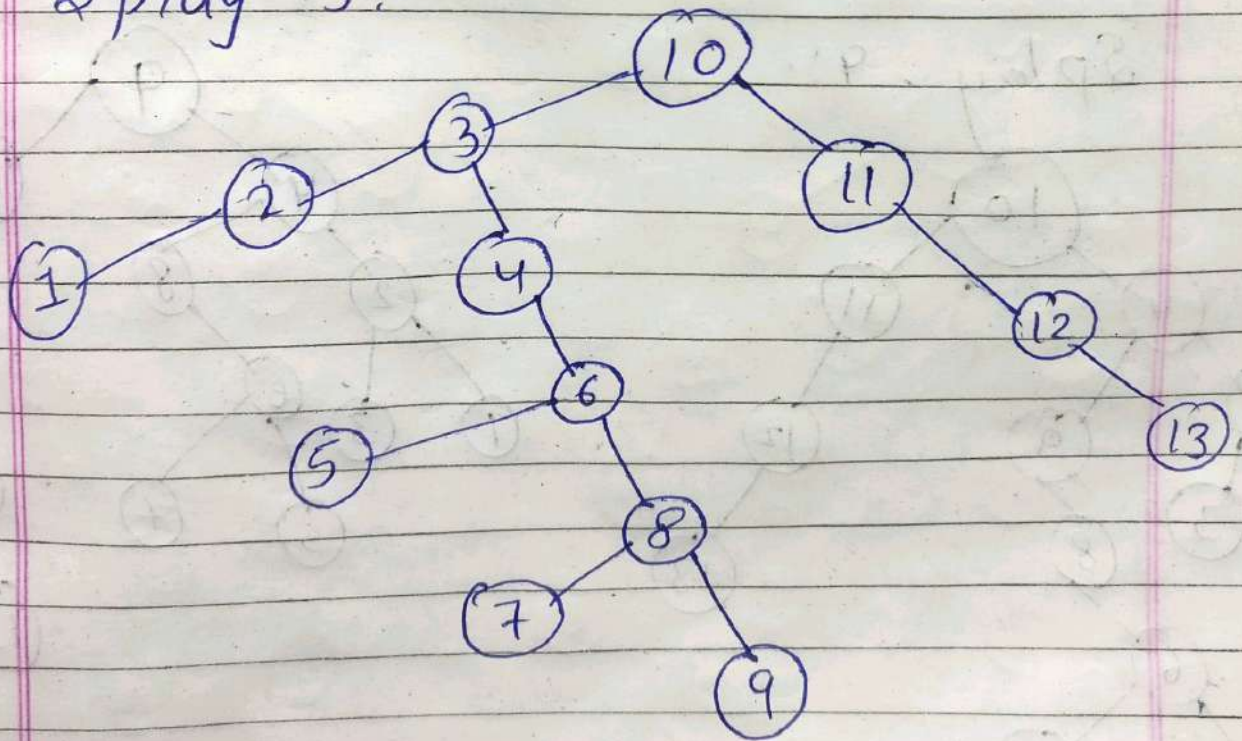
Splay tree :-

Date __/__/____
Page 19

(1)



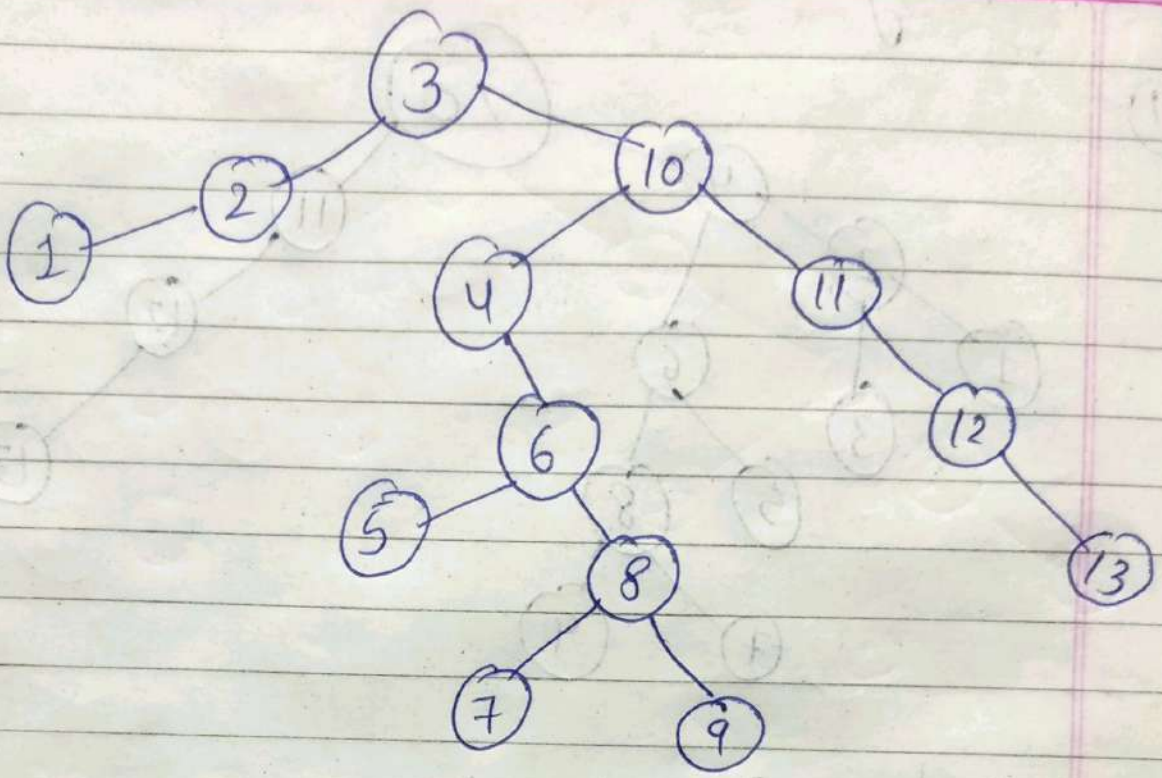
Splay 3 :-



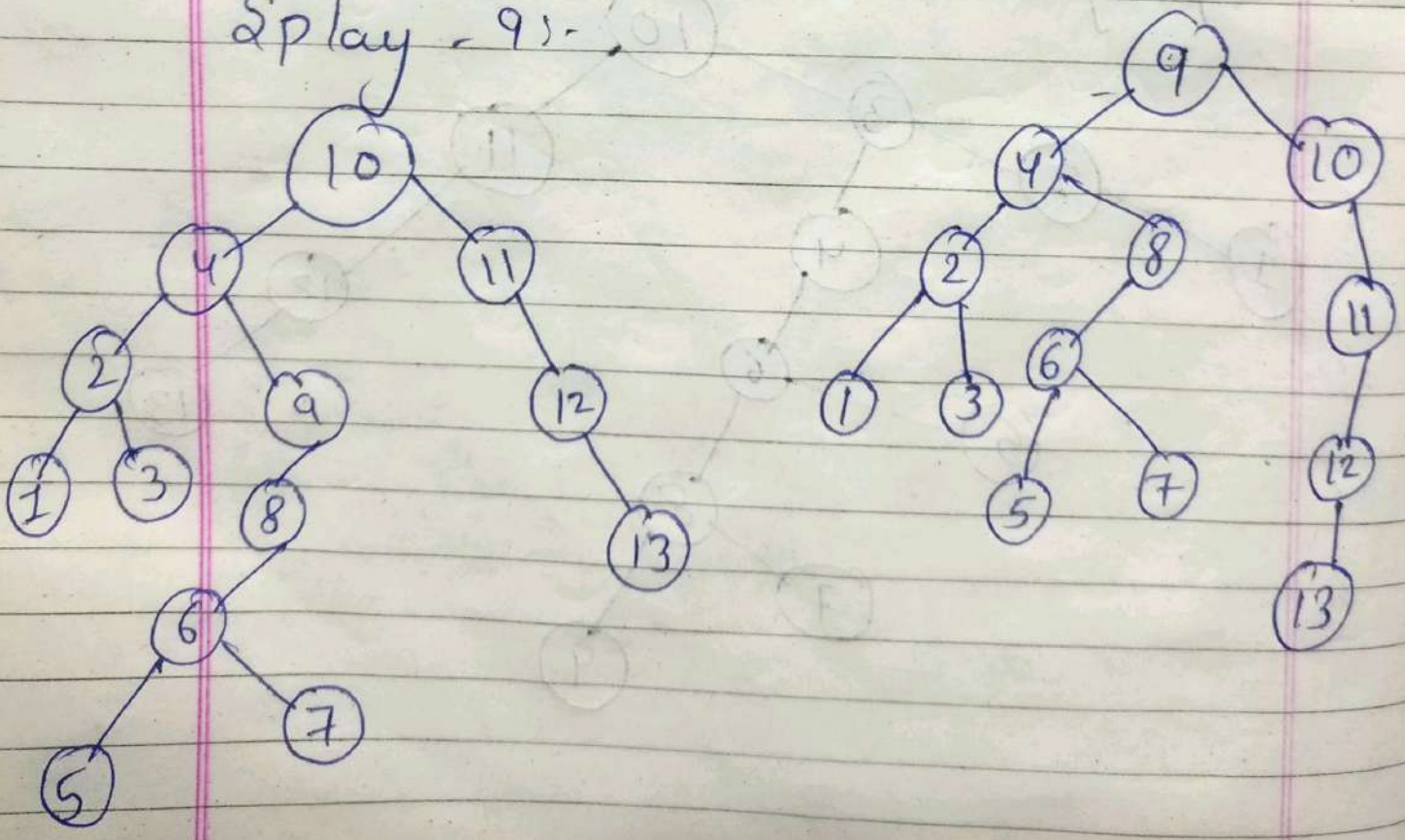
(2)

Date ___/___/___

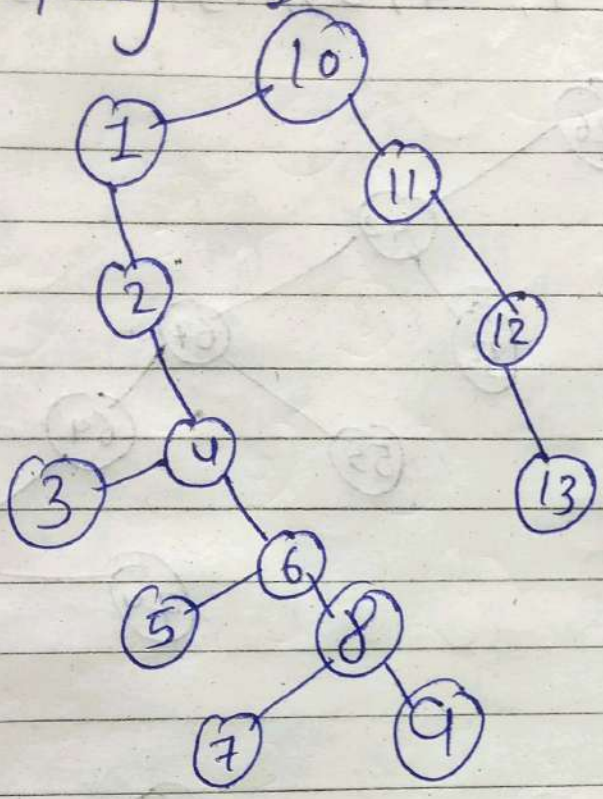
Page 20

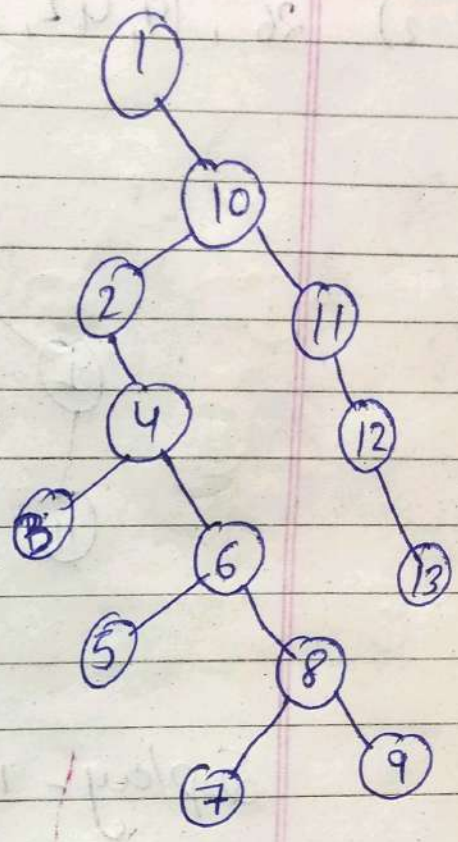


Splay - 9 :-

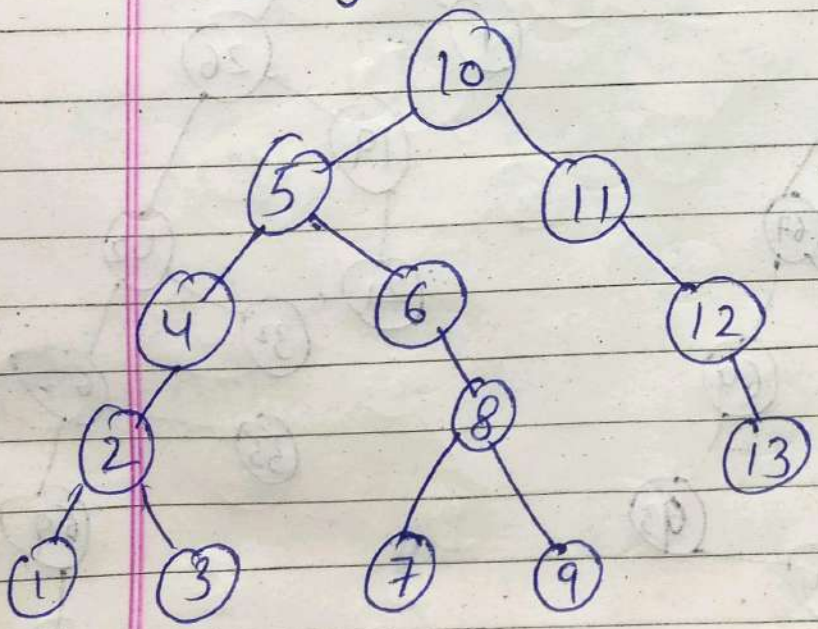


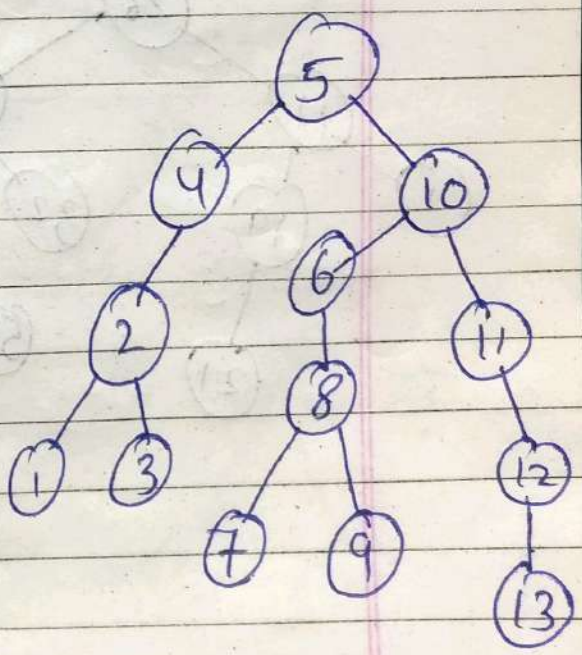
Splay - 1 :-





Splay 5 :-



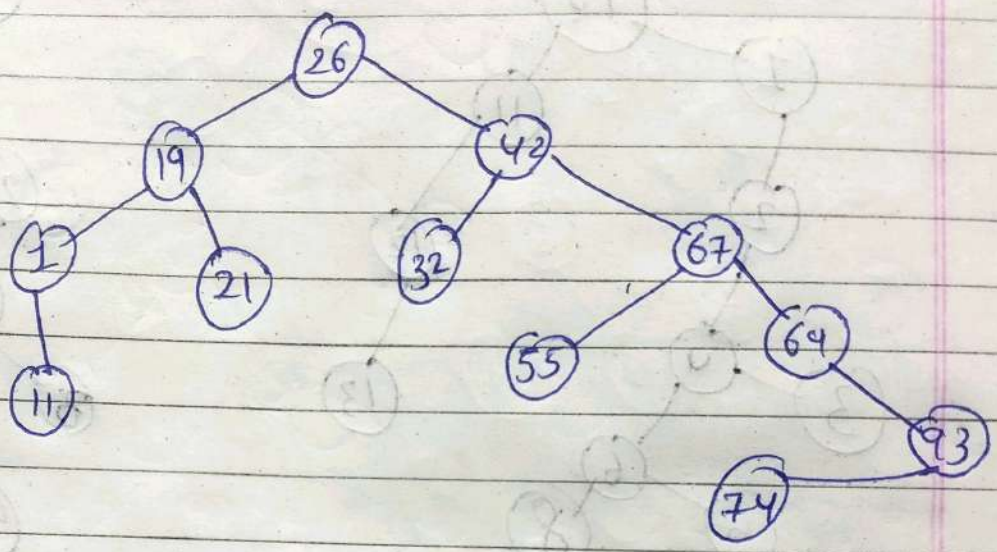


(4)

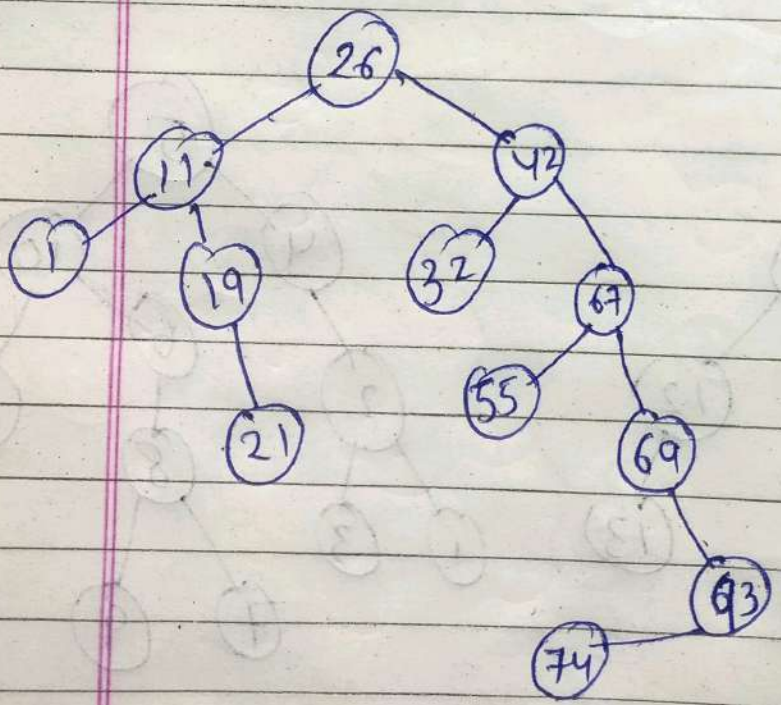
Date / /

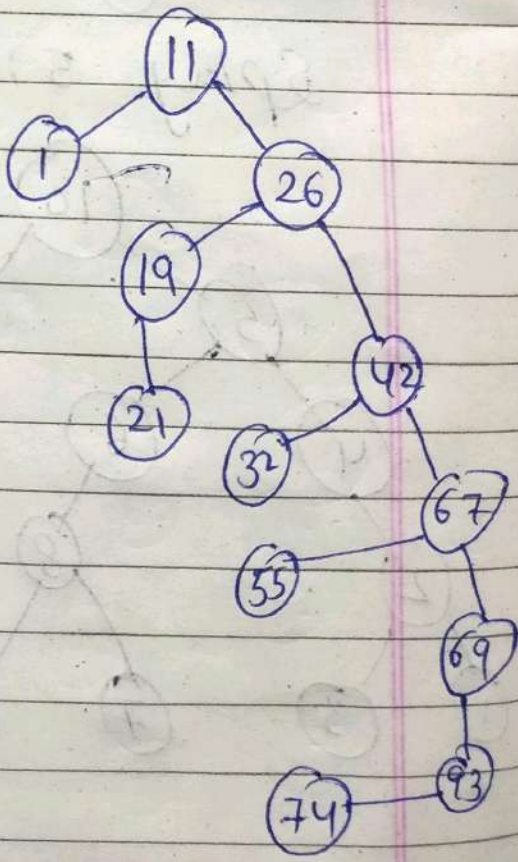
Page 22

2) 26, 19, 42, 67, 21, 1, 69, 55, 93, 74, 32, 11.



Splay - 11:-

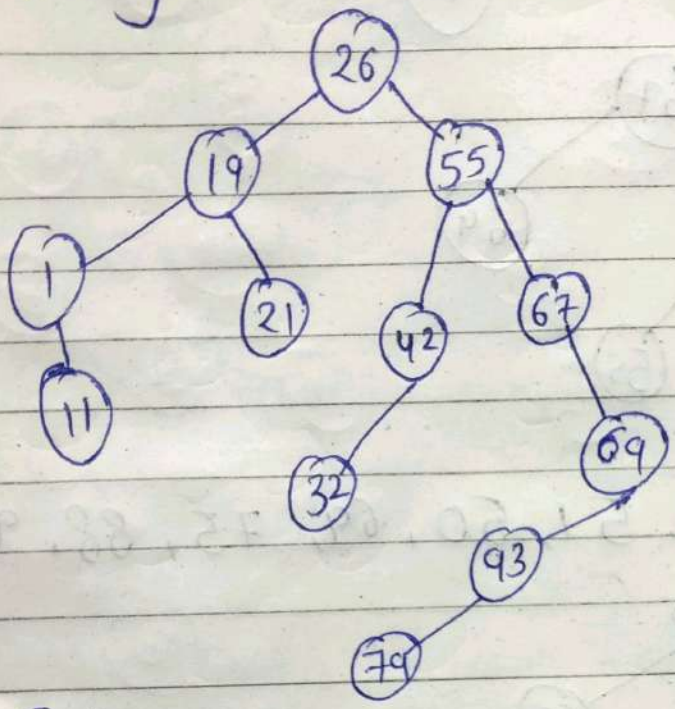


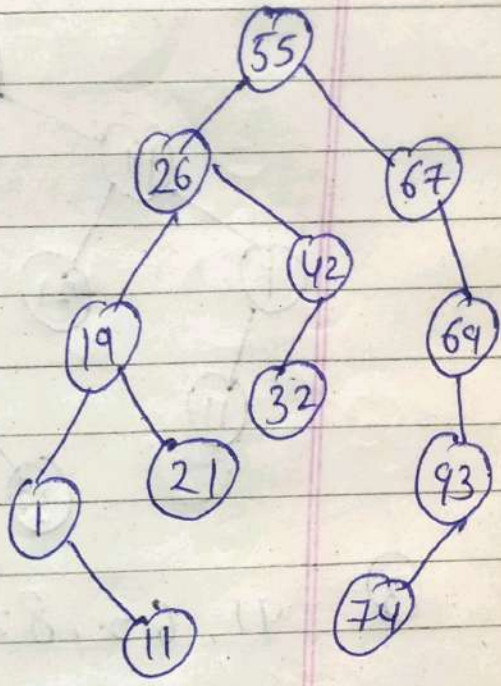


(3)

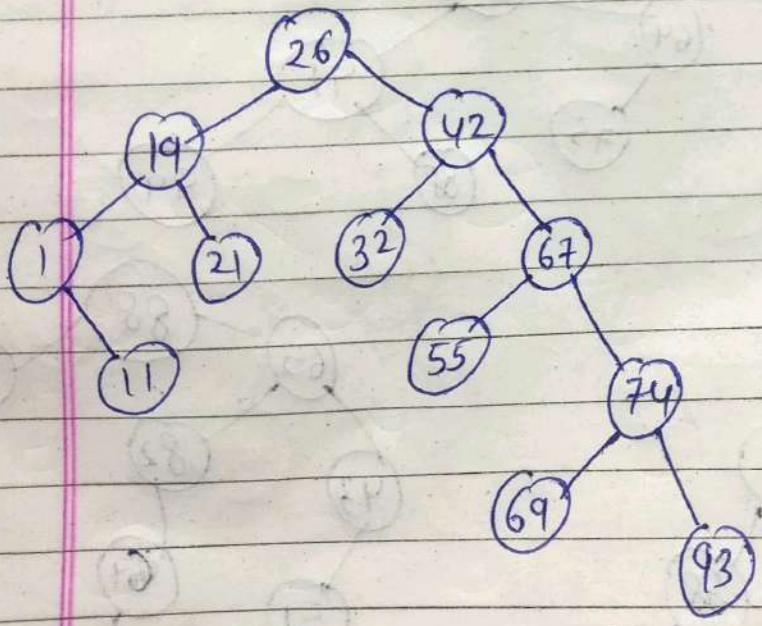
Date __/__/____
Page 23

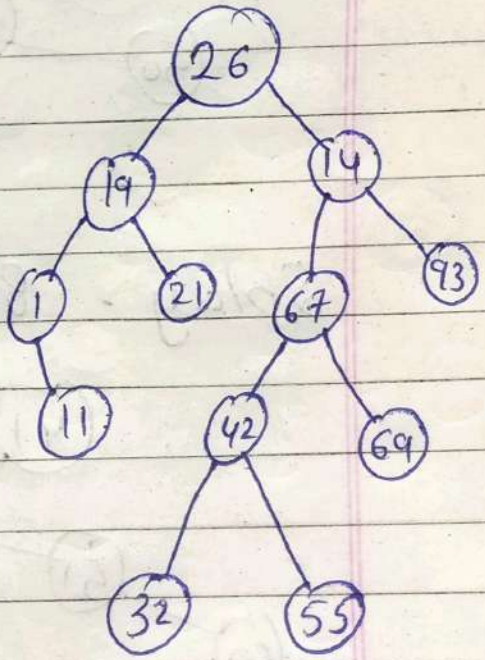
Splay - 55 :-



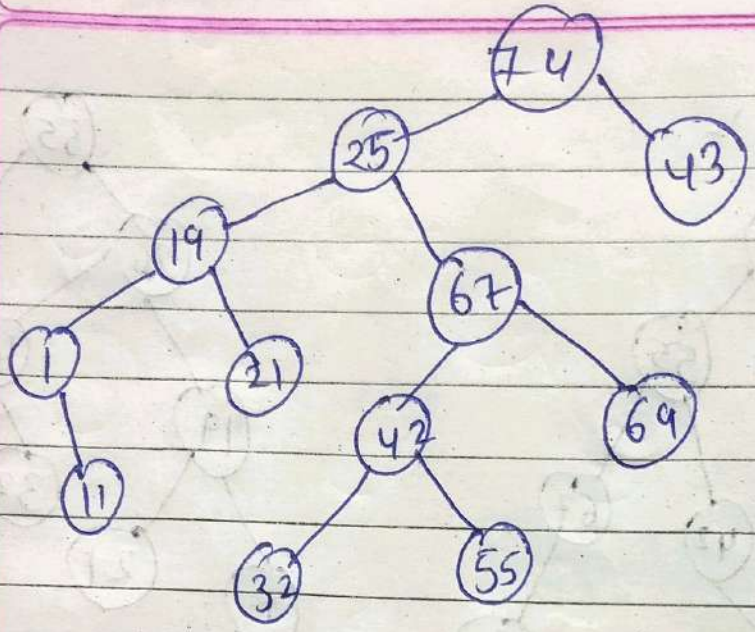


Splay = 74 :-

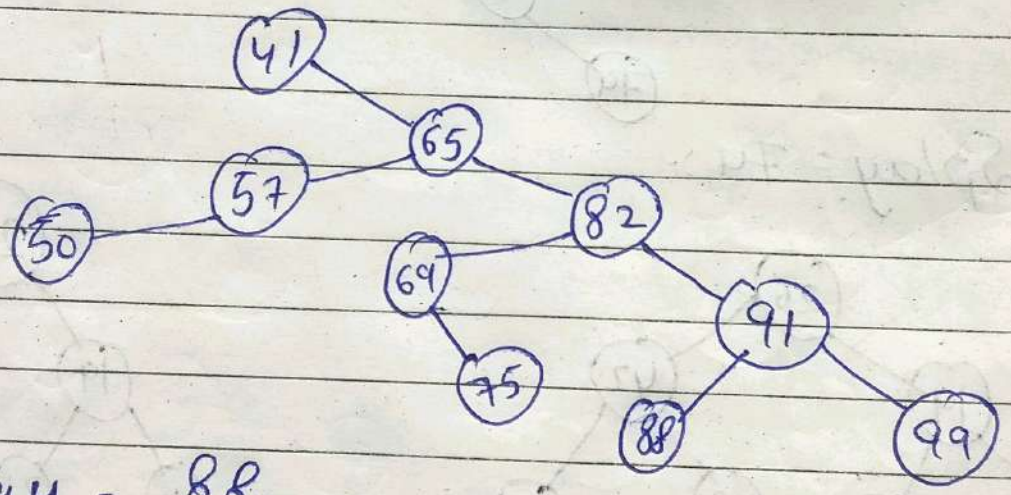




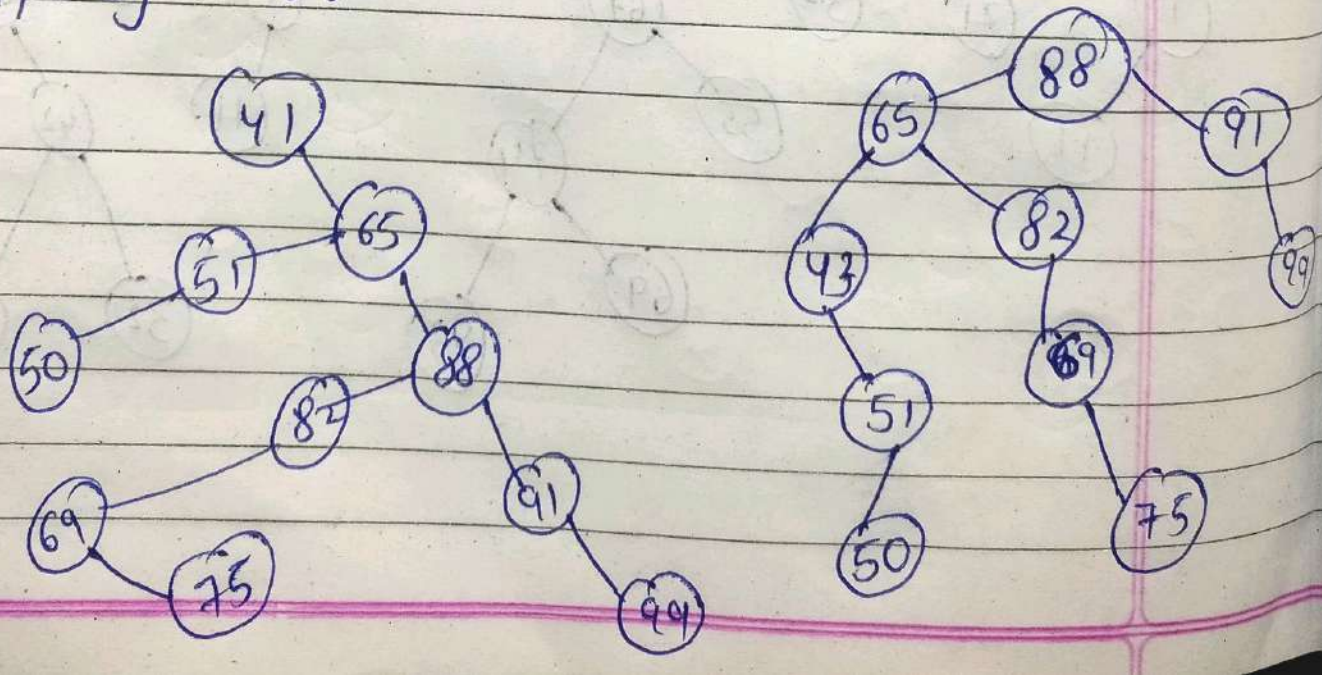
6



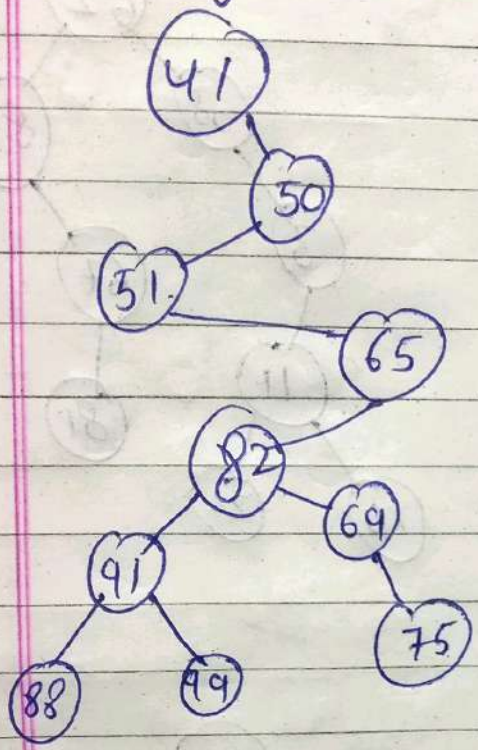
3 41, 65, 82, 91, 51, 50, 69, 75, 88, 99.

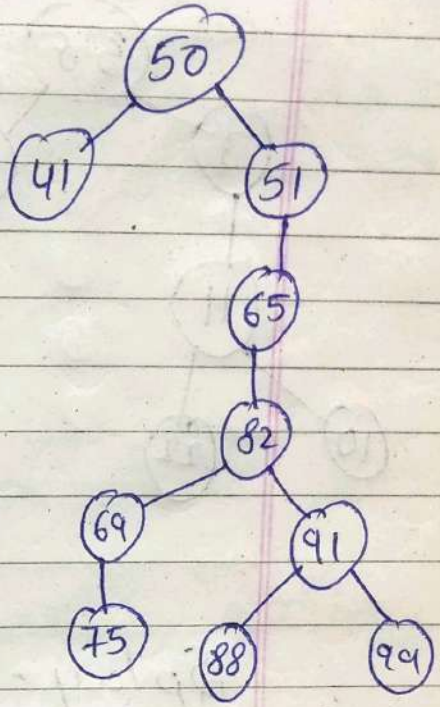


Splay - 88.

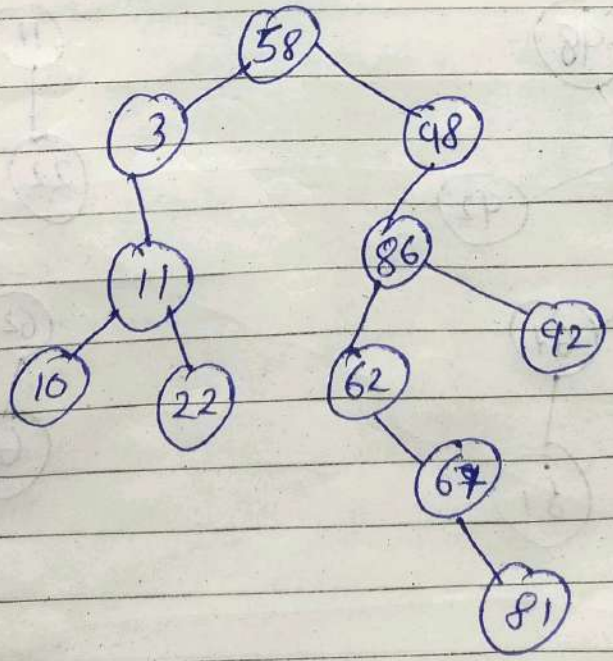


7) Splay - 50



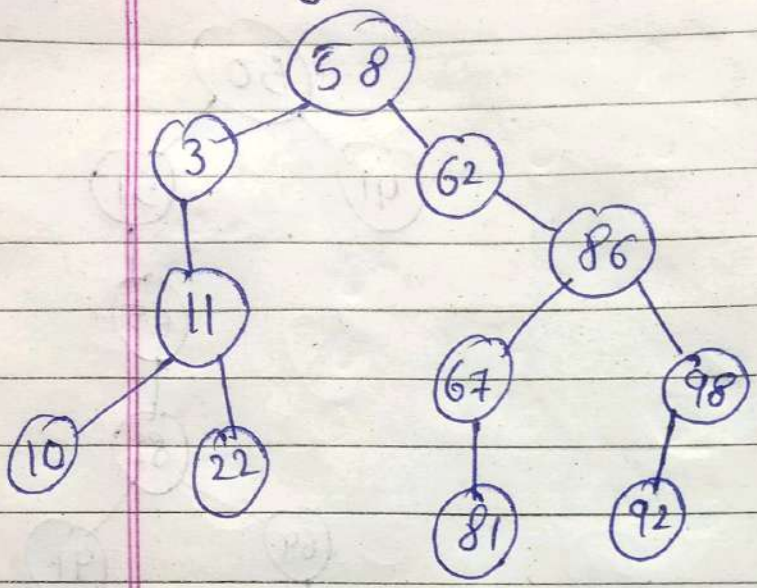


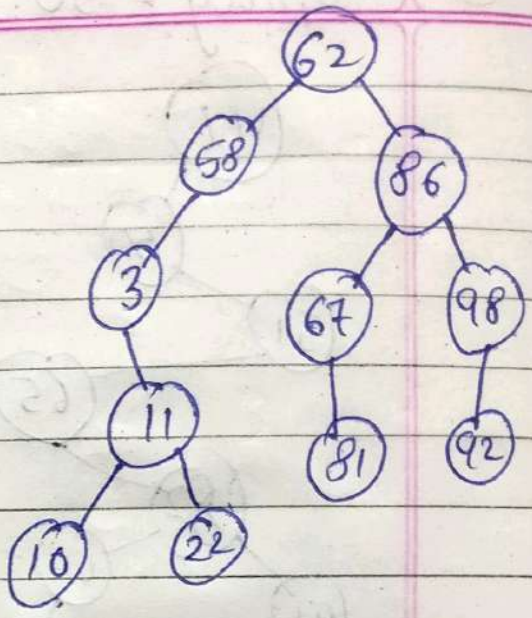
8) 58, 3, 98, 11, 86, 22, 62, 67, 81, 92, 10.



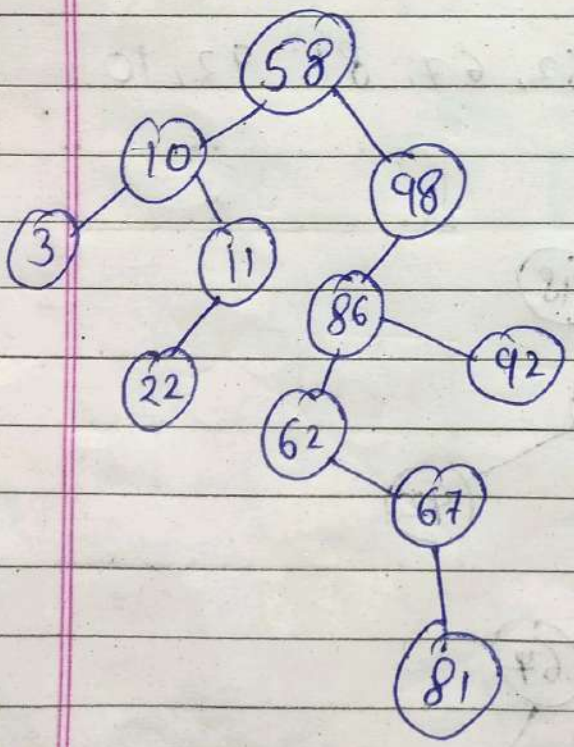
Date __/__/____
Page 26

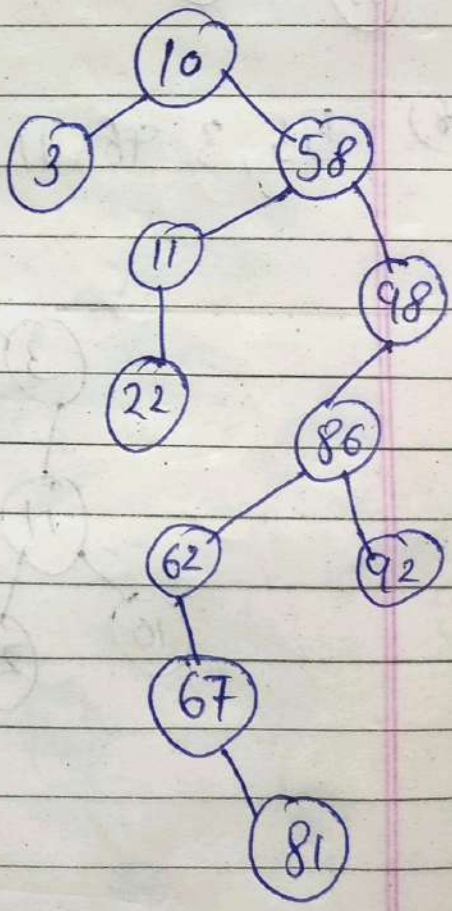
(8) Splay - 62 :-





Splay - 10 :-



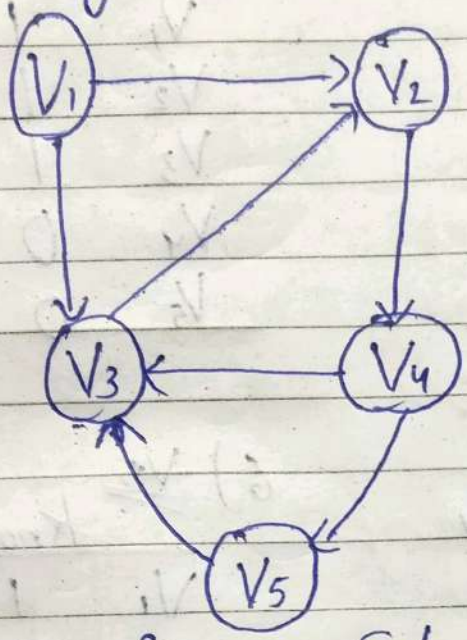


Date __/__/____
Page 27

Assignment - 2

Q.1. Unweighted Shortest Path.

1)



V_1 is Starting Node.

1) Initial State

	Known	dv	Pv
V_1	0	∞	0
V_2	0	∞	0
V_3	0	∞	0
V_4	0	∞	0
V_5	0	∞	0

2) V_1

	Know	dv	Pv
V_1	1	0	0
V_2	0	1	V_1
V_3	0	1	V_1
V_4	0	∞	0
V_5	0	∞	0

3) V_2

	Known	dv	Pv
V_1	1	0	0
V_2	1	1	V_1
V_3	0	1	V_1
V_4	0	2	V_2
V_5	0	∞	0

4) V_3

	Known	dv	Pv
V_1	1	0	0
V_2	1	1	V_1
V_3	1	1	V_1
V_4	0	2	V_2
V_5	0	∞	0

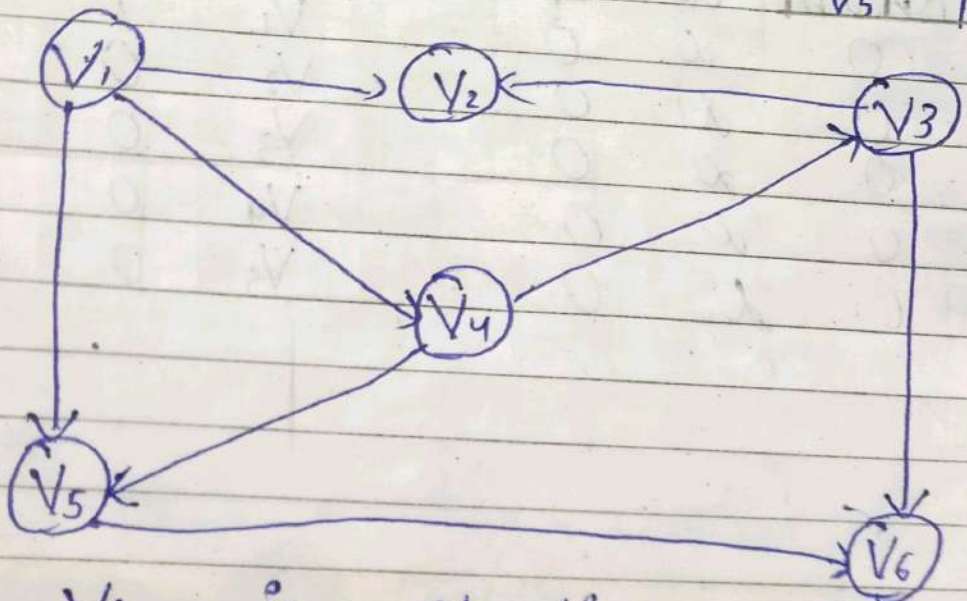
5) V_4

	Known	dv	Pv
V_1	1	0	0
V_2	1	1	V_1
V_3	1	1	V_1
V_4	1	2	V_2
V_5	0	3	V_4

6) V_5

	Known	dv	Pv
V_1	1	0	0
V_2	1	1	V_1
V_3	1	1	V_1
V_4	1	2	V_2
V_5	1	3	V_4

(2)



V_1 is Starting note:-

Date __/__/____
Page 29

1) Initial State

	Known	dv	Pv
V_1	0	∞	0
V_2	0	∞	0
V_3	0	∞	0
V_4	0	∞	0
V_5	0	∞	0
V_6	0	∞	0

2) V_1

	Known	dv	Pv
V_1	1	0	0
V_2	0	1	V_1
V_3	0	∞	0
V_4	0	1	V_1
V_5	0	1	V_1
V_6	0	∞	0

3) V_2

	Known	dv	Pv
V_1	1	0	0
V_2	1	1	V_1
V_3	0	∞	0
V_4	0	1	V_1
V_5	0	1	V_1
V_6	0	∞	0

4) V_4

	Known	dv	Pv
V_1	1	0	0
V_2	1	1	V_1
V_3	0	2	V_4
V_4	1	1	V_1
V_5	0	1	V_1
V_6	0	∞	0

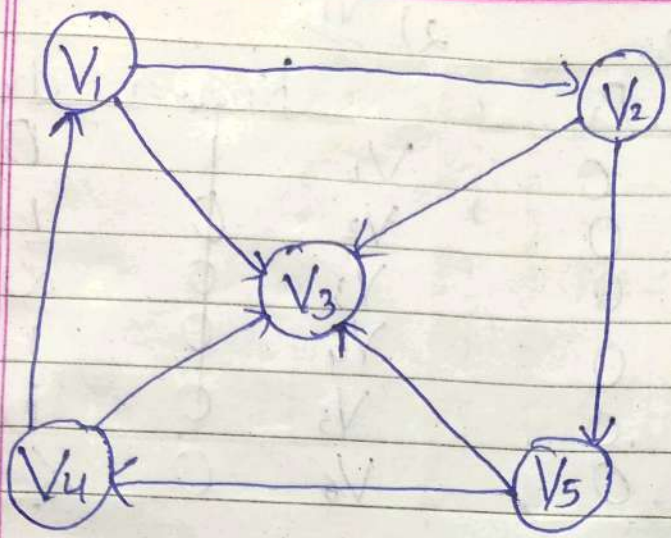
5) V_5

	Known	dv	Pv
V_1	1	0	0
V_2	1	1	V_1
V_3	0	2	V_4
V_4	1	1	V_1
V_5	1	1	V_1
V_6	0	2	V_5

6) V_3

	Known	dv	Pv
V_1	1	0	0
V_2	1	1	V_1
V_3	1	2	V_4
V_4	1	1	V_1
V_5	1	1	V_1
V_6	0	2	V_5

3)



V_1 is Starting Node.

(1) Initial state.

	Known	dv	Pv
V_1	0	∞	0
V_2	0	∞	0
V_3	0	∞	0
V_4	0	∞	0
V_5	0	∞	0

2) V_1

	Known	dv	Pv
V_1	1	0	0
V_2	0	1	V_1
V_3	0	1	V_1
V_4	0	1	V_1
V_5	0	∞	0

3) V_2

	Known	dv	Pv
V_1	1	0	0
V_2	1	1	V_1
V_3	0	1	V_1
V_4	0	1	V_1
V_5	0	2	V_2

4) V_3

	Known	dv	Pv
V_1	1	0	0
V_2	1	1	V_1
V_3	1	1	V_1
V_4	0	1	V_1
V_5	0	2	V_2

5) V_4

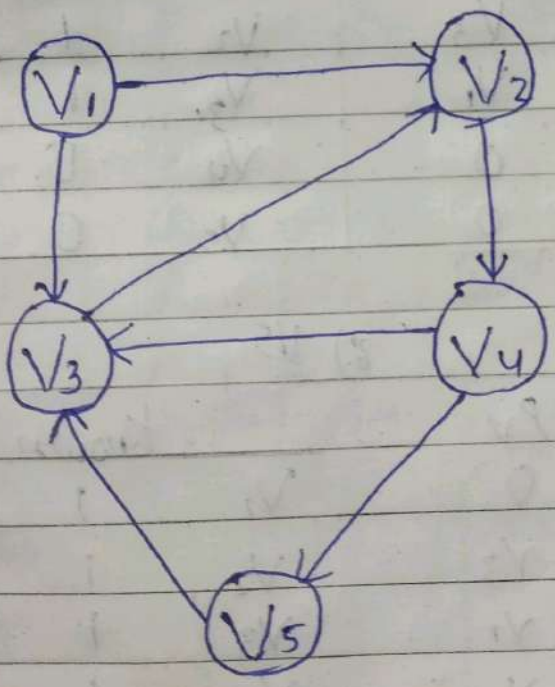
	Known	dv	Pv
V_1	1	0	0
V_2	1	1	V_1
V_3	1	1	V_2
V_4	1	1	V_1
V_5	0	2	V_2

6) V_5

	Known	dv	Pv
V_1	1	0	0
V_2	1	1	V_1
V_3	1	1	V_1
V_4	1	1	V_1
V_5	1	2	V_2

Q.2 weighted & shortest Path.

1)



V_1 is starting node.

1) Initial state

	Known	dv	P_V
V_1	0	0	0
V_2	0	∞	0
V_3	0	∞	0
V_4	0	∞	0
V_5	0	∞	0

2) V_1

	Known	dv	P_V
V_1	1	0	0
V_2	0	6	V_1
V_3	0	1	V_1
V_4	0	∞	0
V_5	0	∞	0

3) V_3

	Known	dv	P_V
V_1	1	0	0
V_2	0	3	V_3
V_3	1	1	V_1
V_4	0	∞	0
V_5	0	∞	0

4) V_2

	Known	dv	P_V
V_1	1	0	0
V_2	1	3	V_3
V_3	1	1	V_1
V_4	0	5	V_2
V_5	0	∞	0

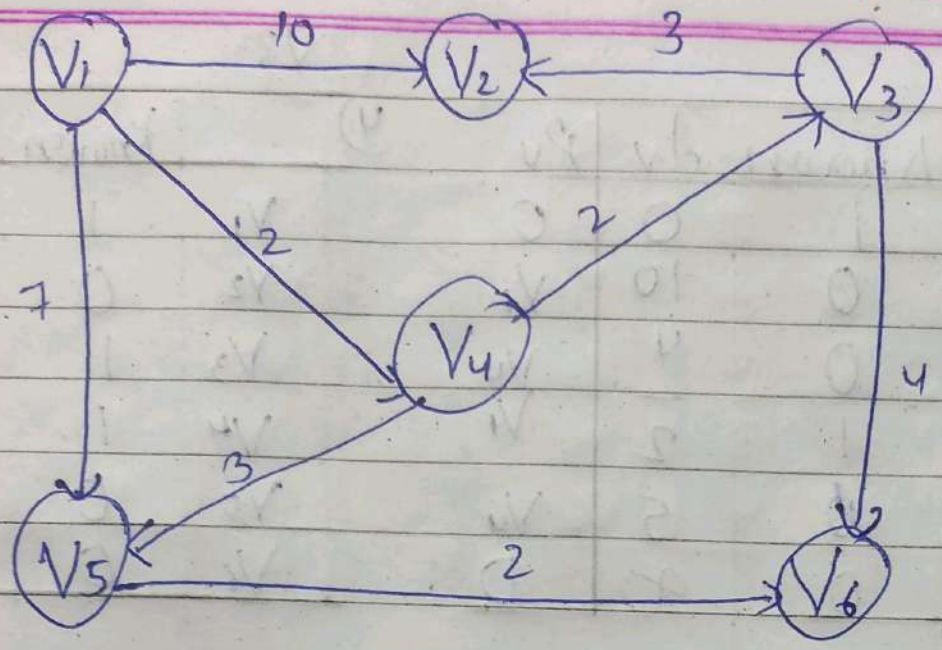
5) V_4

	Known	dv	P_V
V_1	1	0	0
V_2	1	3	V_3
V_3	1	1	V_1
V_4	1	5	V_2
V_5	0	7	V_4

6) V_5

	Known	dv	P_V
V_1	1	0	0
V_2	1	3	V_3
V_3	1	1	V_1
V_4	1	5	V_2
V_5	1	7	V_4

(2)



→ V_1 is Starting Node.

1) Initial State

	known	dv	Pv
V_1	0	0	0
V_2	0	∞	0
V_3	0	∞	0
V_4	0	∞	0
V_5	0	∞	0
V_6	0	∞	0

2) V_1

	known	dv	Pv
V_1	1	0	0
V_2	0	10	V_1
V_3	0	∞	0
V_4	0	2	V_1
V_5	0	7	V_1
V_6	0	∞	0

③. V_4

	Known	dv	Pv
V_1	1	0	0
V_2	0	10	V_1
V_3	0	4	V_4
V_4	1	2	V_1
V_5	0	5	V_4
V_6	0	9	0

④ V_3

	Known	dv	Pv
V_1	1	0	0
V_2	0	7	V_3
V_3	1	4	V_4
V_4	1	2	V_1
V_5	0	5	V_4
V_6	0	8	V_3

⑤ V_5

	Known	dv	Pv
V_1	1	0	0
V_2	0	7	V_3
V_3	1	4	V_4
V_4	1	2	V_1
V_5	1	5	V_4
V_6	0	7	V_5

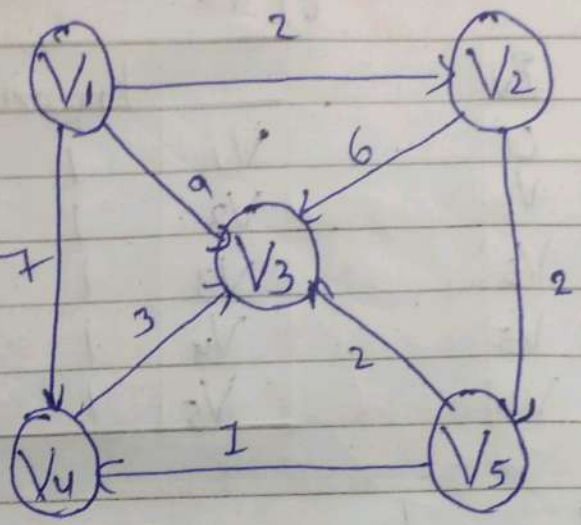
⑥ V_2

	Known	dv	Pv
V_1	1	0	0
V_2	1	7	V_3
V_3	1	4	V_4
V_4	1	2	V_1
V_5	1	5	V_4
V_6	0	7	V_5

⑦ V_6

	Known	dv	Pv
V_1	1	0	0
V_2	1	7	V_3
V_3	1	4	V_4
V_4	1	2	V_1
V_5	1	5	V_4
V_6	1	7	V_5

③



→ V_1 is starting Node

1) Initial state

	known	dv	Pv
V_1	0	0	0
V_2	0	∞	0
V_3	0	∞	0
V_4	0	∞	0
V_5	0	∞	0

2) V_1

	known	dv	Pv
V_1	1	0	0
V_2	0	2	V_1
V_3	0	9	V_1
V_4	0	7	V_1
V_5	0	∞	0

3) V_2

	known	dv	Pv
V_1	1	0	0
V_2	1	2	V_1
V_3	0	8	V_2
V_4	0	7	V_1
V_5	0	4	V_2

4) V_5

	known	dv	Pv
V_1	1	0	0
V_2	1	2	V_1
V_3	0	6	V_5
V_4	0	5	V_5
V_5	1	4	V_2

5) V_4

	Known	dv	Pv
V_1	1	0	0
V_2	1	2	V_1
V_3	0	6 / 5	V_5
V_4	1	5	V_5
V_5	1	4	V_2

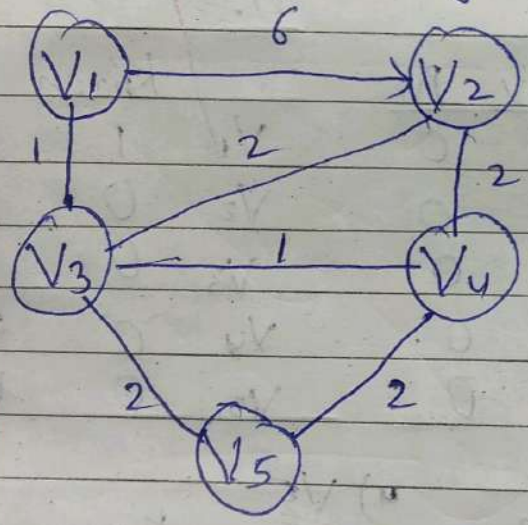
6) V_3

	Known	dv	Pv
V_1	1	0	0
V_2	1	2	V_1
V_3	1	6	V_5
V_4	1	5	V_5
V_5	1	4	V_2

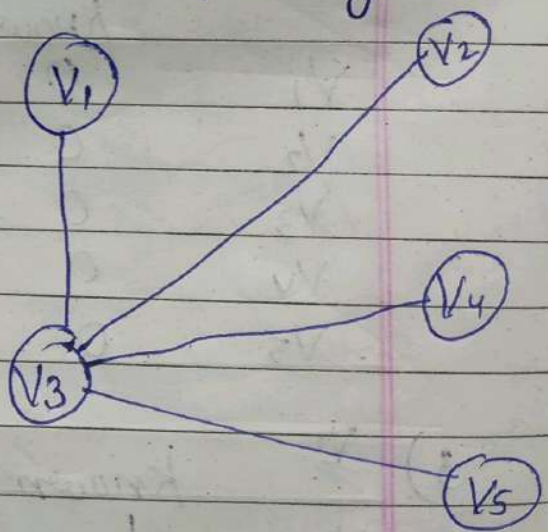
Q3 minimum Spanning Tree :-

* Kruskal's Algorithm :-

(i)

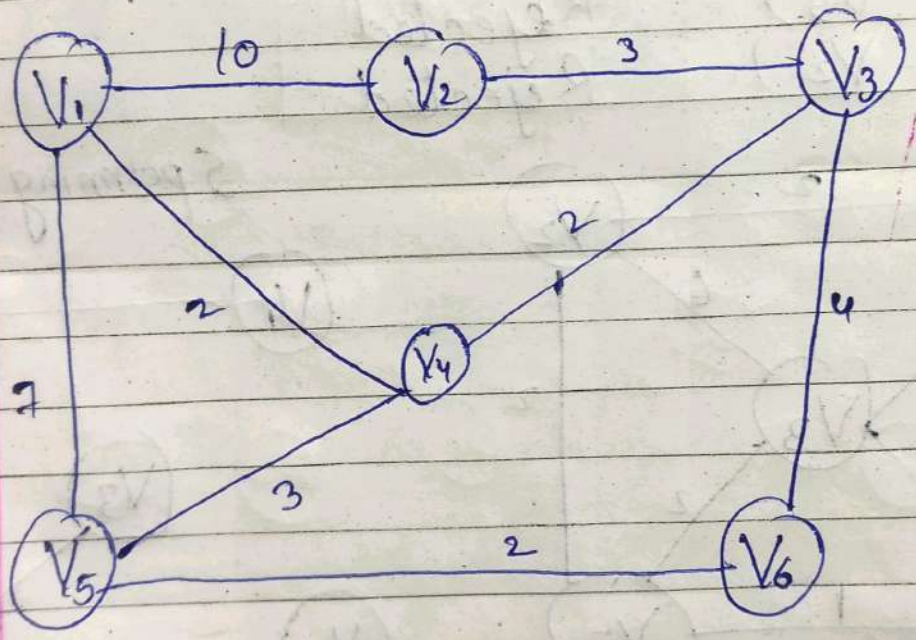


Spanning Tree

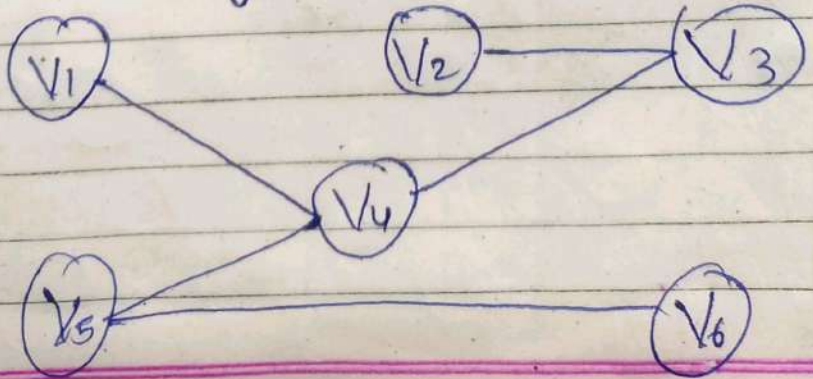


Weight		
1	(V_1, V_3)	Accepted
1	(V_3, V_4)	Accepted
2	(V_2, V_3)	Accepted
2	(V_2, V_4)	Rejected
2	(V_3, V_5)	Accepted
2	(V_4, V_5)	Rejected
6	(V_1, V_2)	Rejected.

2)



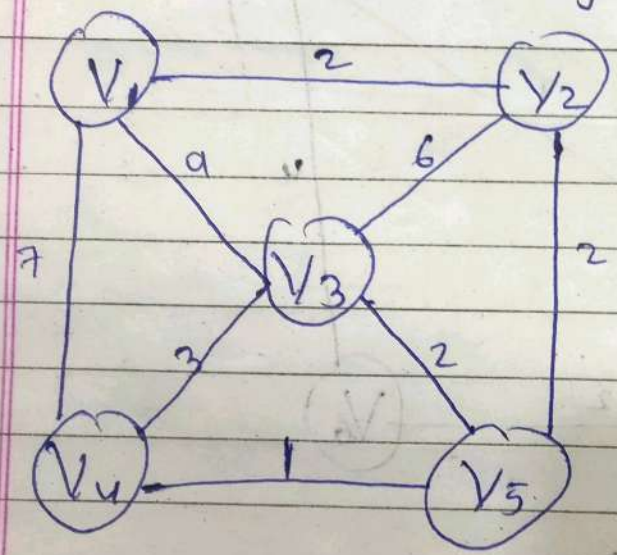
Spanning tree



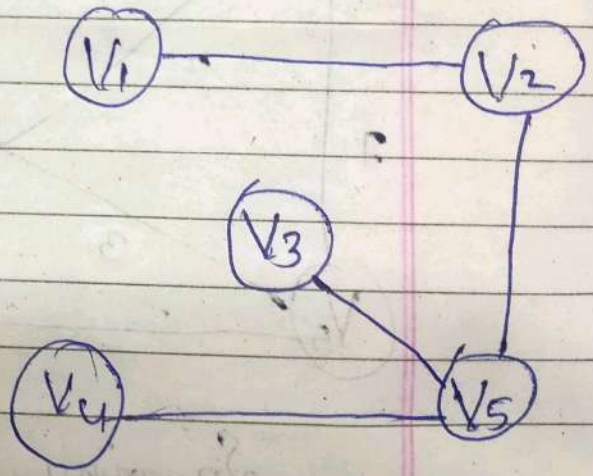
* Kruskal's

2	(V_1, V_4)	Accepted
2	(V_5, V_8)	Accepted
2	(V_3, V_4)	Accepted
3	(V_2, V_3)	Accepted
3	(V_4, V_5)	Accepted
4	(V_3, V_6)	Rejected
7	(V_1, V_5)	Rejected
10	(V_1, V_2)	Rejected

3)



Spanning tree

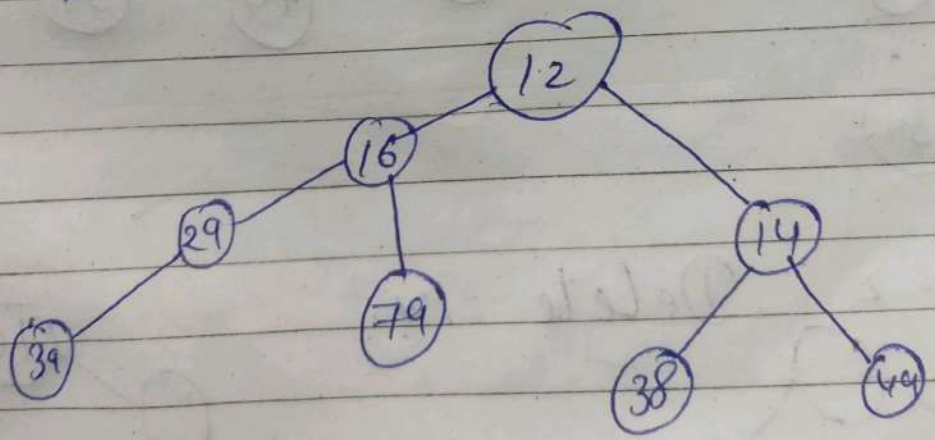


* Kruskals

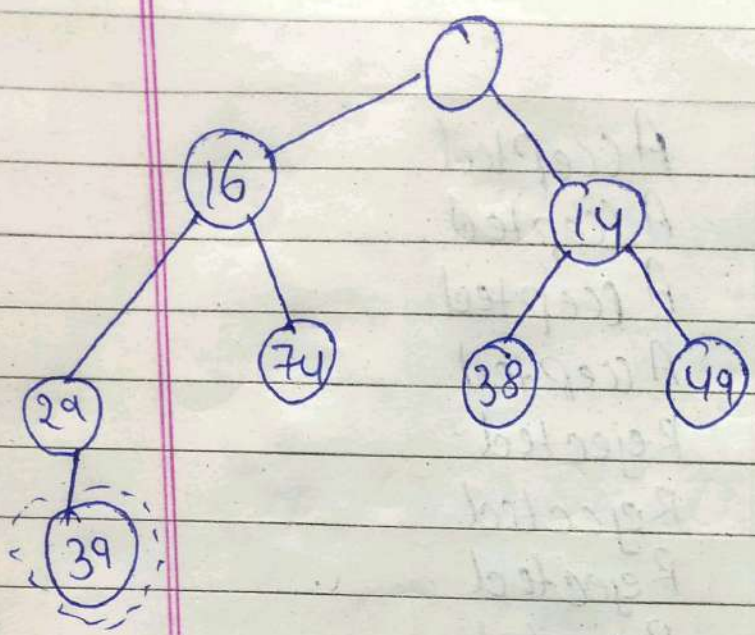
1	(V_4, V_5)	Accepted
2	(V_1, V_2)	Accepted
2	(V_2, V_5)	Accepted
2	(V_3, V_5)	Accepted
3	(V_3, V_4)	Rejected
6	(V_2, V_3)	Rejected
7	(V_1, V_4)	Rejected
9	(V_1, V_3)	Rejected.

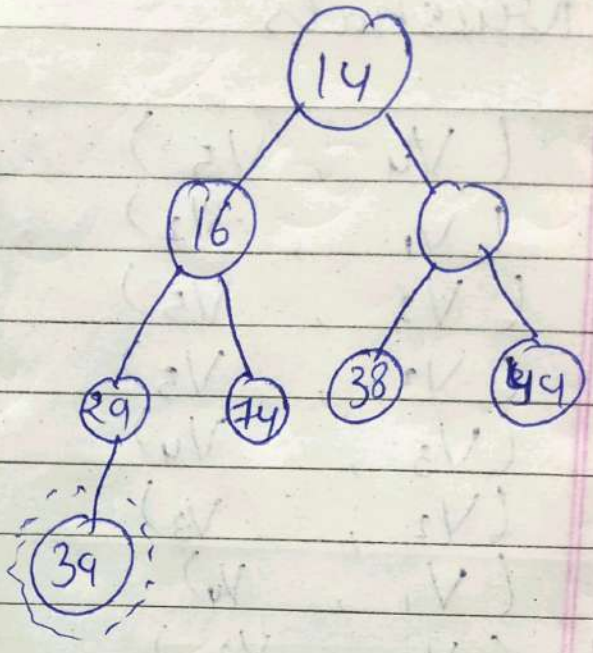
① Heap Sort.

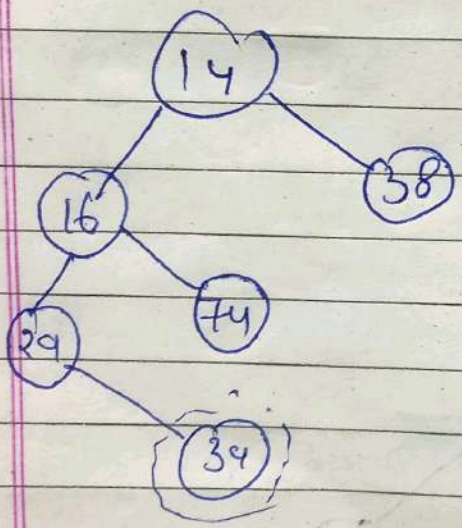
Delete min.

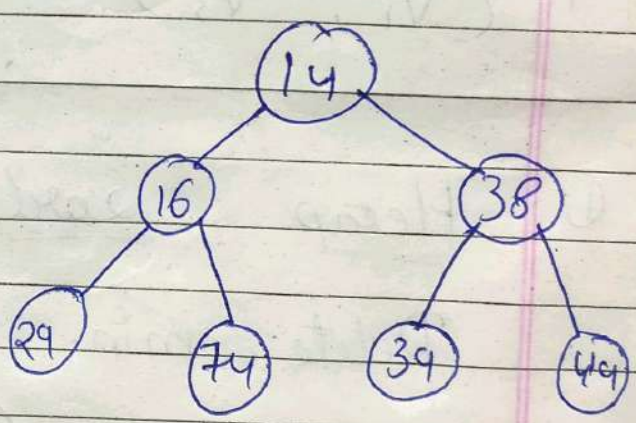


Step - 1 - Delete - 12.

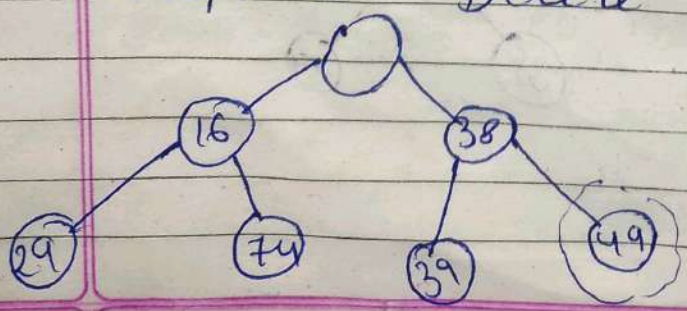


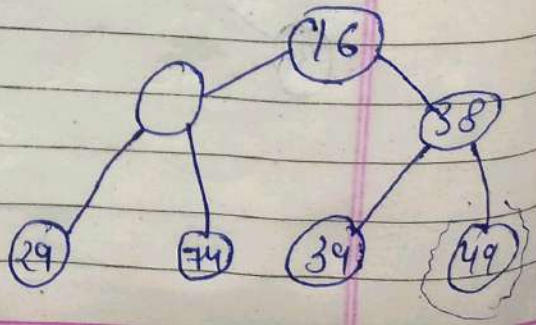


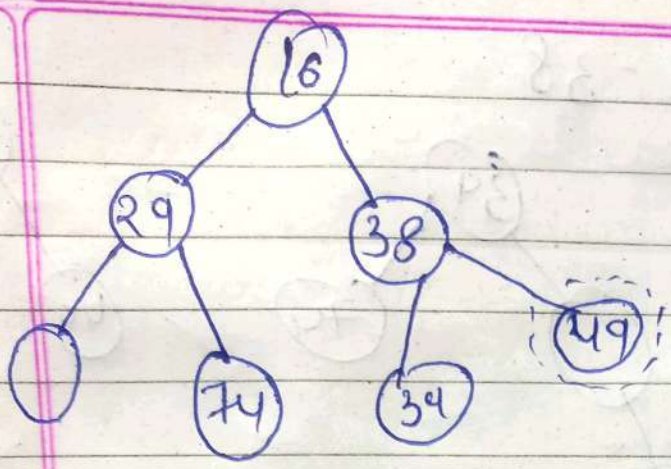


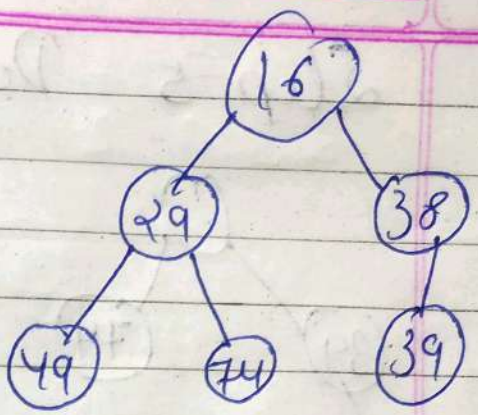


Step - 2 Delete - 14.

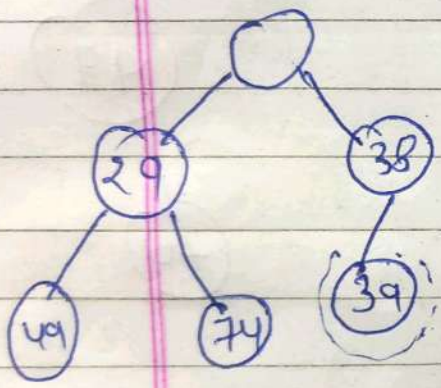


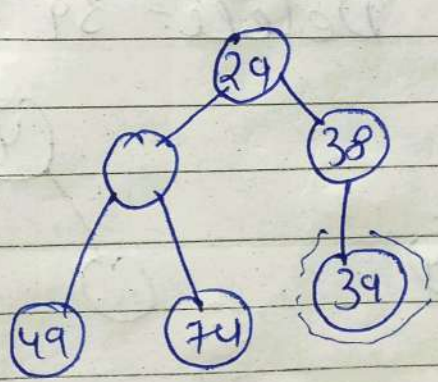


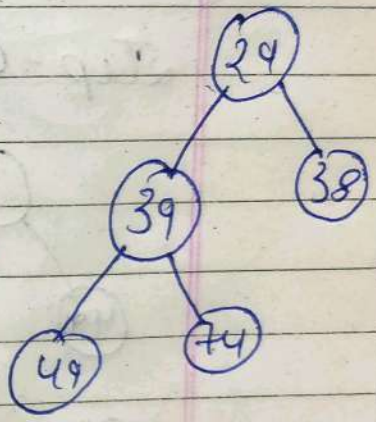




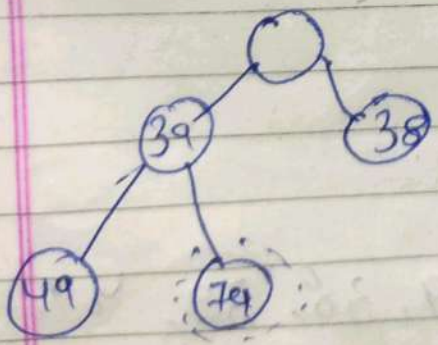
Step - 3 - Delete -16.

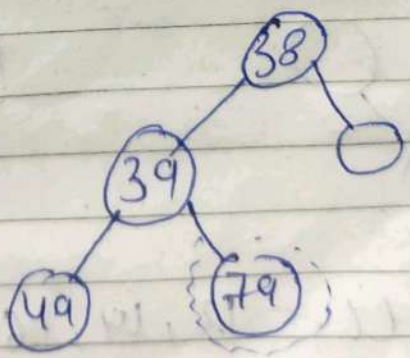


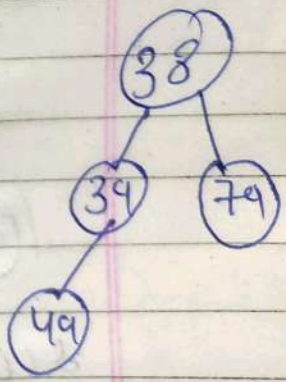




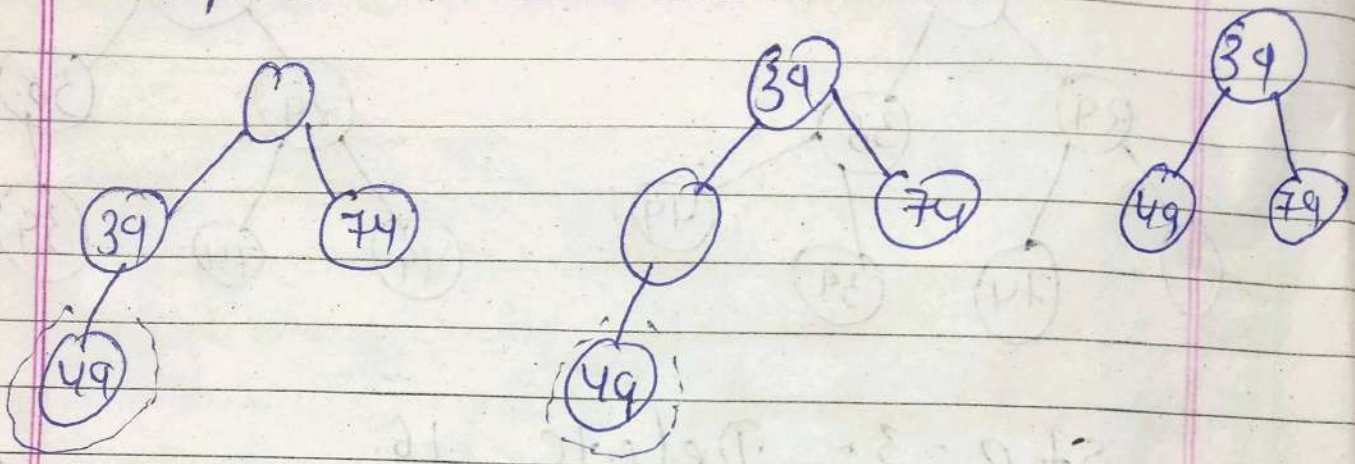
Step - 4 - Delete -29.



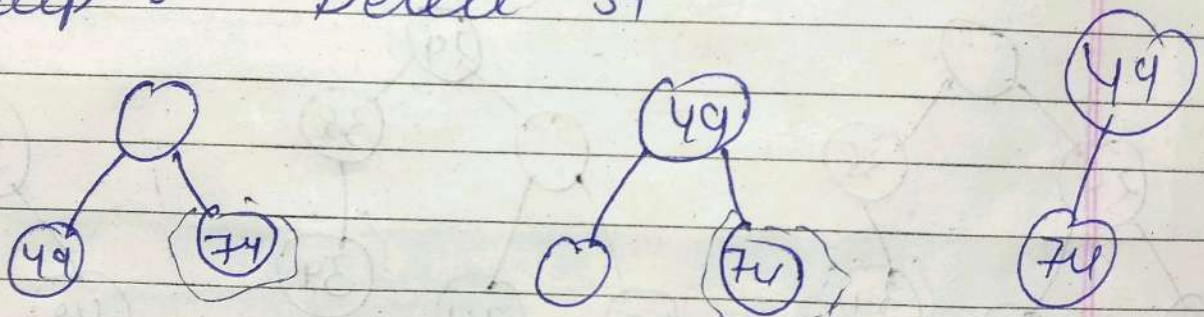




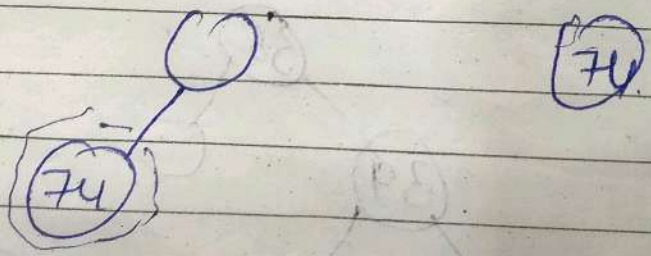
Step-5 - Delete - 38



Step-6 - Delete - 39

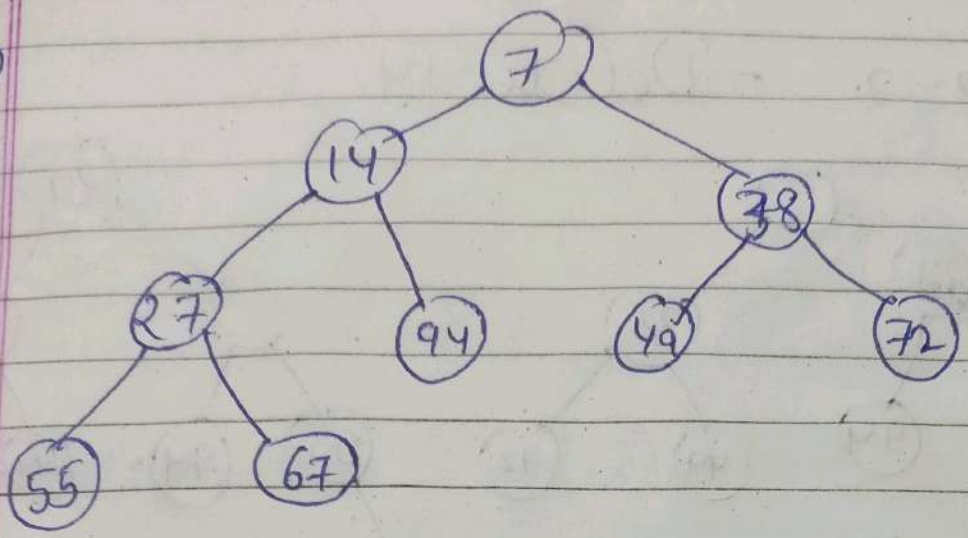


Step-7 - Delete - 49.

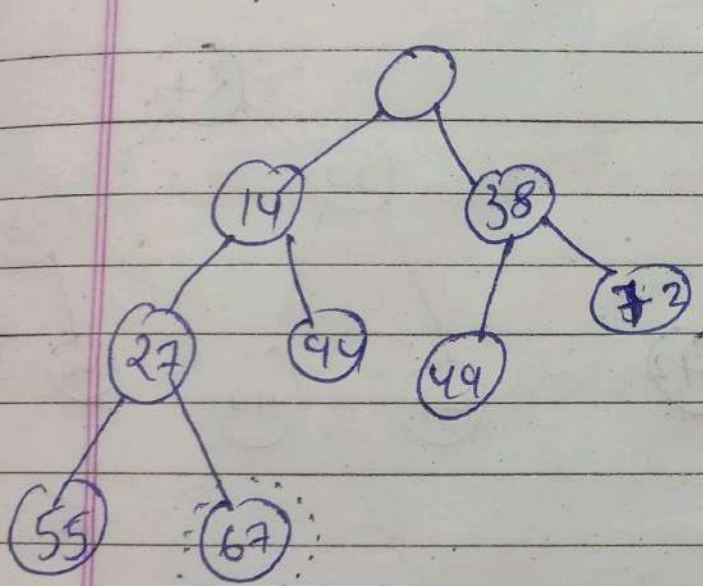


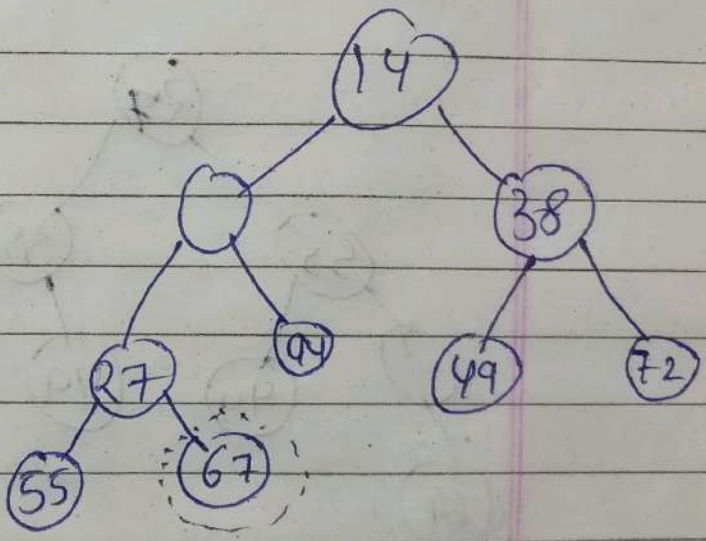
Result = 12, 14, 16, 29, 38, 39, 49, 74.

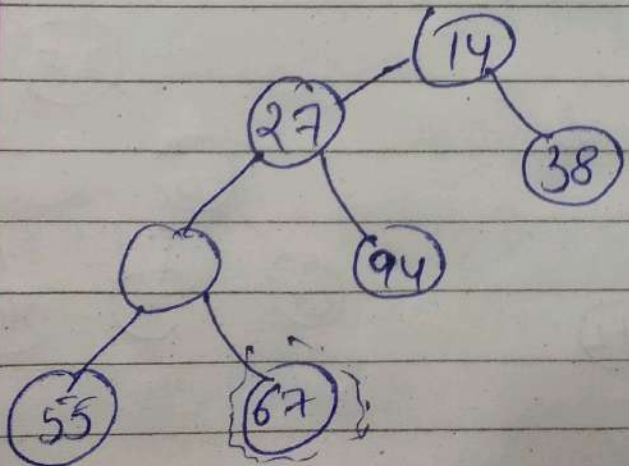
(2)

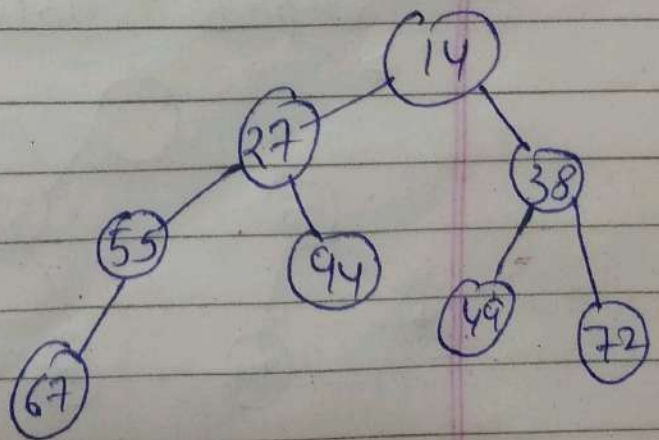


Step - 1 - Delete 7

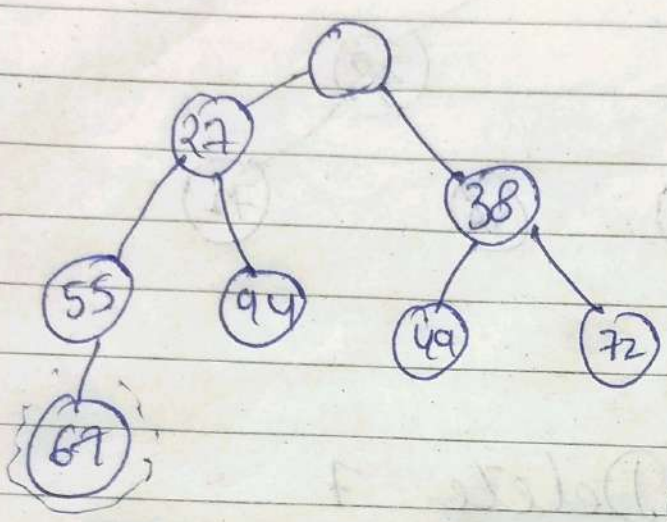


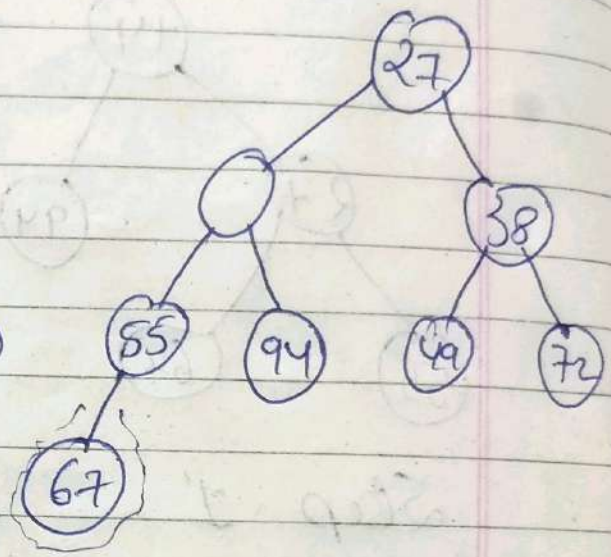


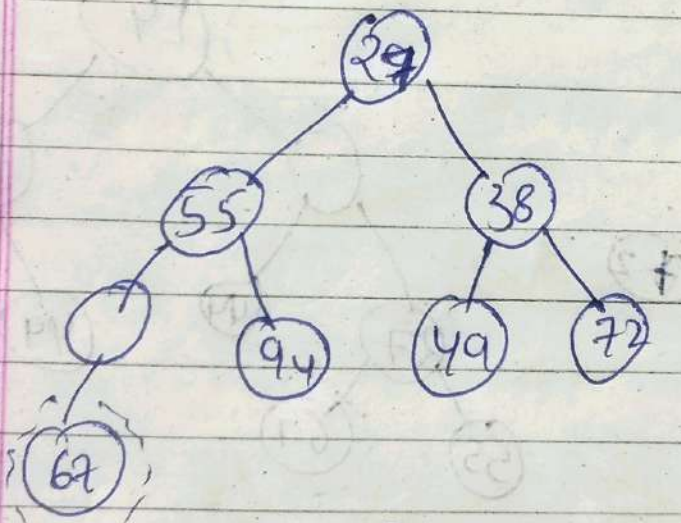


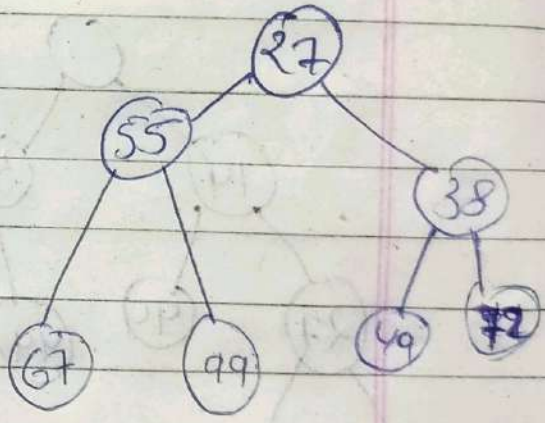


Step-2 - Delete-14.

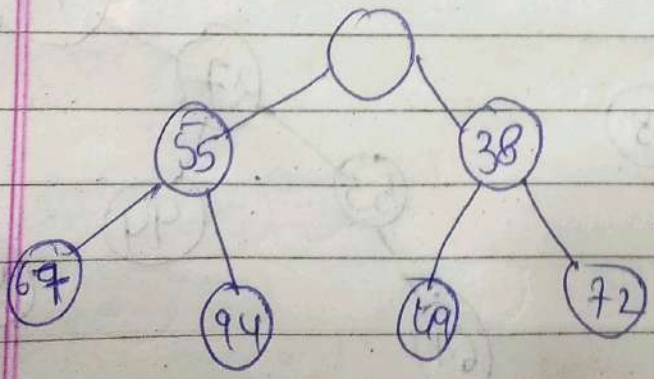


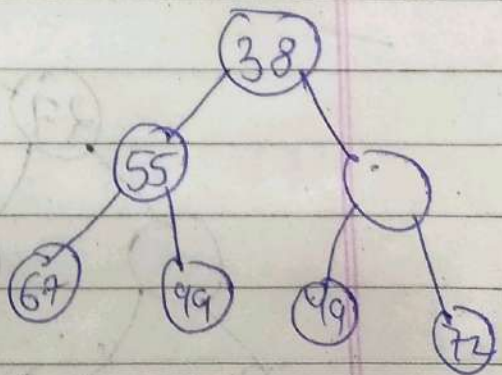


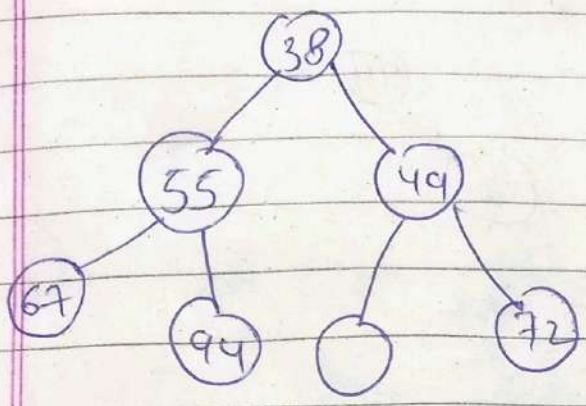


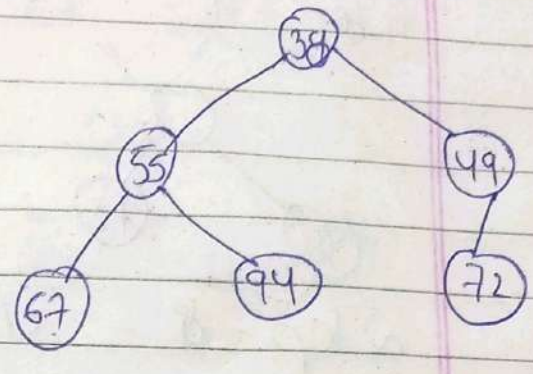


Step-3. Delete-27

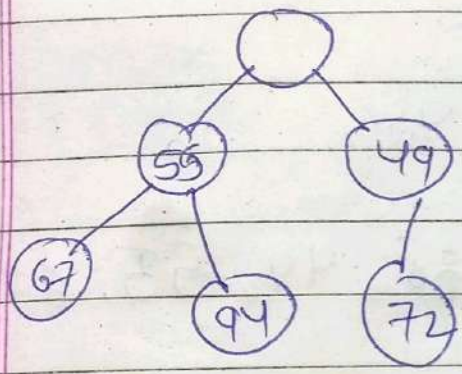


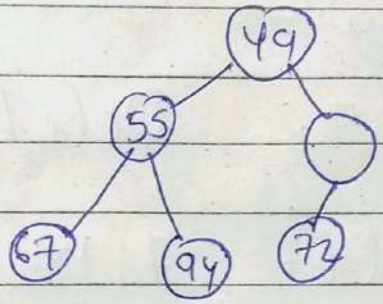


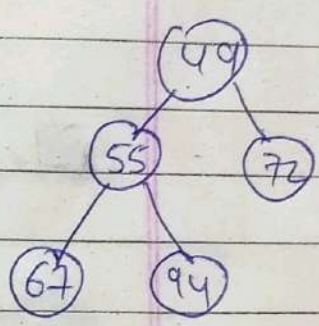




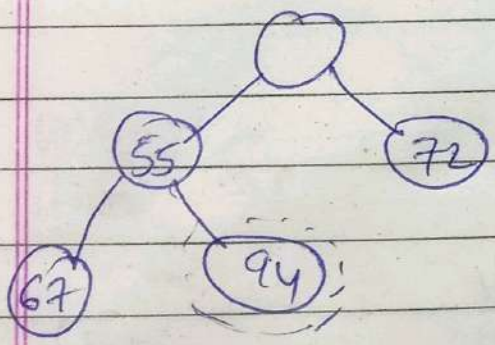
Step - 4 - Delete - 38

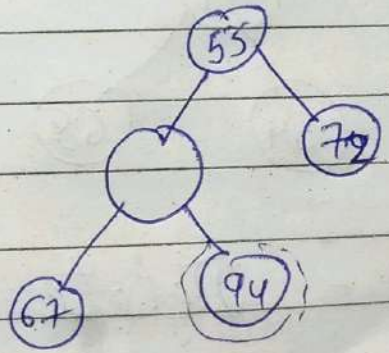


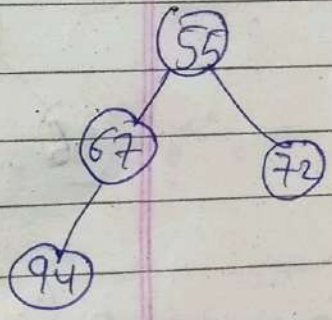




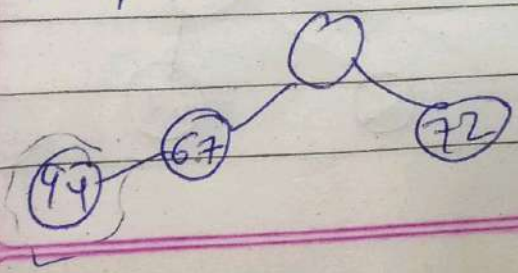
Step - 5 - Delete 49

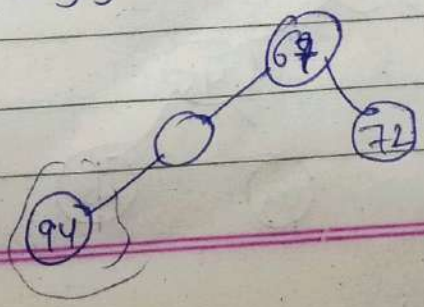


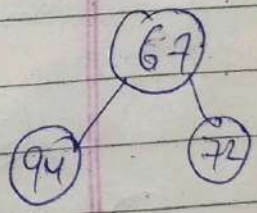




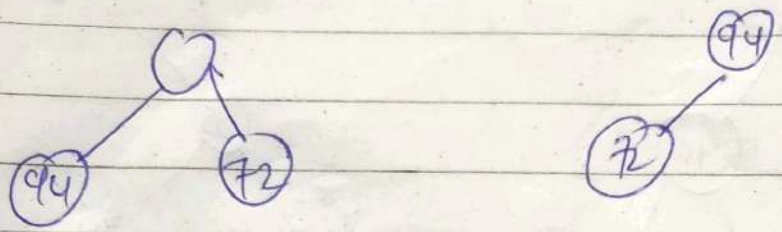
Step - 6 - Delete - 55







Step - 7 - Delete - 67

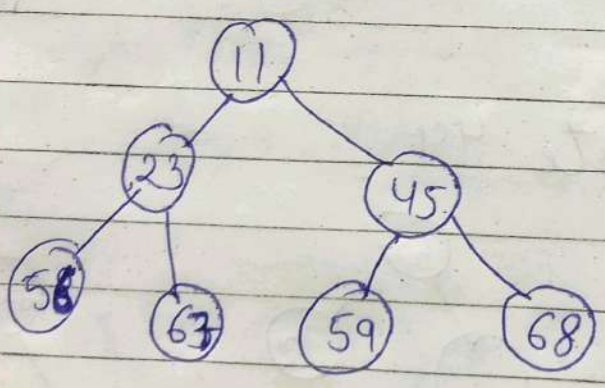


Step - 8 - delete - 72

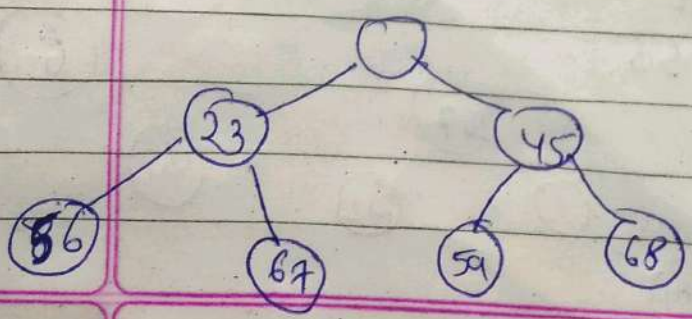
Step - 9. Delete - 94.

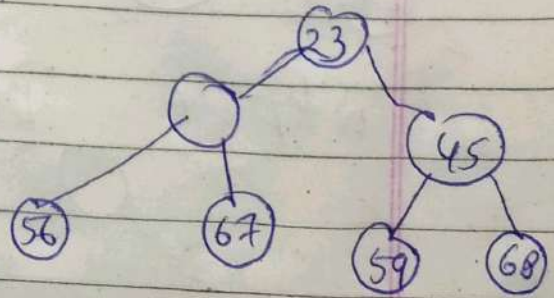
result = 7, 14, 21, 38, 49, 55, 67, 72, 94

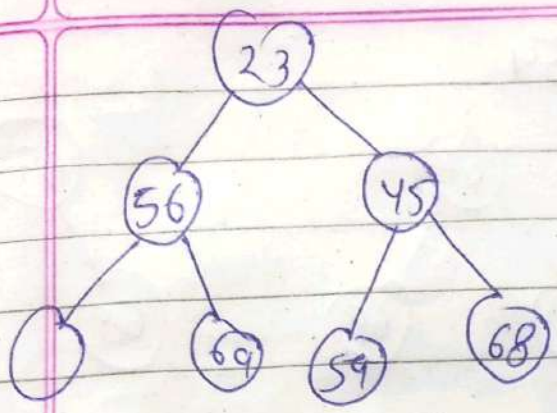
(3)

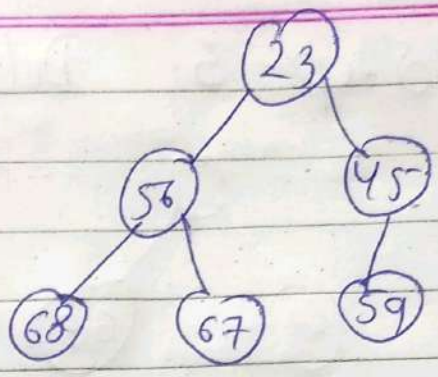


Step - 1 - Delete - 11.

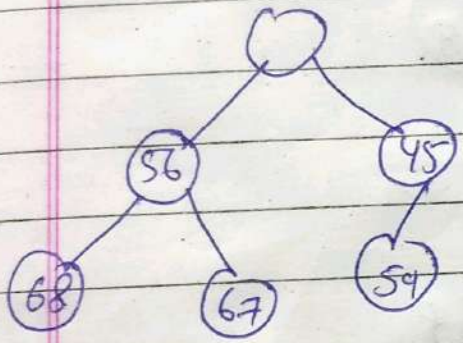


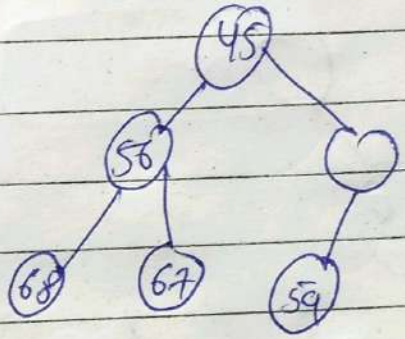


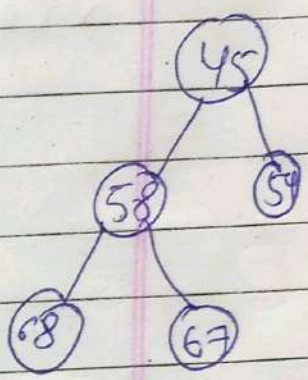




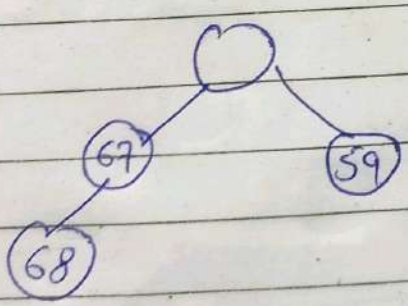
Step - 2 - Delete - 23.

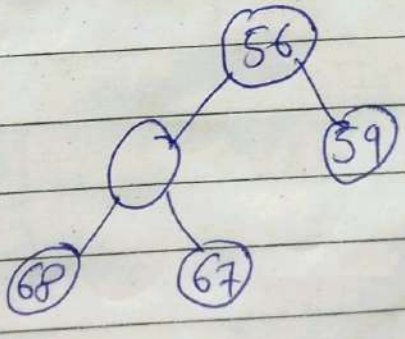


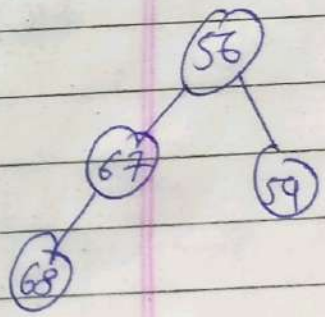




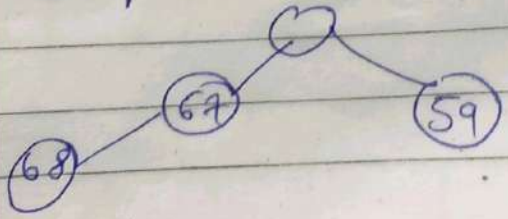
Step - 3 - Delete - 45

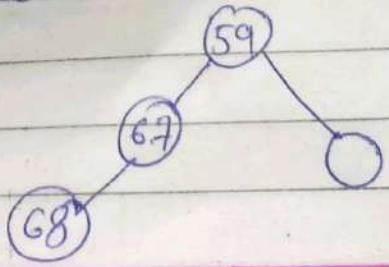


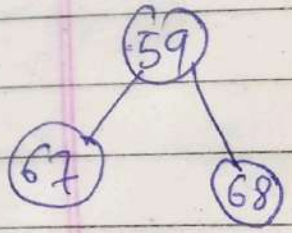




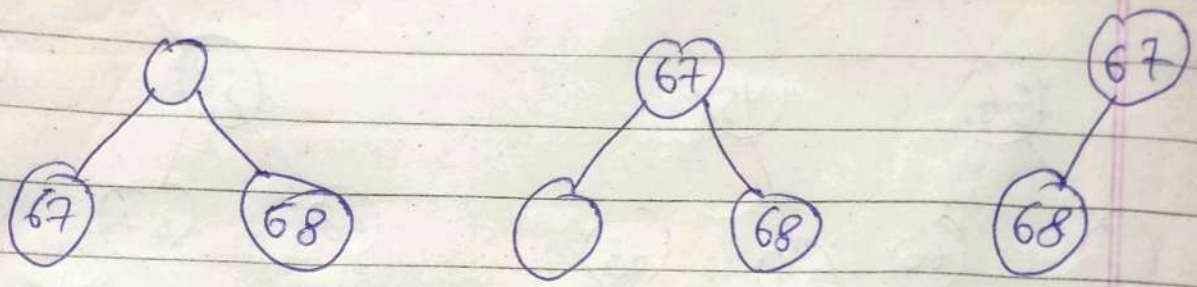
Step - 4. - Delete - 56.



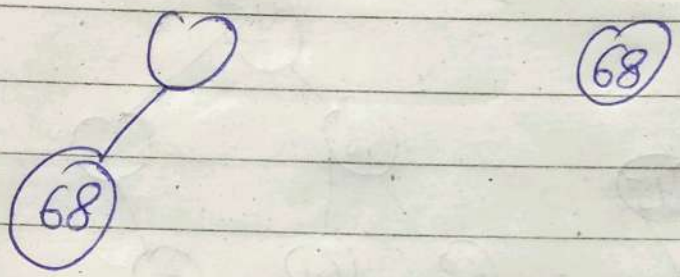




Step - 5. Delete - 54



Step - 6. Delete - 67



Step - 7 delete - 68

result - 11, 23, 45, 56, 59, 67, 68

Date / /
Page 49

Assignment -3.

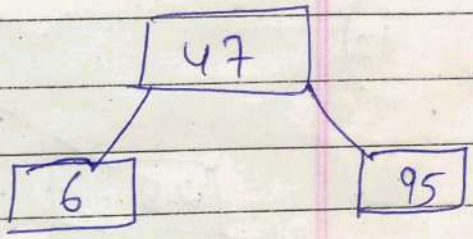
* B - Tree :→ order - 3.

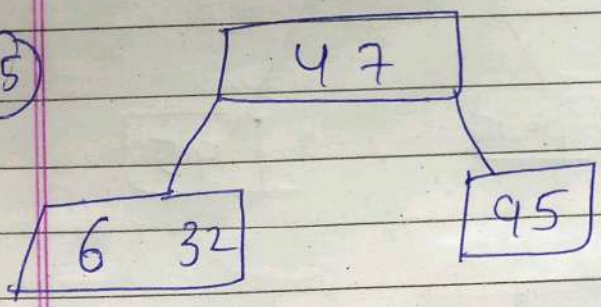
6, 95, 47, 32, 28, 75, 87, 35, 51, 16, 59, 89, 97, 99.

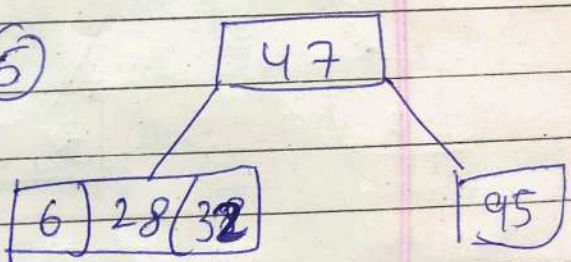
1) [6]

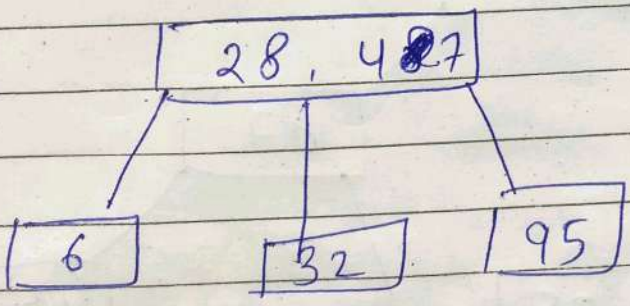
2) [6 95]

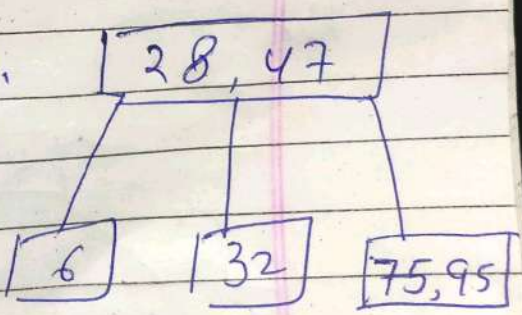
3) [6 95 47]

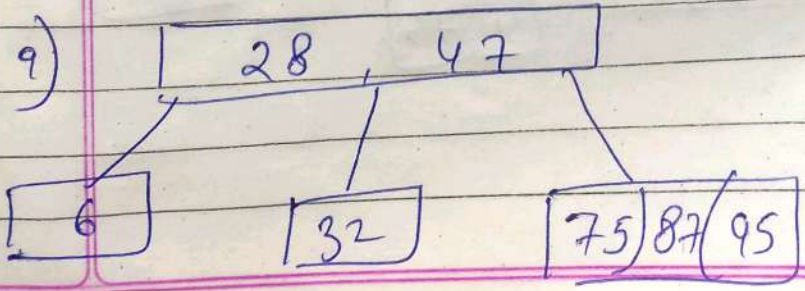
4)


5)


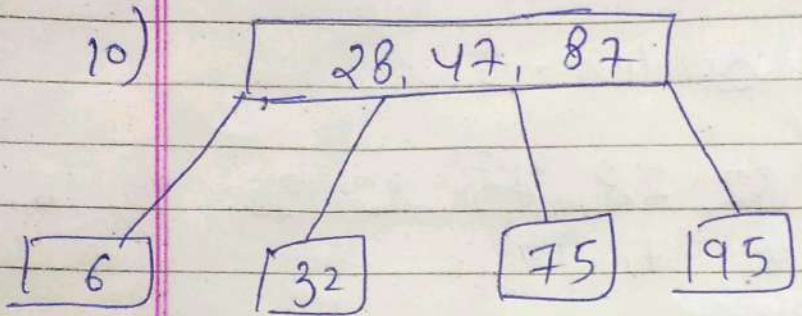
5)


7)


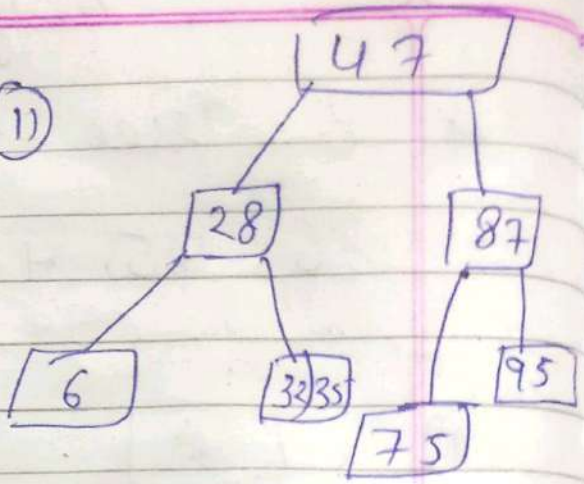
8.


9)


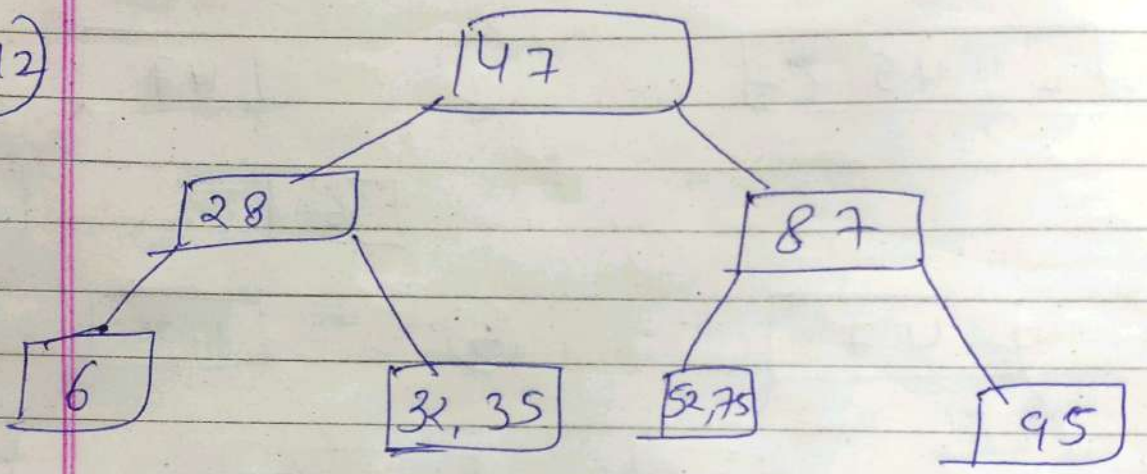
10)


28, 47, 87
6
32
75
95

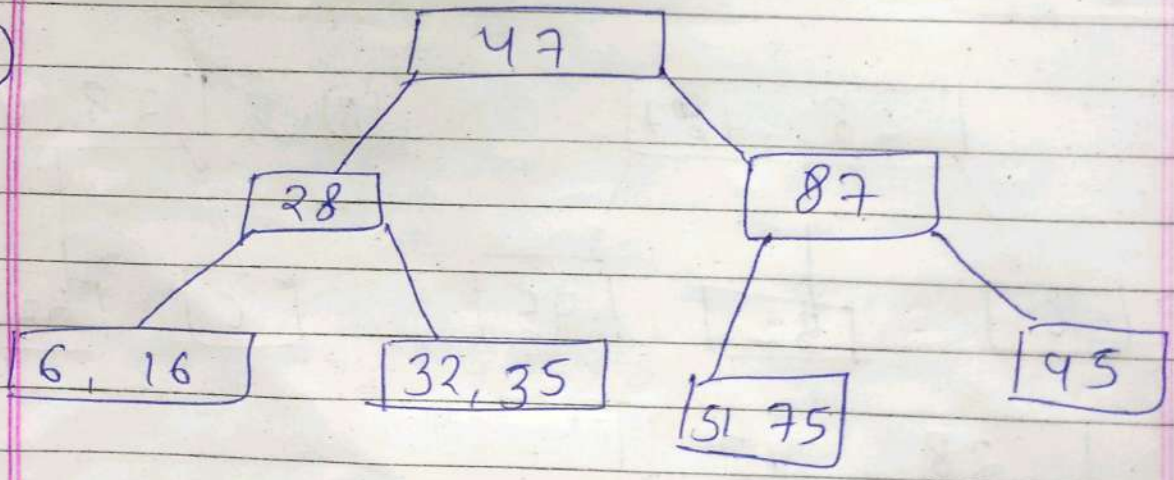
11)


47
28
87
6
32
35
95
75

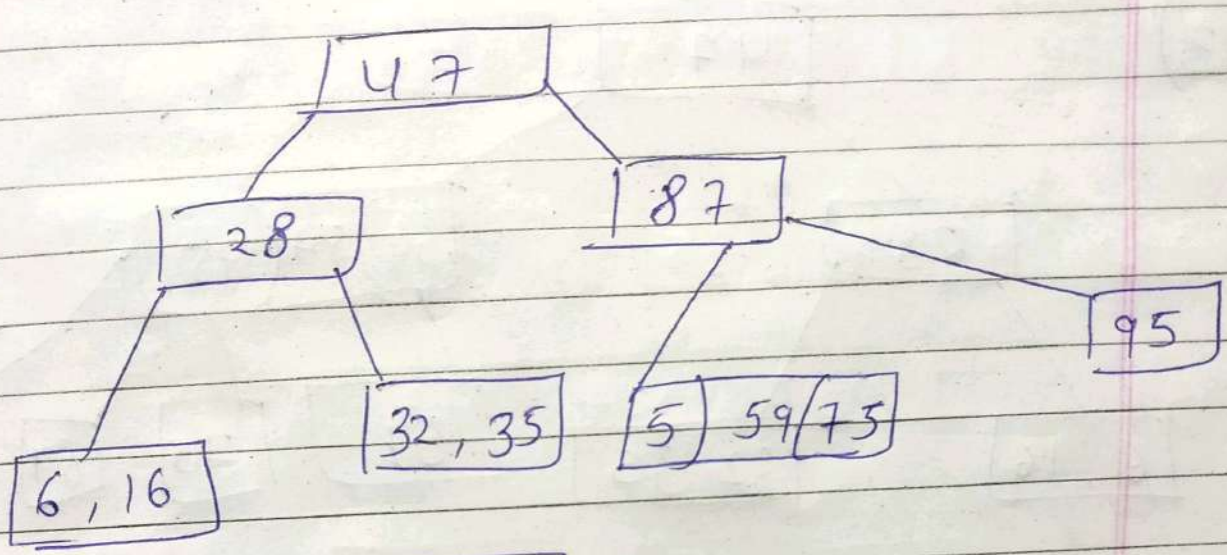
12)


47
28
87
6
32, 35
52, 75
95

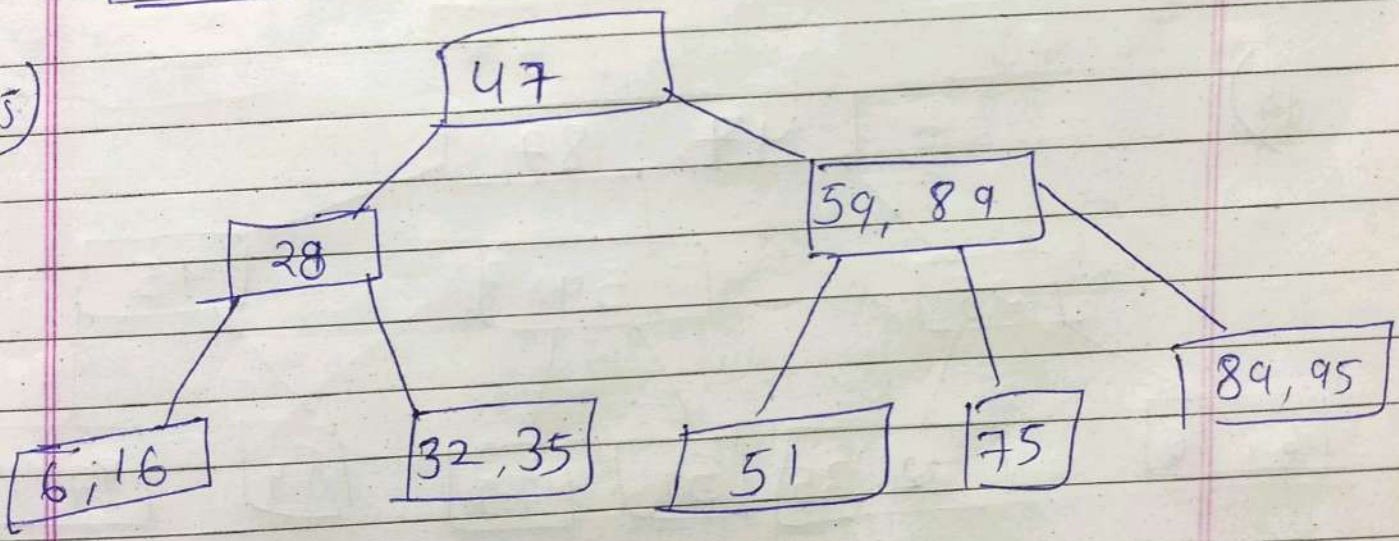
13)


47
28
87
6, 16
32, 35
51 75
95

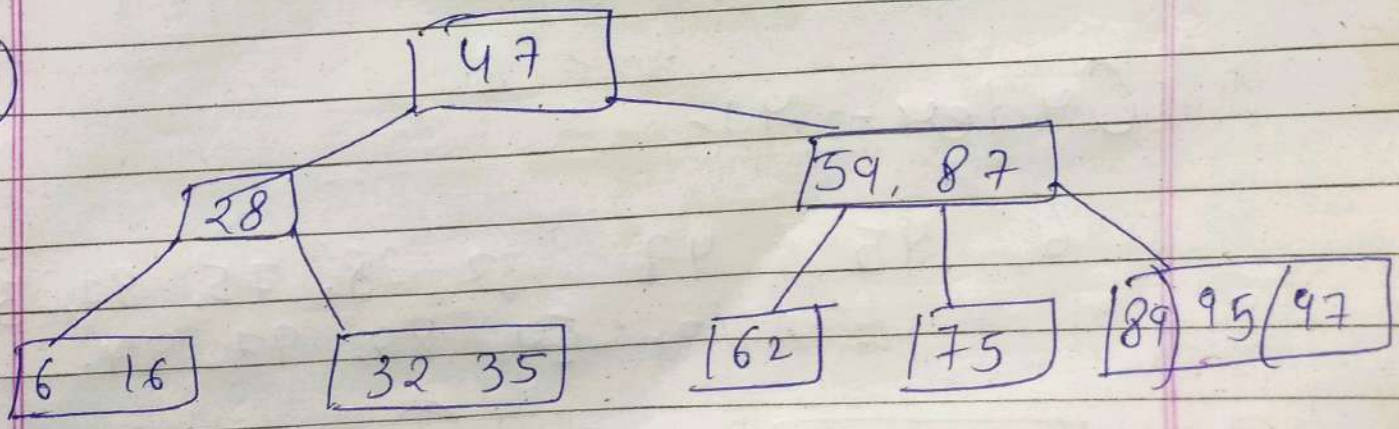
14)


47
28
87
95
6, 16
32, 35
5
59
75

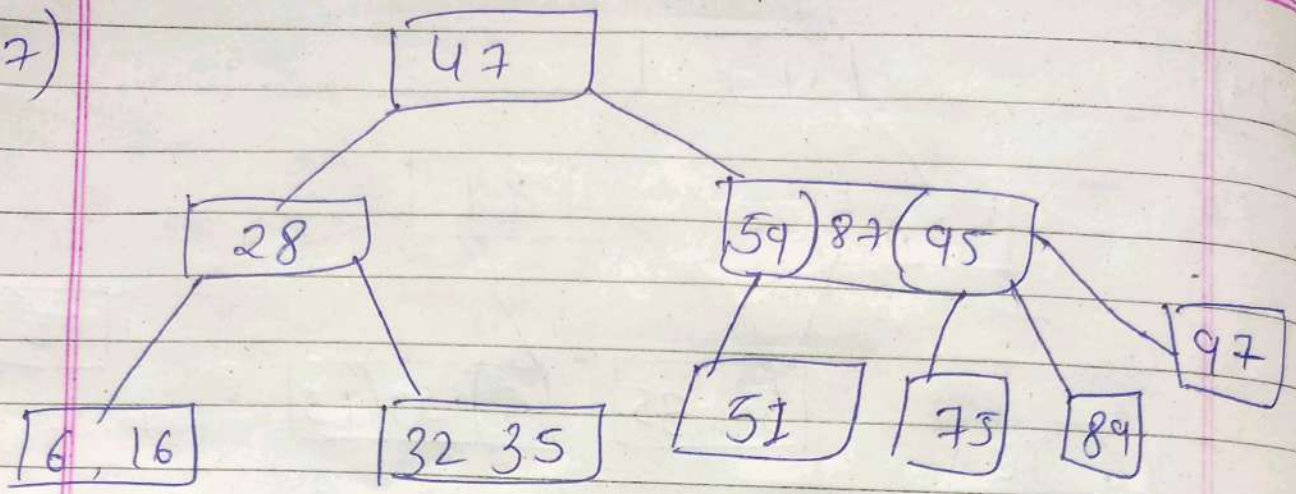
15)


47
28
59, 89
6, 16
32, 35
51
75
89, 95

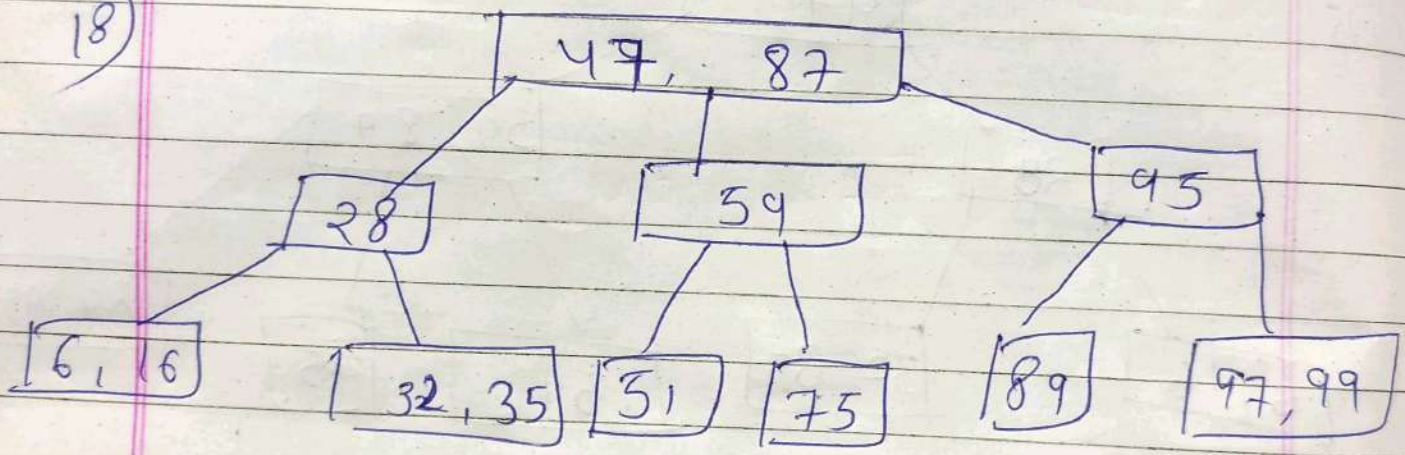
16)


47
28
59, 87
6 16
32 35
62
75
89
95
97

17)



18)



Order - 4 :-

6, 95, 47, 32, 28, 75, 87, 38, 51, 16, 59, 84, 97, 99.

(1) 6

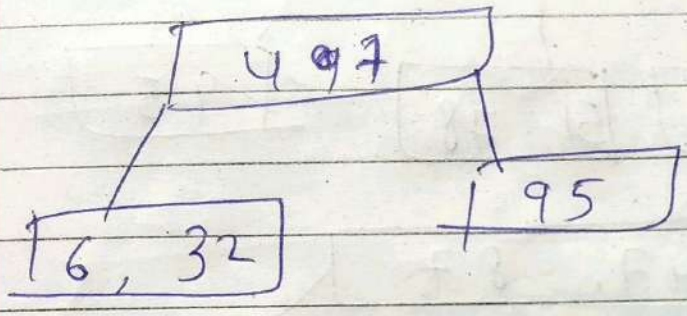
(2) 6, 95

Date / /
Page

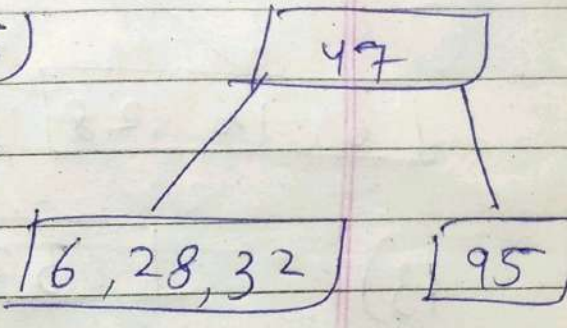
③ [6, 47 95]

④ [6, 32, 47, 95]

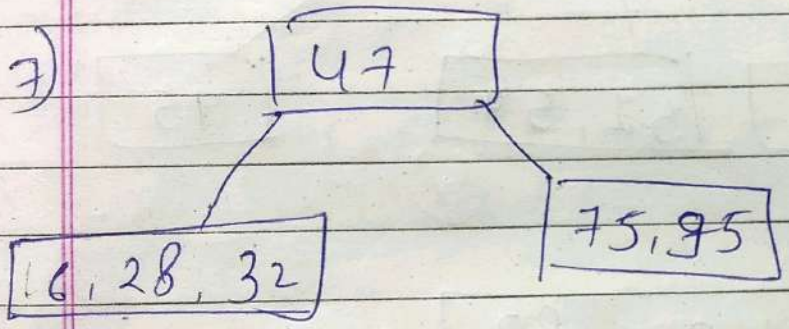
⑤



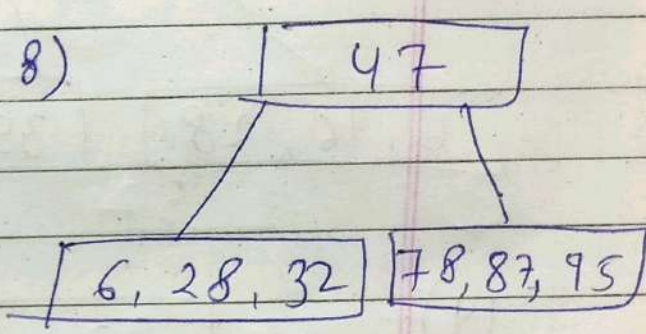
⑥



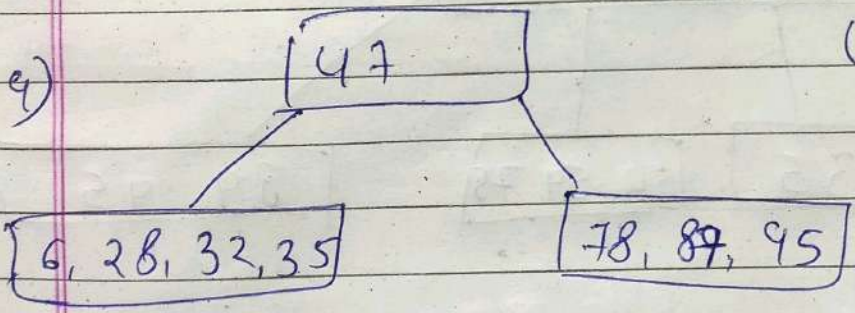
⑦



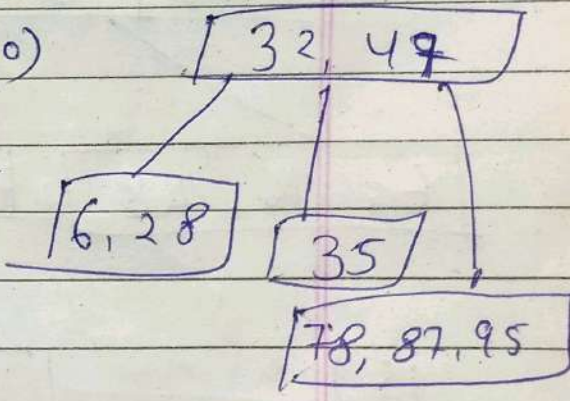
⑧



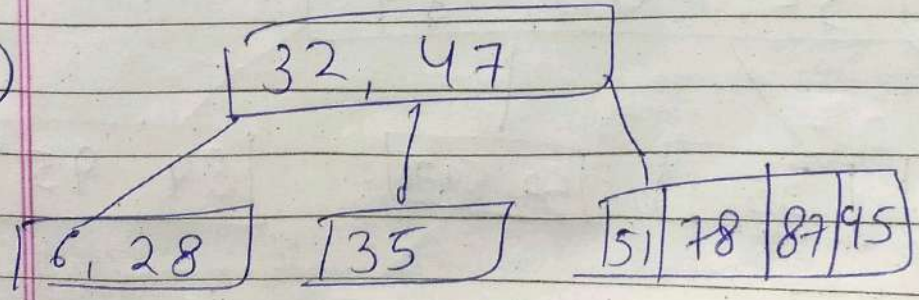
⑨



⑩



⑪

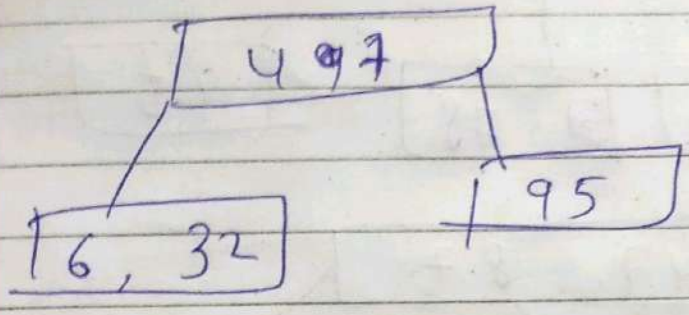


Date / /
Page

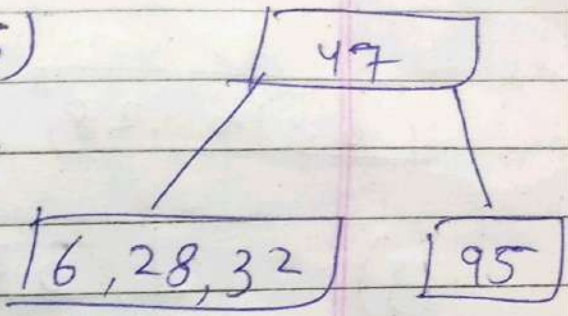
3) 6, 47 95

4) 6, 32, 47, 95

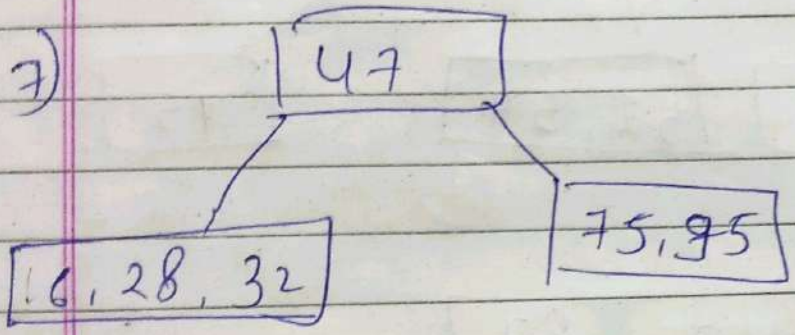
5)



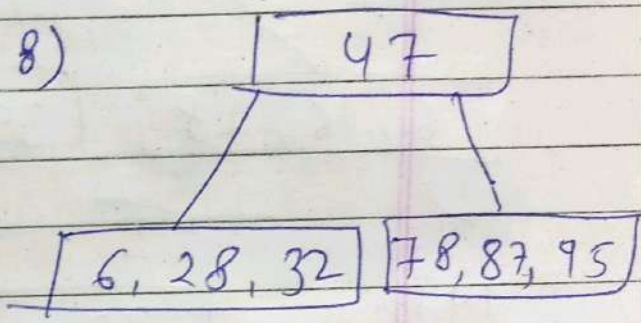
6)



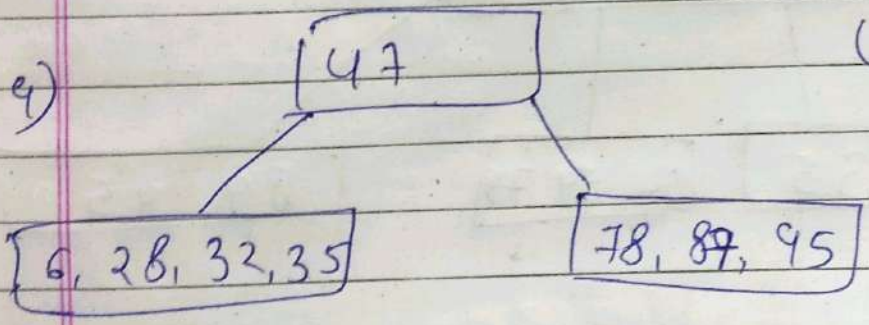
7)



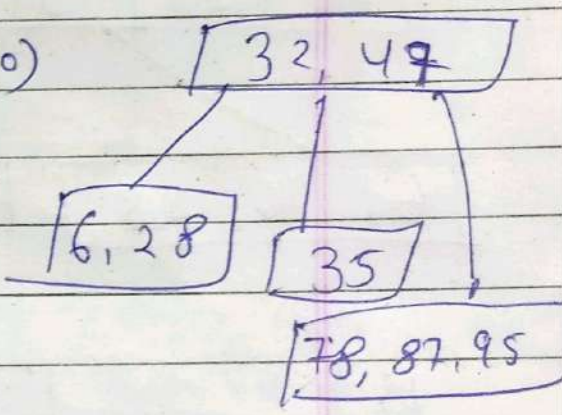
8)



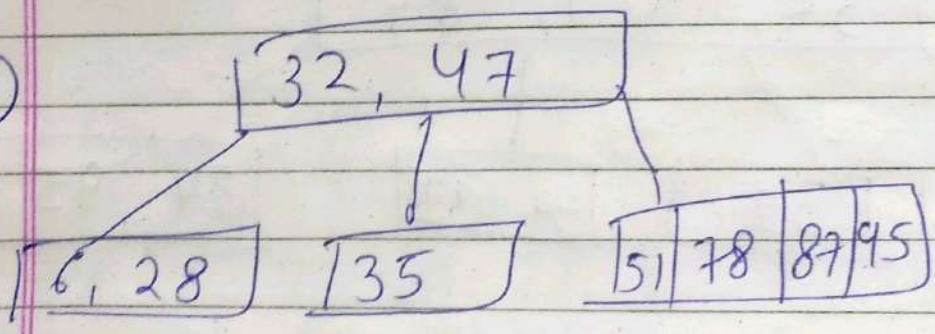
9)



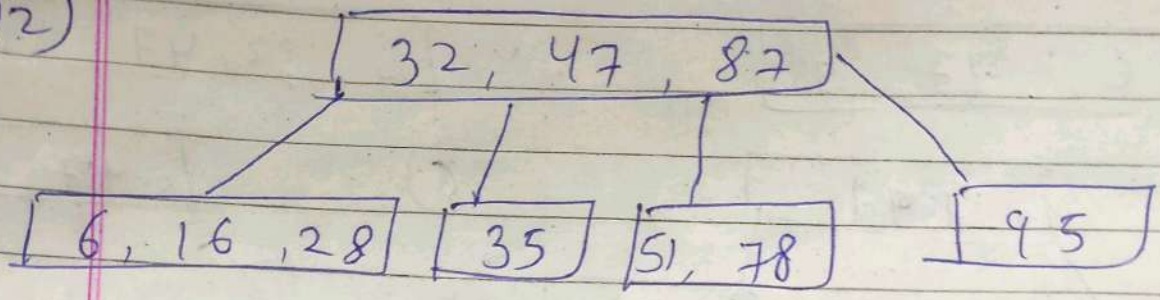
(10)



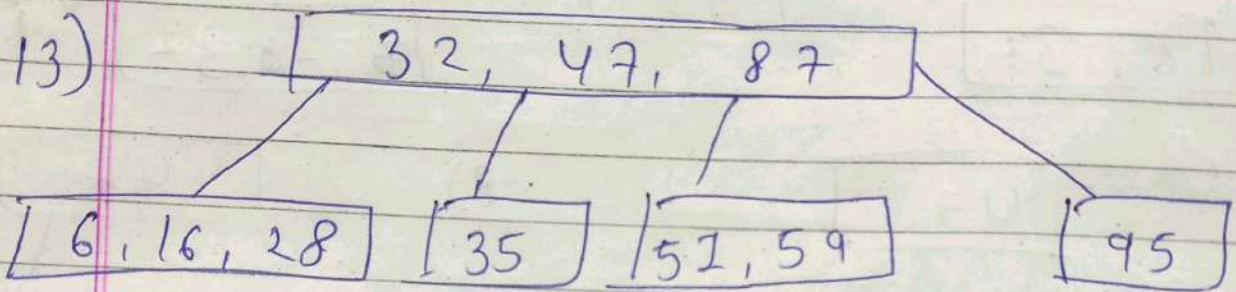
11)



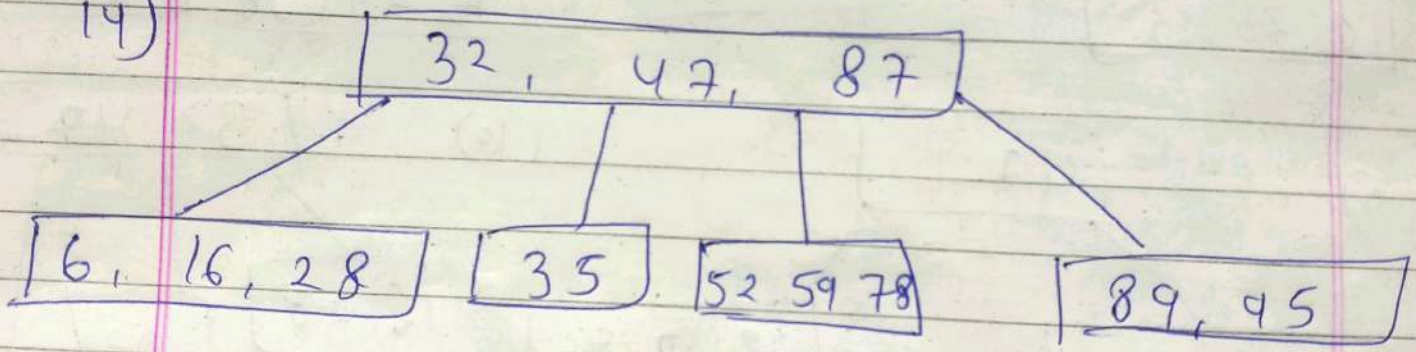
12)


32, 47, 87
6, 16, 28
35
51, 78
95

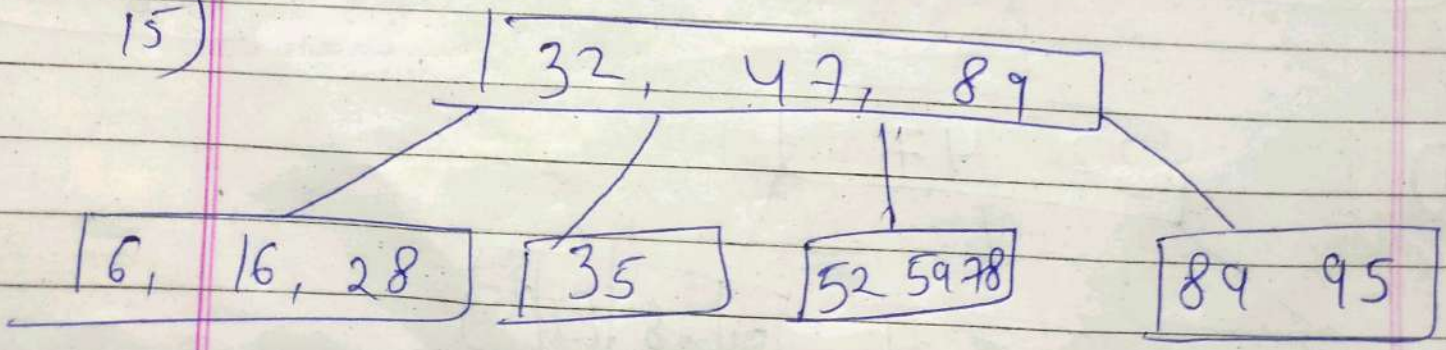
13)


32, 47, 87
6, 16, 28
35
51, 59
95

14)

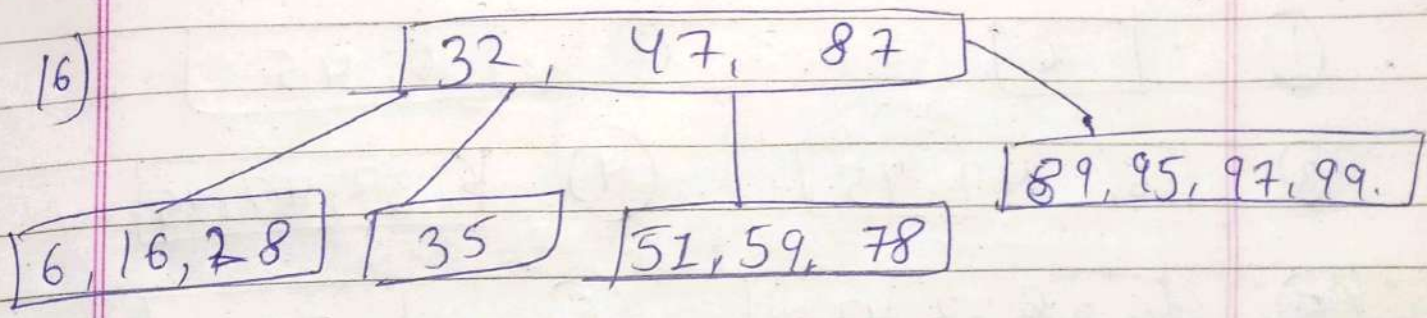

32, 47, 87
6, 16, 28
35
52 59 78
89, 95

15)

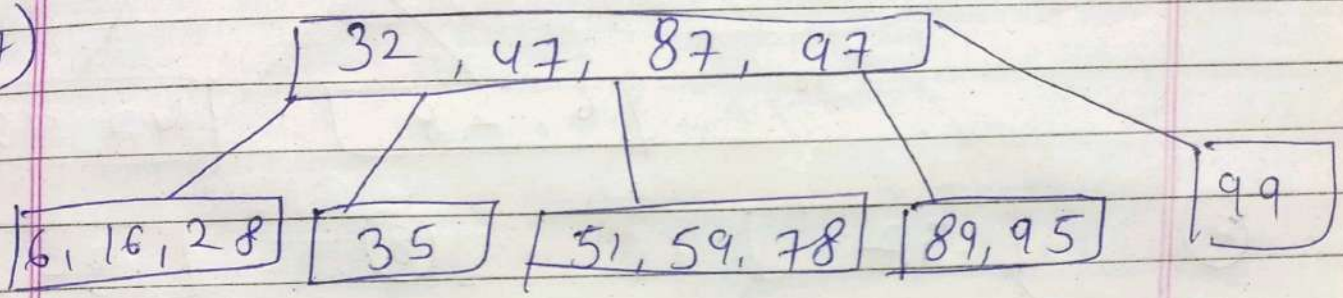

32, 47, 89
6, 16, 28
35
52 59 78
89 95

Date / /
Page

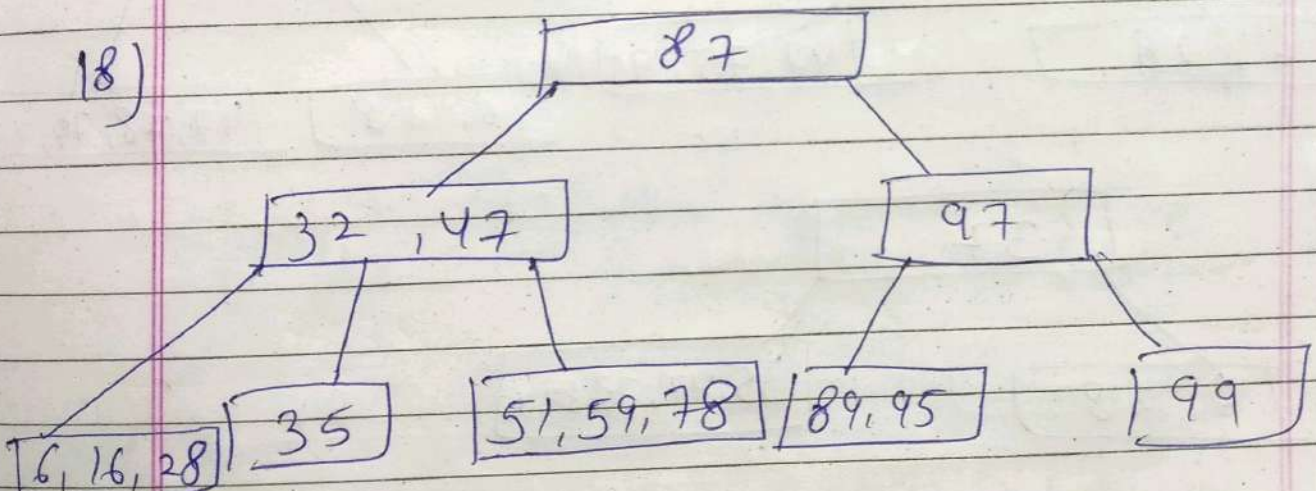
16)



17)



18)



Order - 5.

6, 95, 47, 32, 28, 75, 87, 35, 51, 16, 59, 89, 97, 99.

Date / /
Page

① [6]

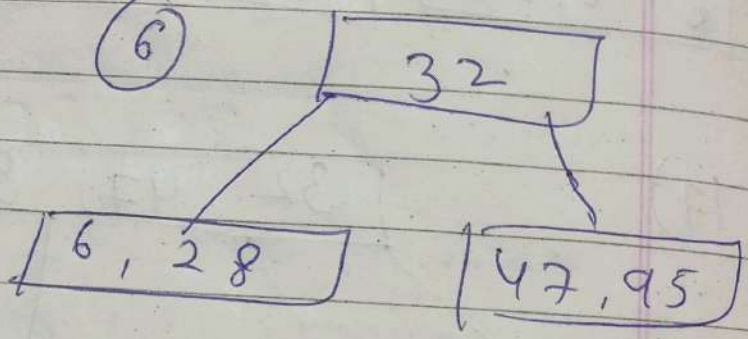
② [6, 95]

③ [6, 47, 95]

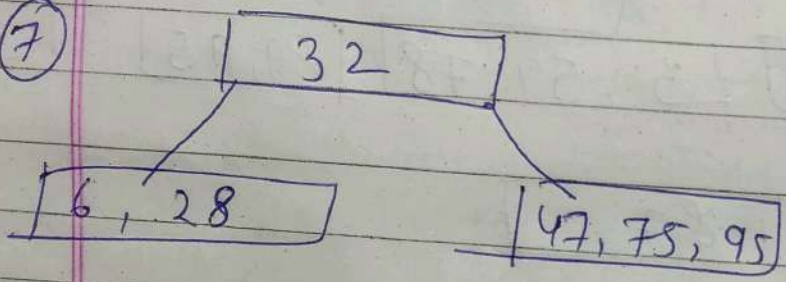
④ [6, 32, 47, 95]

⑤ [6, 28, 32, 47, 95]

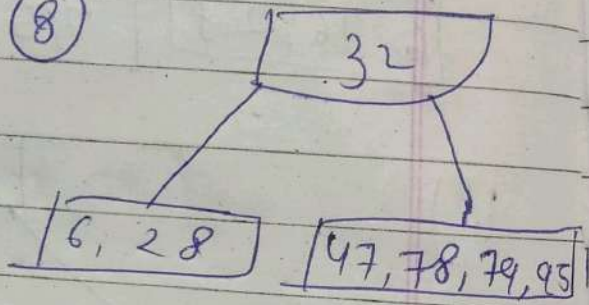
⑥



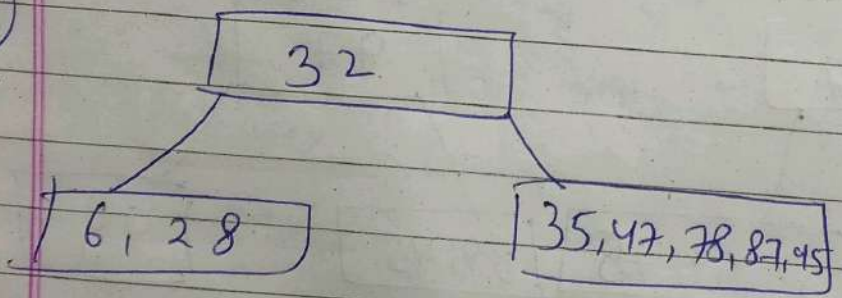
⑦



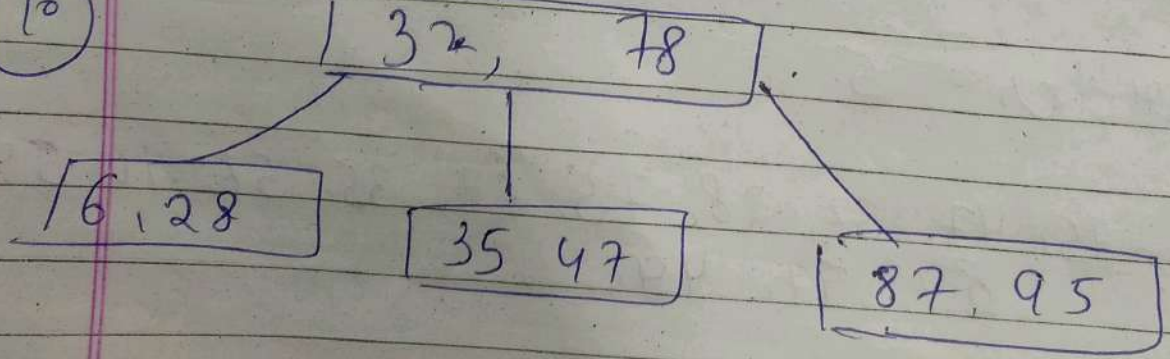
⑧



⑨

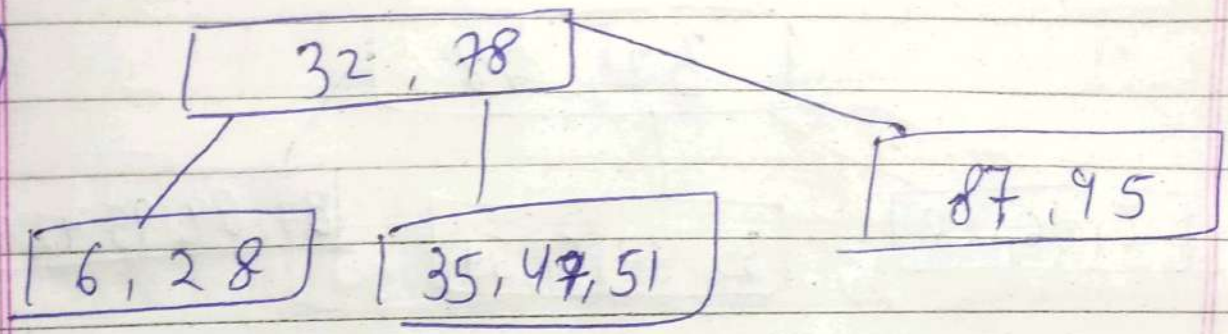


⑩

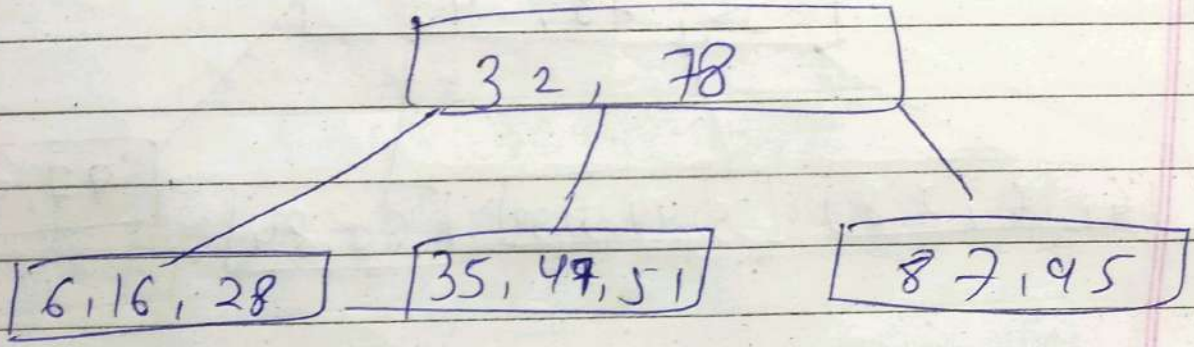


Date / /
Page

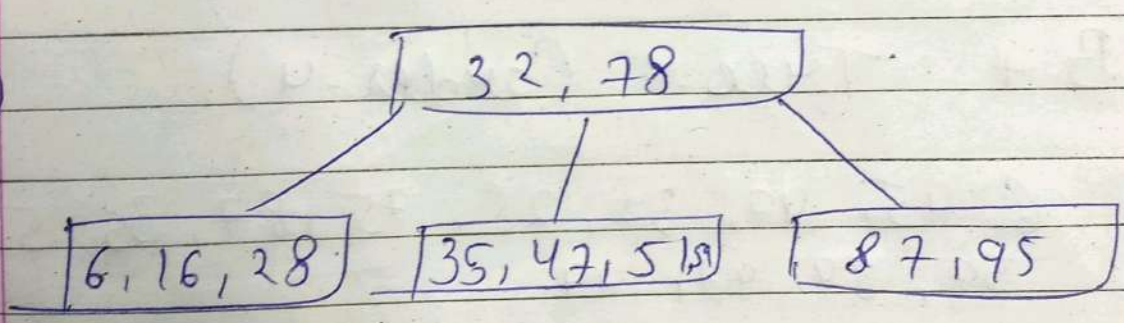
(11)



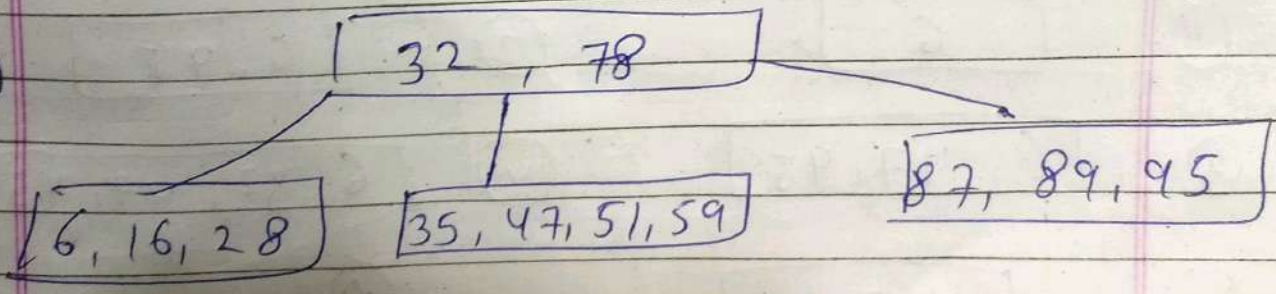
(12)



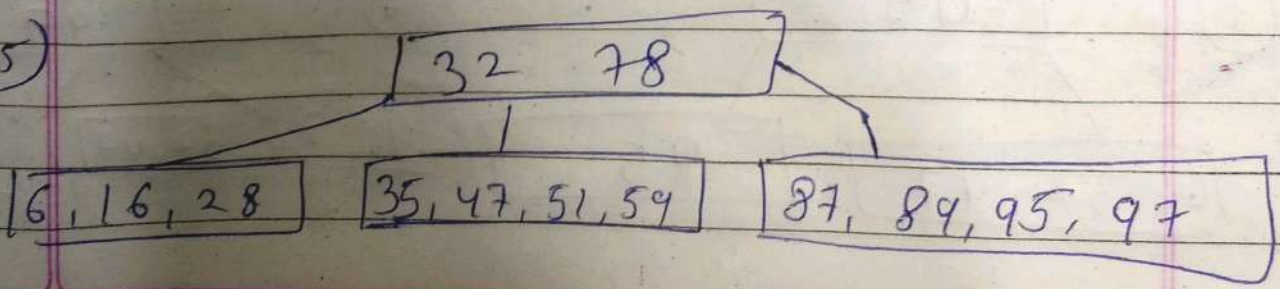
(13)



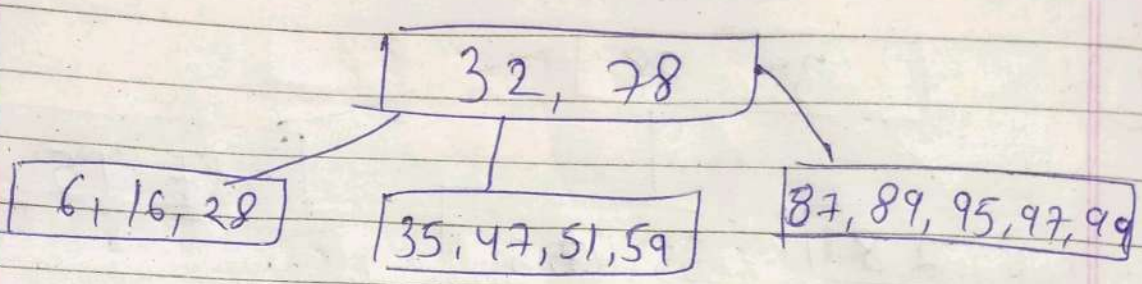
(14)



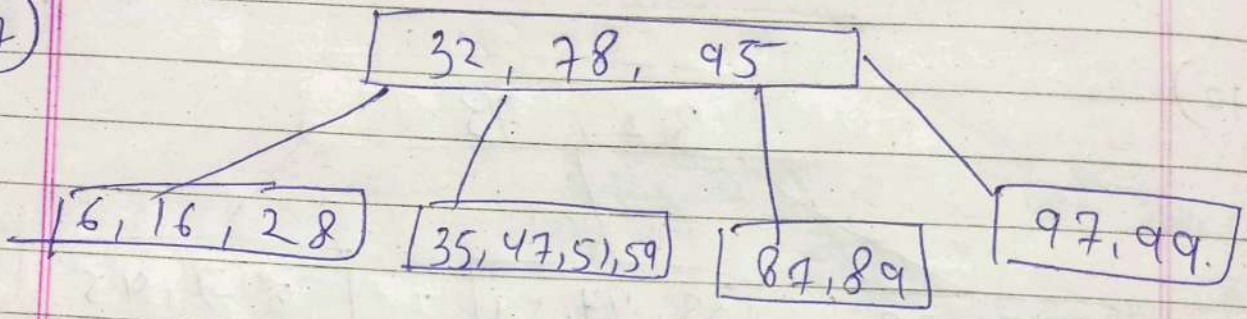
(15)



16)



(17)



B + Tree. (order-4)

6, 95, 47, 32, 28, 75, 87, 35, 51, 16, 59, 89, 97, 99.

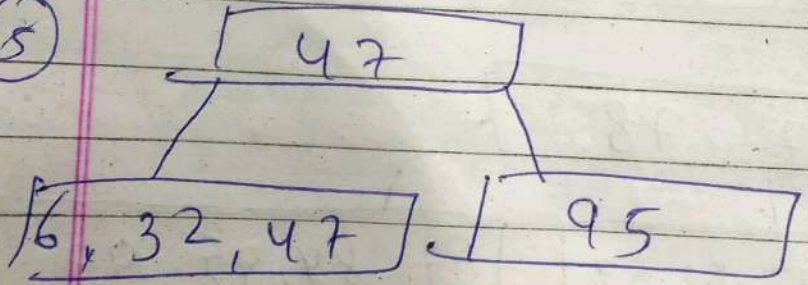
① 6

② 6, 95

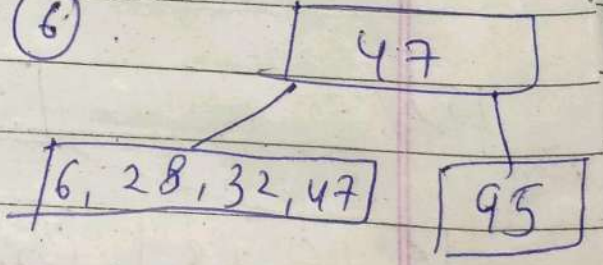
③ 6, 47, 95

④ 6, 32, 47, 95

⑤

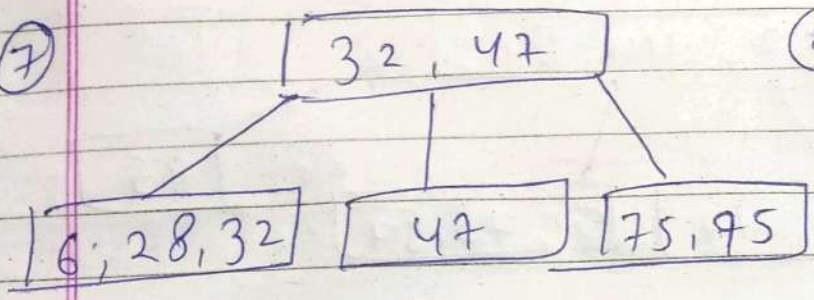


⑥

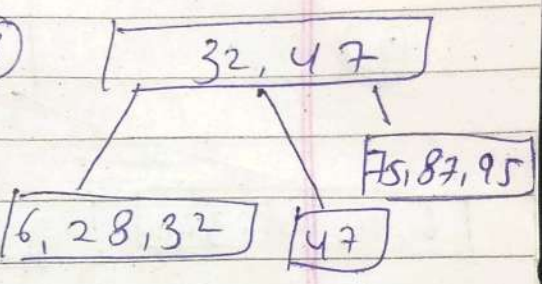


Date / /
Page

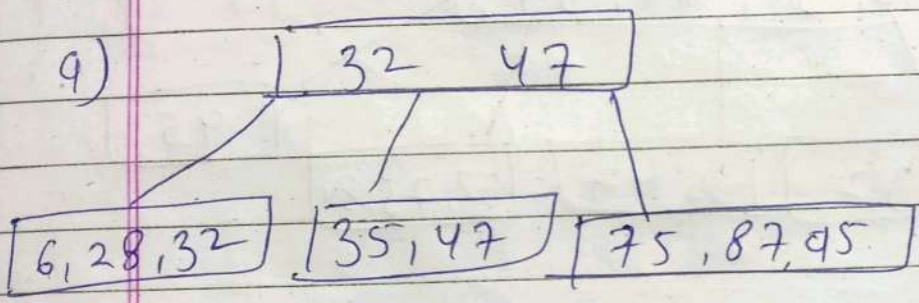
(7)



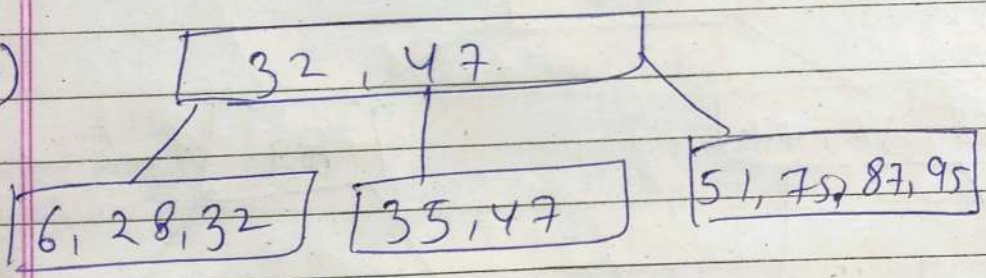
(8)



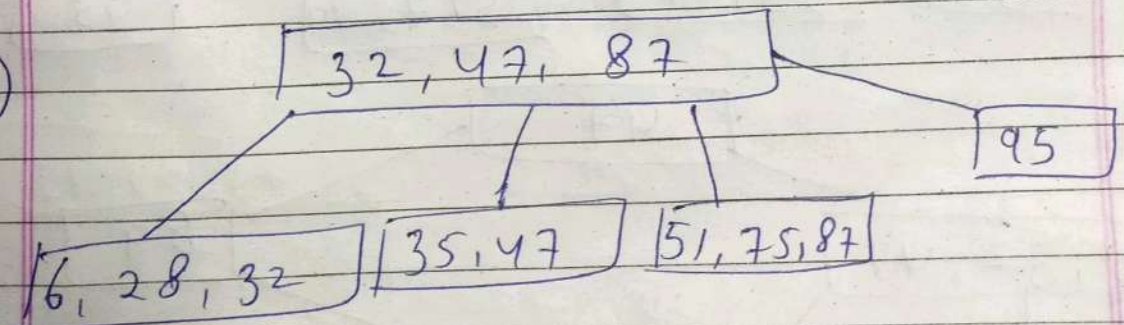
9)



10)

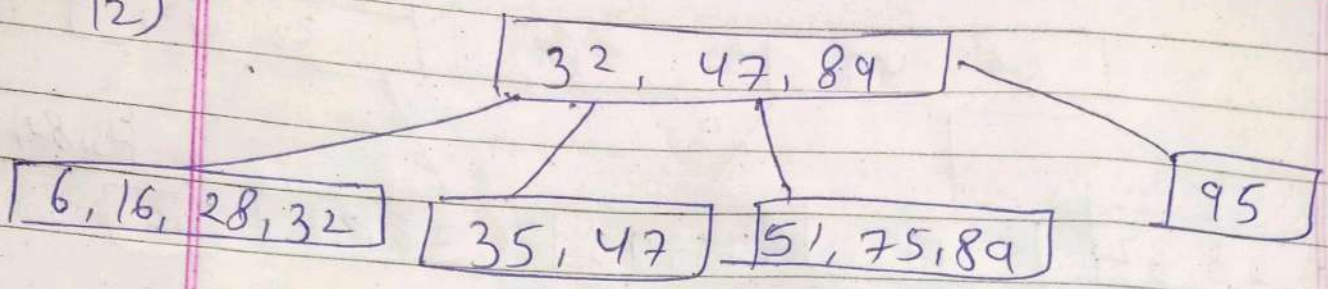


(11)

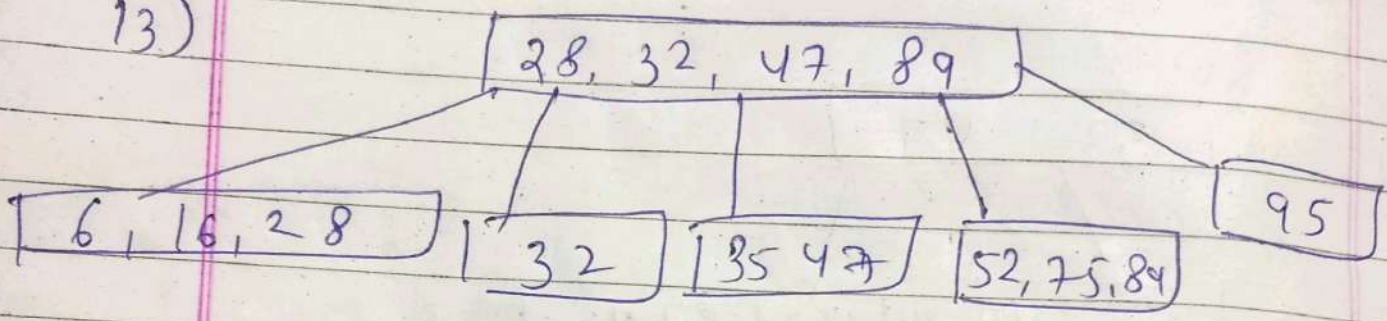


Date __/__/____
Page ____

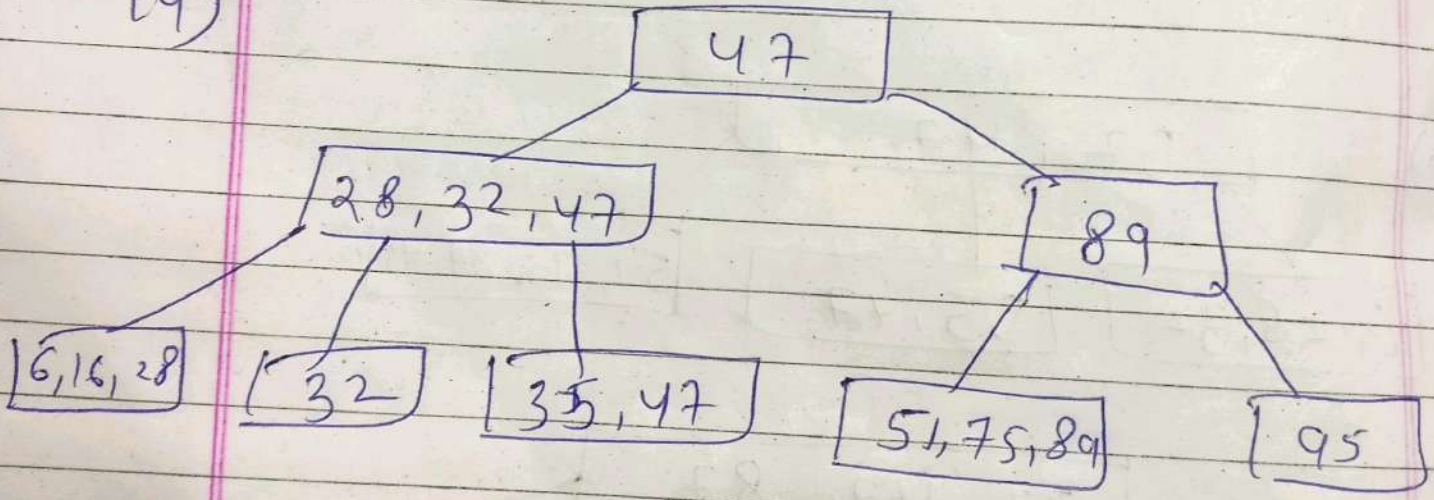
12)



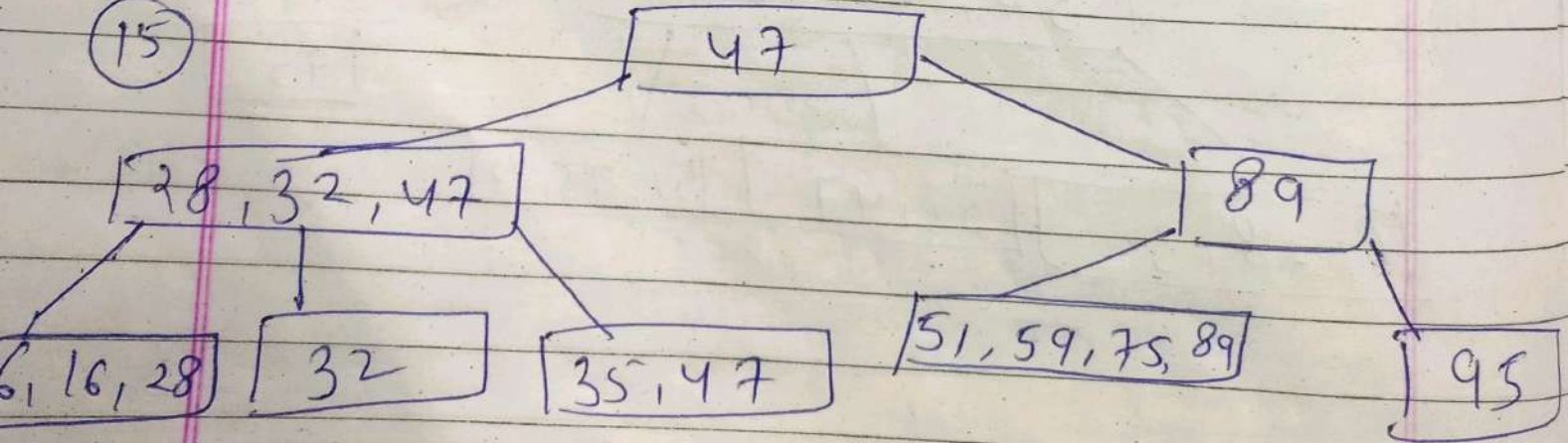
13)



14)

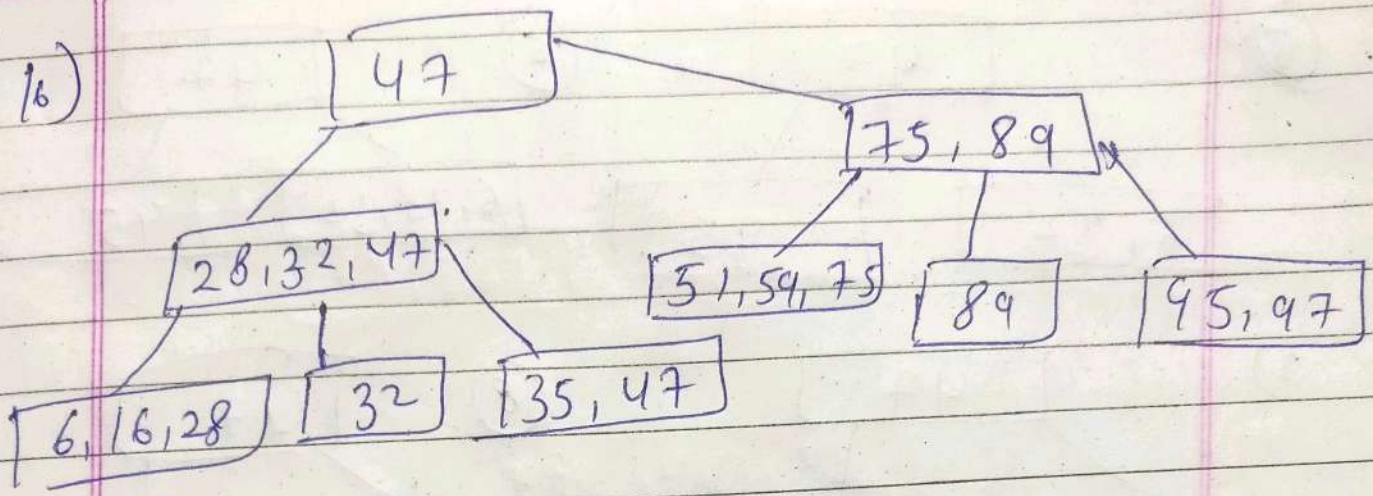


15

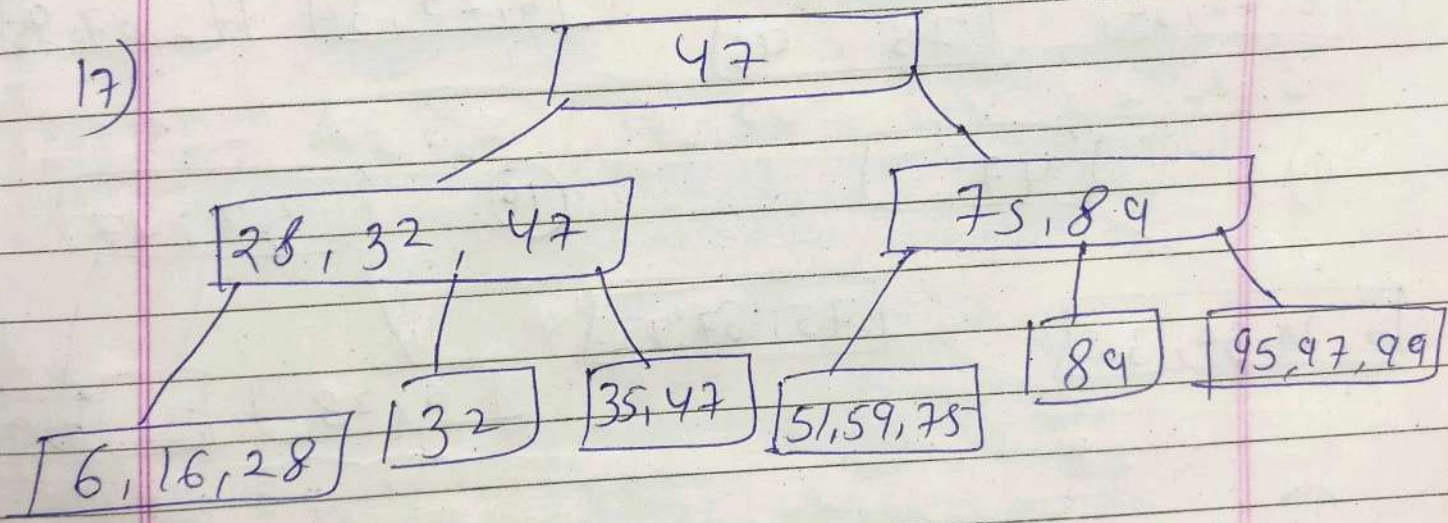


Date / /
Page

16)



17)



B * Tree (order - 4)

6, 95, 47, 32, 28, 78, 87, 95, 52, 16, 59, 89, 97, 99.

① 6

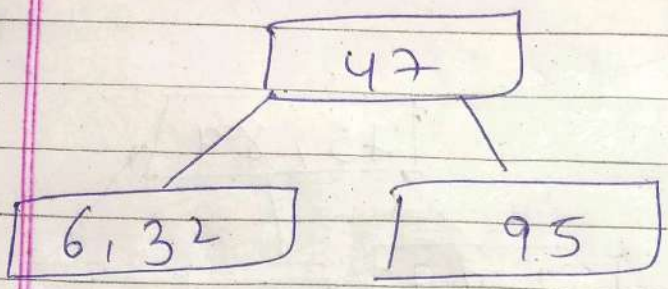
② 6, 95

③ 6, 47, 95

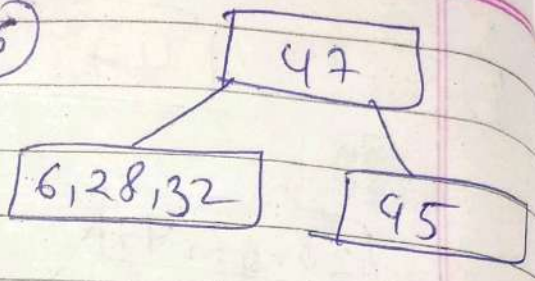
④ 6, 32, 47, 95

Date __/__/____
Page _____

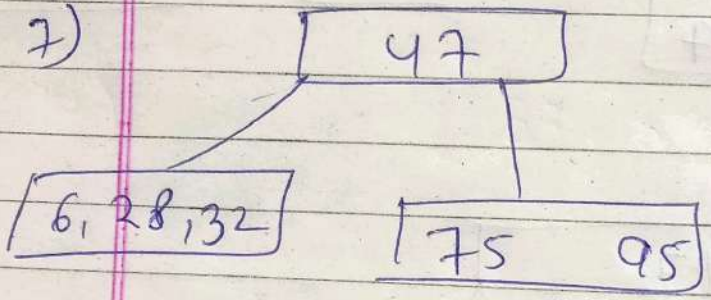
(5)



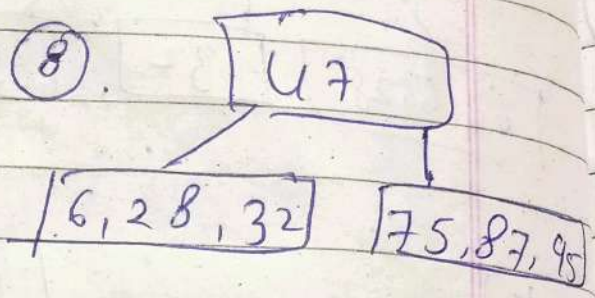
(6)



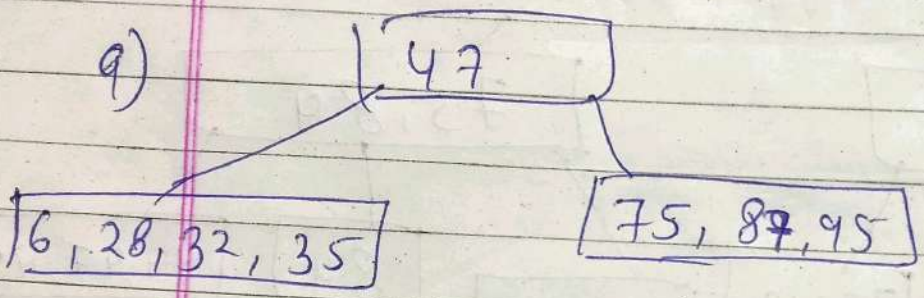
7)



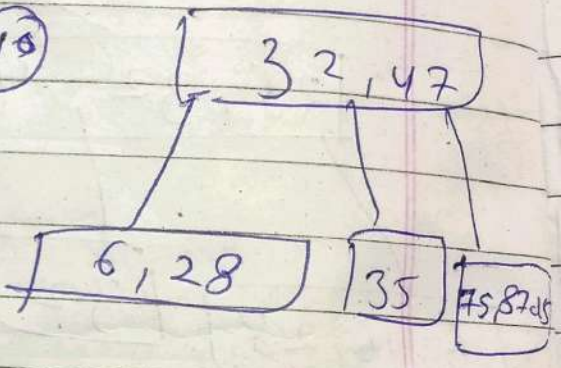
(8).



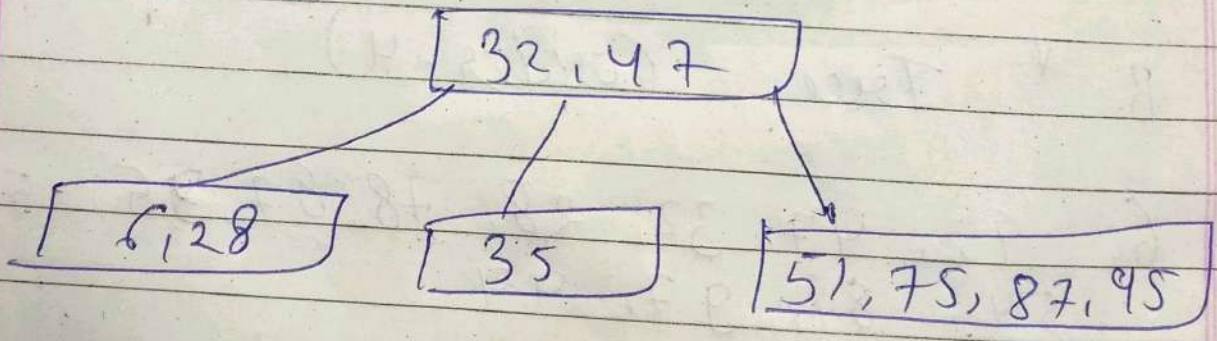
9)



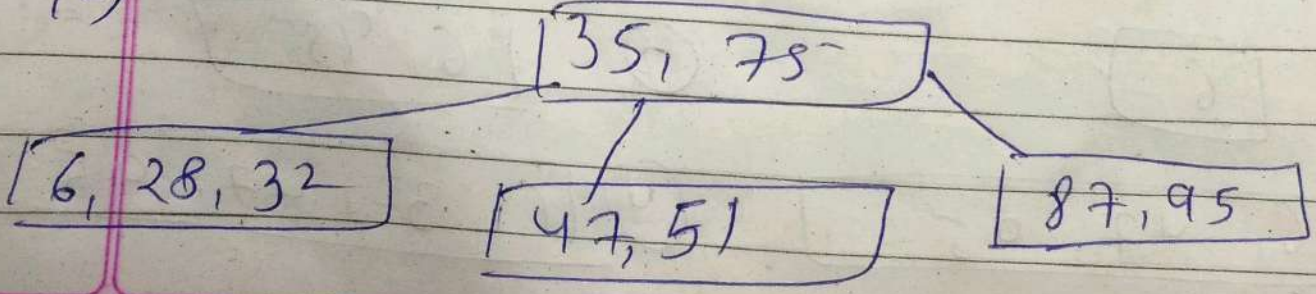
(10)



(11)

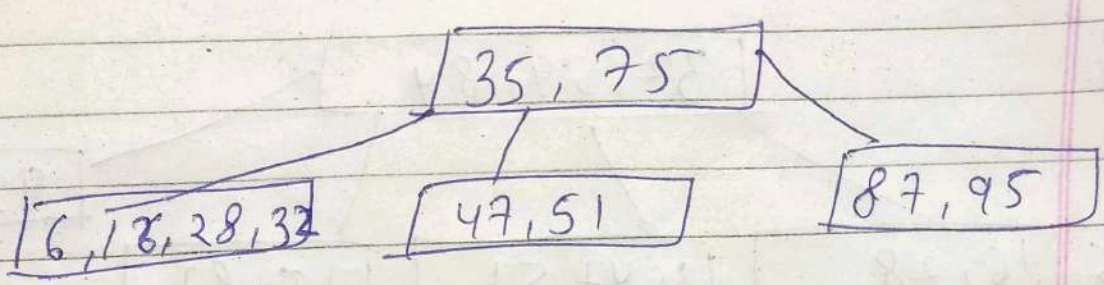


12)

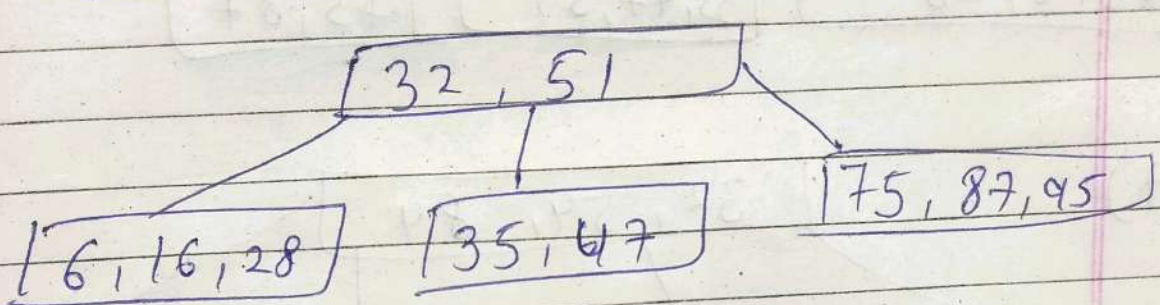


Date / /
Page

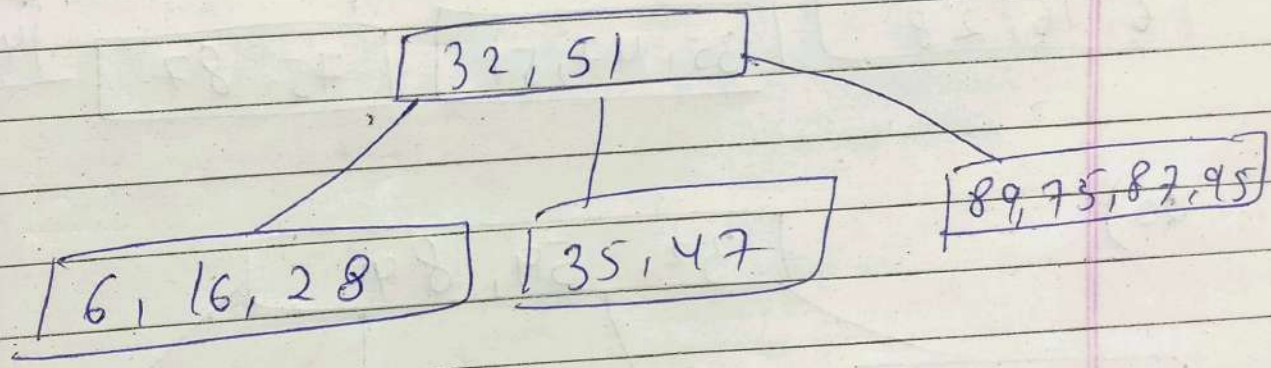
13)



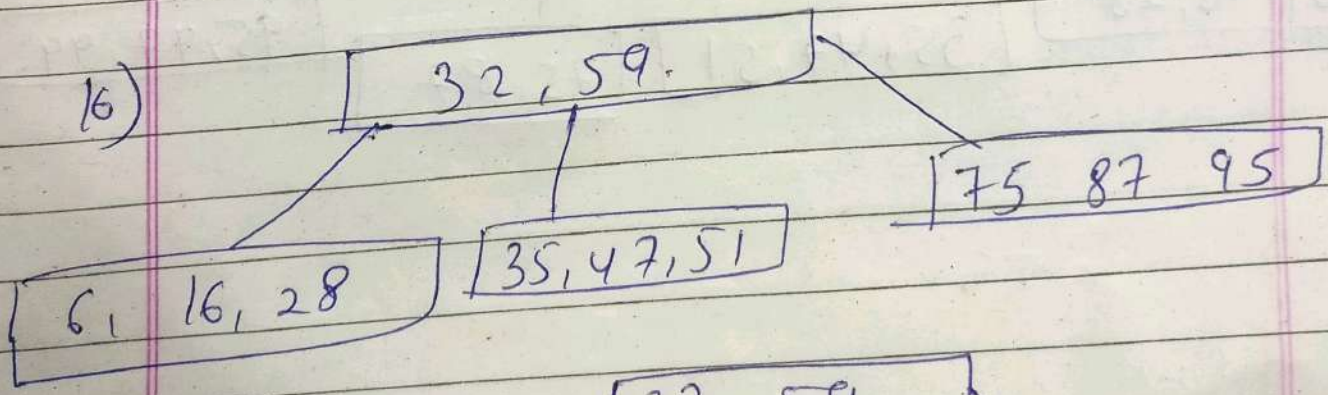
14)



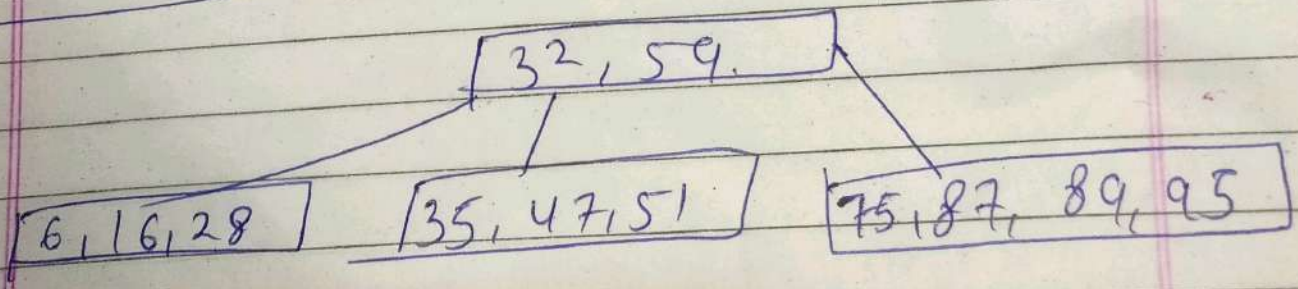
15)



16)

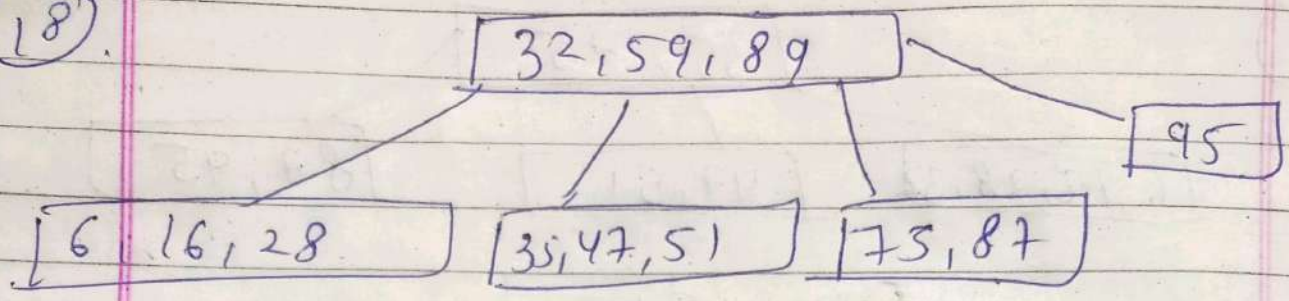


17)

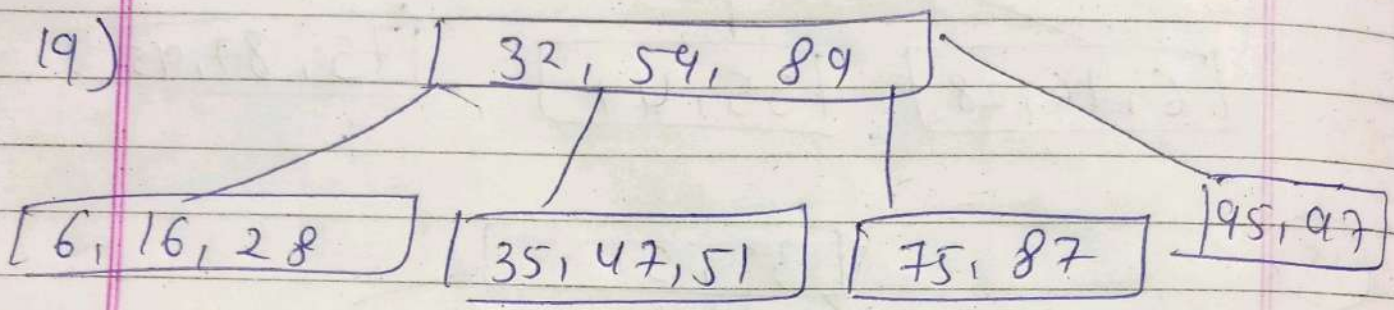


Date / /
Page

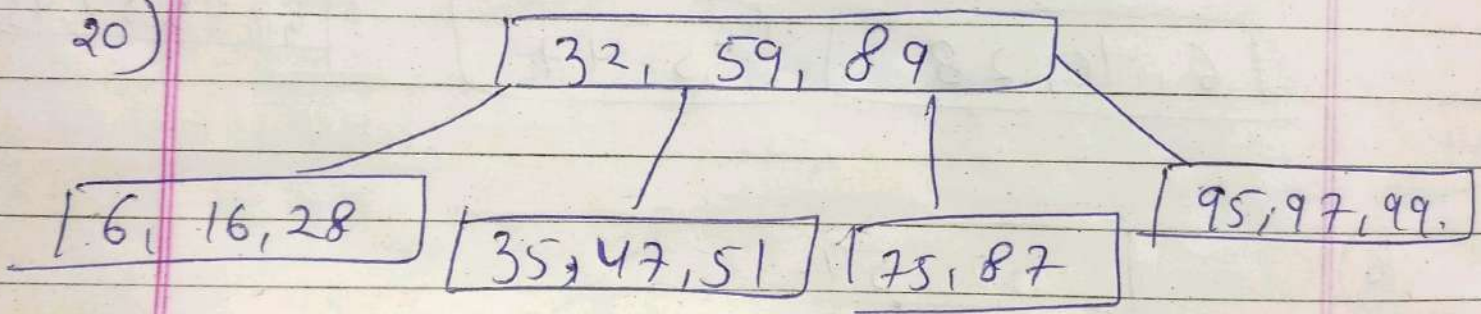
(18)



19)



20)



D E P A R T M E N T O F C O M P U T E R S C I E N C E
R O L L W A L A C O M P U T E R C E N T R E
G U J A R A T U N I V E R S I T Y
M.C.A. – III

R O L L NO : 30
N A M E : Ajinkya Rathod
S U B J E C T : DATA STRUCTURE

NO.	TITLE	DATE	SIGN
	ASSIGNMENT -1		
1	Polynomial Addition with Array	11/12/2020	
2	Polynomial Multiplication with Array	11/12/2020	
3	Polynomial Addition with LL	11/12/2020	
4	Polynomial Multiplication with LL	11/12/2020	
5	Singly Linked List (3 Insertions, 3 Deletions, Counting, Display List)	11/12/2020	
6	Stack with Array	11/12/2020	
7	Stack with LL	11/12/2020	
8	Factorial with Recursion	11/12/2020	
9	Fibonacci Series with Recursion	11/12/2020	
10	Any other Series with Recursion(eg. sine series)	11/12/2020	
11	Equation validation with brackets(using Array And Linklist)	11/12/2020	
12	Infix to Postfix Conversion	11/12/2020	
13	Postfix Expression Evaluation	11/12/2020	
14	Queue with Array	11/12/2020	
15	Queue with Linked List	11/12/2020	
16	Circular Queue with Array	11/12/2020	
17	Circular Queue with Linked List	11/12/2020	
18	Circular Linked List(All Insertions, All Deletions & Display)	11/12/2020	
	ASSIGNMENT -2		
1	Doubly Linklist(All Operation)	11/12/2020	
2	Doubly Circuler Linklist(All Operation)	11/12/2020	
3	Bubble Sort	11/12/2020	
4	Selection Sort	11/12/2020	
5	Insertion Sort	11/12/2020	

D E P A R T M E N T O F C O M P U T E R S C I E N C E
R O L L W A L A C O M P U T E R C E N T R E
G U J A R A T U N I V E R S I T Y
M.C.A. – III

R O L L NO : 30
N A M E : Ajinkya Rathod
S U B J E C T : DATA STRUCTURE

6	Linear Search	11/12/2020	
7	Binary Search	11/12/2020	
8	Expressio Tree	11/12/2020	
9	Binary Search Tree(Recursive)	11/12/2020	
10	Binary Search Tree(Non-Recursive)	11/12/2020	
	ASSIGNMENT -3		
1	UnWeighted Graph Shortest Path	11/12/2020	
2	Dijkastra's Weighted Graph Shortest Path	11/12/2020	
3	Priority Queue Using MIN Heap	11/12/2020	
4	MAX Heap	11/12/2020	
5	Heap Sort	11/12/2020	
6	Quick Sort	11/12/2020	
7	Radix Sort	11/12/2020	
8	Shell Sort	11/12/2020	
9	Merge Sort	11/12/2020	
10	BFS DFS	11/12/2020	
11	Prims Minimum Spanning Tree	11/12/2020	
12	Kruskals Minimum Spanning Tree	11/12/2020	

Roll No : 30
Name : Ajinkya Rathod
Subject : Data Structure
Class : MCA-3

ASSIGNMENT -1

==========
1. Polynomial Addition with Array
==========

```
#include<stdio.h>
#include<conio.h>

void main()
{
	int n,n1,n2,poly1[10]={0}, poly2[10]={0},poly3[10]={0},i,j,res;

	printf("\n Enter Max of polynomial:");
	scanf("%d",&n1);

	printf("\n Enter Polynomial 1..");
	for(i=0;i<=n1;i++)
	{
		printf("\n Enter Poly co_%d :", i);
		scanf("%d",&poly1[i]);
	}

	printf("\n Enter Max of polynomial 2:");
	scanf("%d",&n2);

	printf("\n Enter Polynomial 2..");
	for(i=0;i<=n2;i++)
	{
		printf("\n Enter Poly co_%d :", i);
		scanf("%d",&poly2[i]);
	}

	printf("\n Polynomial 1..=> ");
	for(i=n1;i>=0;i--)
	{
		if(poly1[i]>=0)
		{
```

```
                if(i!=n1)
                {
                        printf("+ ");
                }
        }
        printf("%dX^%d ",poly1[i],i);
    }

    printf("\n Polynomial 2..=> ");
    for(i=n2;i>=0;i--)
    {
        if(poly2[i]>=0)
        {
                if(i!=n2)
                {
                        printf("+ ");
                }
        }
        printf("%dX^%d ",poly2[i],i);
    }

    res=(n1>n2)?n1:n2;
    for(i=0;i<=res;i++)
    {
        poly3[i]=poly1[i]+poly2[i];
    }

    printf("\n Addition of two polynomial");
    printf("\n-------------------------- \n");
    for(i=(res);i>=0;i--)
    {

        if(poly3[i]>=0)
        {
                if(i!=(res))
                {
                        printf("+ ");
                }
        }
        printf("%dX^%d ",poly3[i],i);
    }
    getch();
}
```

==========
OUTPUT
==========

```
 Enter Max of polynomial:5

 Enter Polynomial 1..
 Enter Poly co_0 :-9

 Enter Poly co_1 :6

 Enter Poly co_2 :3

 Enter Poly co_3 :2

 Enter Poly co_4 :2

 Enter Poly co_5 :2

 Enter Max of polynomial 2:4

 Enter Polynomial 2..
 Enter Poly co_0 :4

 Enter Poly co_1 :-9

 Enter Poly co_2 :0

 Enter Poly co_3 :1

 Enter Poly co_4 :3

 Polynomial 1..=> 2X^5 + 2X^4 + 2X^3 + 3X^2 + 6X^1 -9X^0
 Polynomial 2..=> 3X^4 + 1X^3 + 0X^2 -9X^1 + 4X^0
 Addition of two polynomial
--------------------------
2X^5 + 5X^4 + 3X^3 + 3X^2 -3X^1 -5X^0

========
OUTPUT
========

 Enter Max of polynomial:4

 Enter Polynomial 1..
 Enter Poly co_0 :-9

 Enter Poly co_1 :6

 Enter Poly co_2 :3
```

```
 Enter Poly co_3 :0

 Enter Poly co_4 :2

 Enter Max of polynomial 2:4

 Enter Polynomial 2..
 Enter Poly co_0 :4

 Enter Poly co_1 :-9

 Enter Poly co_2 :0

 Enter Poly co_3 :1

 Enter Poly co_4 :3

 Polynomial 1..=> 2X^4 + 0X^3 + 3X^2 + 6X^1 -9X^0
 Polynomial 2..=> 3X^4 + 1X^3 + 0X^2 -9X^1 + 4X^0
 Addition of two polynomial
--------------------------
5X^4 + 1X^3 + 3X^2 -3X^1 -5X^0
```

==========

2. Polynomial Multiplication with Array

==========

```
#include<stdio.h>
#include<conio.h>

void main()
{
	int n,n1,n2,poly1[10], poly2[10],poly3[50][50],poly4[50]
[50],i,j,res,coff,pw,k=0,sum;

	printf("\n Enter Max of polynomial:");
	scanf("%d",&n1);

	printf("\n Enter Polynomial 1..");
	for(i=0;i<=n1;i++)
	{
		printf("\n Enter Poly co_%d :", i);
		scanf("%d",&poly1[i]);
	}
```

```
printf("\n Enter Max of polynomial 2:");
scanf("%d",&n2);

printf("\n Enter Polynomial 2..");
for(i=0;i<=n2;i++)
{
	printf("\n Enter Poly co_%d :", i);
	scanf("%d",&poly2[i]);
}

printf("\n Polynomial 1..=> ");
for(i=n1;i>=0;i--)
{
	if(poly1[i]>=0)
	{
		if(i!=n1)
		{
			printf("+ ");
		}
	}
	printf("%dX^%d ",poly1[i],i);
}

printf("\n Polynomial 2..=> ");
for(i=n2;i>=0;i--)
{
	if(poly2[i]>=0)
	{
		if(i!=n2)
		{
			printf("+ ");
		}
	}
	printf("%dX^%d ",poly2[i],i);
}

if(n1==n2)
{
	res=n1;
}
else if(n1>n2)
{
	res=n1;
}
else
{
	res=n2;
```

```
}

k=0;
for(i=0;i<=n1;i++)
{
	for(j=0;j<=n2;j++)
	{
		coff=poly1[i]*poly2[j];
		pw=i+j;

		poly3[k][0]=coff;
		poly3[k][1]=pw;
		k++;
	}
}


for(k=0;k<=(n1+n2);k++)
{
	sum=0;
	for(i=0;i<((n1+1)*(n2+1));i++)
	{
		if(poly3[i][1]==k)
		{
			sum = sum + poly3[i][0];
		}
	}
	poly4[k][0]=sum;
	poly4[k][1]=k;
}


printf("\n Multiplication of Polynomial");
printf("\n----------------------------\n");

for(i=(n1+n2);i>=0;i--)
{
	for(j=0;j<1;j++)
	{
		if(poly4[i][j]>=0)
		{
			if(i!=(n1+n2))
			{
				printf("+ ");
			}
		}
	printf("%dX^%d ",poly4[i][j], poly4[i][j+1]);
	}
```

```
        }
        getch();
}
```

```
=========
OUTPUT
=========

 Enter Max of polynomial:4

 Enter Polynomial 1..
 Enter Poly co_0 :-9

 Enter Poly co_1 :6

 Enter Poly co_2 :3

 Enter Poly co_3 :0

 Enter Poly co_4 :2

 Enter Max of polynomial 2:4

 Enter Polynomial 2..
 Enter Poly co_0 :4

 Enter Poly co_1 :-9

 Enter Poly co_2 :0

 Enter Poly co_3 :1

 Enter Poly co_4 :3

 Polynomial 1..=> 2X^4 + 0X^3 + 3X^2 + 6X^1 -9X^0
 Polynomial 2..=> 3X^4 + 1X^3 + 0X^2 -9X^1 + 4X^0
 Multiplication of Polynomial
----------------------------
6X^8 + 2X^7 + 9X^6 + 3X^5 -13X^4 -36X^3 -42X^2 + 105X^1 -36X^0
```

```
================================
3. Polynomial Addition with Link List
================================
```

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

struct Poly
{
	int coff;
	int pw;
	struct Poly *next;
};

struct Poly * Create_Poly(struct Poly *first,struct Poly **last )
{
	int n;
	printf("\n Enter Max of polynomial:");
	scanf("%d",&n);

	while(n!=-1)
	{
		struct Poly *new_node=NULL;

		new_node=(struct Poly *)malloc(sizeof(struct Poly));

		printf("\n Enter coeffcient:");
		scanf("%d",&new_node->coff);
		printf("\n Enter Power:");
		scanf("%d",&new_node->pw);
		new_node->next=NULL;

		if(first==NULL)
		{
			first=new_node;
			*last=first;
		}
		else
		{
			(*last)->next=new_node;
			(*last)=new_node;
		}
		n--;
	}
	printf("\n List Created Successfully.");
	return first;
}
void Display(struct Poly *first)
{
	struct Poly *p;
	p=first;
```

```
	while(p!=NULL)
	{
		if(p->coff>=0)
		{
				if(p!=first)
				{
					printf("+ ");
				}

		}
		printf("%dX^%d ",p->coff,p->pw);
		p=p->next;
	}


}

void  Addition(struct Poly *first,struct Poly *first_2,struct Poly *first_3,struct
Poly **last_3)
{
	int n,n1,n2,p1_coff=0,p2_coff=0,sum;
	struct Poly *p,*p2,*outer_first,*outer_first_2;

	n1=first->pw;		//max coeff poly1
	n2=first->pw;		//max coeff poly2
	if(n1>n2)
		n=n1;//n is the maximum coefficient from two poly
	else
		n=n2;

	outer_first=first;
	outer_first_2=first_2;


	while(outer_first!=NULL || outer_first_2!=NULL)
	{
		struct Poly *new_node=NULL;

		new_node=(struct Poly *)malloc(sizeof(struct Poly));

			new_node->next=NULL;

		p=first;
		p2=first_2;
		p1_coff=0;
		p2_coff=0;
```

```
while(n<=p->pw && p->next!=NULL)
{

	if(p->pw==n)
	{
		p1_coff=p->coff;
	}

	if(n==0 && p->next->pw==0)
	{
		p1_coff=p->next->coff;
	}
	p=p->next;

}
while(n<=p2->pw && p2->next!=NULL)
{
	if(p2->pw==n)
	{
		p2_coff=p2->coff;
	}
	if(n==0 && p2->next->pw==0)
	{
		p2_coff=p2->next->coff;
	}
	p2=p2->next;
}


sum=(p1_coff)+(p2_coff);
new_node->coff=sum;
new_node->pw=n;

/// add the node
if(first_3==NULL)
{
	first_3=new_node;
	(*last_3)=first_3;
}
else
{
	(*last_3)->next=new_node;
	(*last_3)=new_node;
}


if(outer_first!=NULL)
{
```

```
                outer_first=outer_first->next;
        }

        if(outer_first_2!=NULL)
        {
                outer_first_2=outer_first_2->next;
        }
        n--;

    }

    printf("\n Addition of two Polynomial");
    printf("\n--------------------------\n");

    Display(first_3);
}

void main()
{
    struct Poly *first=NULL;
    struct Poly *last=NULL;

    struct Poly *first_2=NULL;
    struct Poly *last_2=NULL;

    struct Poly *first_3=NULL;
    struct Poly *last_3=NULL;

    int ch,count;
    char choice='n';

    printf("\n---------------");
    printf("\n 1.Create two Polynomial");
    printf("\n 2.Display");
    printf("\n 3.Addition");
    printf("\n---------------");

    do
    {
        printf("\n Enter Your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                    first=Create_Poly(first,&last);
                    first_2=Create_Poly(first_2,&last_2);
                    break;
            case 2:
```

```
                                printf("\n Polynomial 1.. -> ");
                                Display(first);

                                printf("\n Polynomial 2.. -> ");
                                Display(first_2);
                                break;
                        case 3:
                                Addition(first,first_2,first_3,&last_3);
                                break;
                }
                printf("\n Do You want to continue:");
                scanf(" %c",&choice);


        }while(choice=='y' || choice=='Y');


        getch();
}
```

==========
OUTPUT
==========

```
---------------
 1.Create two Polynomial
 2.Display
 3.Addition
---------------
 Enter Your choice:1

 Enter Max of polynomial:4

 Enter coeffcient:2

 Enter Power:4

 Enter coeffcient:0

 Enter Power:3

 Enter coeffcient:3

 Enter Power:2

 Enter coeffcient:6
```

```
 Enter Power:1

 Enter coeffcient:-9

 Enter Power:0

 List Created Successfully.
 Enter Max of polynomial:4

 Enter coeffcient:3

 Enter Power:4

 Enter coeffcient:1

 Enter Power:3

 Enter coeffcient:0

 Enter Power:2

 Enter coeffcient:-9

 Enter Power:1

 Enter coeffcient:4

 Enter Power:0

 List Created Successfully.
 Do You want to continue:y

 Enter Your choice:2

 Polynomial 1.. -> 2X^4 + 0X^3 + 3X^2 + 6X^1 -9X^0
 Polynomial 2.. -> 3X^4 + 1X^3 + 0X^2 -9X^1 + 4X^0
 Do You want to continue:y

 Enter Your choice:3

  Addition of two Polynomial
--------------------------
5X^4 + 1X^3 + 3X^2 -3X^1 -5X^0
 Do You want to continue:n


=========
OUTPUT-2
```

```
=========

--------------
 1.Create two Polynomial
 2.Display
 3.Addition
--------------
 Enter Your choice:1

 Enter Max of polynomial:5

 Enter coeffcient:2

 Enter Power:5

 Enter coeffcient:2

 Enter Power:4

 Enter coeffcient:2

 Enter Power:3

 Enter coeffcient:3

 Enter Power:2

 Enter coeffcient:6

 Enter Power:1

 Enter coeffcient:-9

 Enter Power:0

 List Created Successfully.
 Enter Max of polynomial:4

 Enter coeffcient:3

 Enter Power:4

 Enter coeffcient:1

 Enter Power:3

 Enter coeffcient:0
```

```
 Enter Power:2

 Enter coeffcient:-9

 Enter Power:1

 Enter coeffcient:4

 Enter Power:0

 List Created Successfully.
 Do You want to continue:y

 Enter Your choice:2

 Polynomial 1.. -> 2X^5 + 2X^4 + 2X^3 + 3X^2 + 6X^1 -9X^0
 Polynomial 2.. -> 3X^4 + 1X^3 + 0X^2 -9X^1 + 4X^0
 Do You want to continue:y

 Enter Your choice:3

 Addition of two Polynomial
--------------------------
2X^5 + 5X^4 + 3X^3 + 3X^2 -3X^1 -5X^0
 Do You want to continue:n
```

==========
4. Polynomial Multiplication with LL
==========

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

struct Poly
{
        int coff;
        int pw;
        struct Poly *next;
};

struct Poly * Create_Poly(struct Poly *first,struct Poly **last )
{
        int n;
```

```
	printf("\n Enter Max of polynomial:");
	scanf("%d",&n);

	while(n!=-1)
	{
		struct Poly *new_node=NULL;

		new_node=(struct Poly *)malloc(sizeof(struct Poly));

		printf("\n Enter coeffcient:");
		scanf("%d",&new_node->coff);
		printf("\n Enter Power:");
		scanf("%d",&new_node->pw);
		new_node->next=NULL;

		if(first==NULL)
		{
			first=new_node;
			*last=first;
		}
		else
		{
			(*last)->next=new_node;
			(*last)=new_node;
		}
		n--;
	}
	printf("\n List Created Successfully.");
	return first;
}
void Display(struct Poly *first)
{
	struct Poly *p;
	p=first;

	while(p!=NULL)
	{
		if(p->coff>=0)
		{
				if(p!=first)
				{
					printf("+ ");
				}

		}
		printf("%dX^%d ",p->coff,p->pw);
		p=p->next;
	}
```

```
}

void  Multiplication(struct Poly *first,struct Poly *first_2,struct Poly
*first_3,struct Poly **last_3)
{
	int n,n1,n2,coff1,coff2,pw1,pw2,max_co,sum=0,found=0;

	struct Poly *p,*p2,*first_temp=NULL,*last_temp=NULL;

	n1=first->pw;		//max coeff poly1
	n2=first->pw;		//max coeff poly2


	//cross multiply
	p=first;
	while(p!=NULL)
	{
		coff1=p->coff;
		pw1=p->pw;
		p2=first_2;
		while(p2!=NULL)
		{
			struct Poly *new_node=NULL;

			new_node=(struct Poly *)malloc(sizeof(struct Poly));
			new_node->next=NULL;

			coff2=p2->coff;
			pw2=p2->pw;


			new_node->coff=(coff1*coff2);
			new_node->pw=(pw1+pw2);

			/// add the node
			if(first_temp==NULL)
			{
				first_temp=new_node;
				last_temp=first_temp;
			}
			else
			{
				last_temp->next=new_node;
				last_temp=new_node;
			}
```

```
                p2=p2->next;
        }
        p=p->next;
}


max_co=(n1+n2);

for(int i=max_co;i>=0;i--)
{
        p=first_temp;
        sum=0;
        found=0;
        while(p!=NULL)
        {
                if(p->pw==i)
                {
                        sum=sum+p->coff;
                        found=1;
                }

                p=p->next;
        }

        if(found)
        {
                struct Poly *new_node=NULL;
                new_node=(struct Poly *)malloc(sizeof(struct Poly));
                new_node->next=NULL;
                new_node->coff=sum;
                new_node->pw=i;

                if(first_3==NULL)
                {
                        first_3=new_node;
                        (*last_3)=first_3;
                }
                else
                {
                        (*last_3)->next=new_node;
                        (*last_3)=new_node;
                }

        }

}
```

```
        printf("\n Multiplication of two Polynomial");
        printf("\n--------------------------\n");

        Display(first_3);
}

void main()
{
        struct Poly *first=NULL;
        struct Poly *last=NULL;

        struct Poly *first_2=NULL;
        struct Poly *last_2=NULL;

        struct Poly *first_3=NULL;
        struct Poly *last_3=NULL;

        int ch,count;
        char choice='n';

        printf("\n---------------");
        printf("\n 1.Create two Polynomial");
        printf("\n 2.Display");
        printf("\n 3.Addition");
        printf("\n---------------");

        do
        {
                printf("\n Enter Your choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                        first=Create_Poly(first,&last);
                                        first_2=Create_Poly(first_2,&last_2);
                                        break;
                        case 2:
                                        printf("\n Polynomial 1.. -> ");
                                        Display(first);

                                        printf("\n Polynomial 2.. -> ");
                                        Display(first_2);
                                        break;
                        case 3:
                                        Multiplication(first,first_2,first_3,&last_3);
```

```
                                    break;
                }
                printf("\n Do You want to continue:");
                scanf(" %c",&choice);


        }while(choice=='y' || choice=='Y');


        getch();
}
```

==========
OUTPUT
==========

```
---------------
 1.Create two Polynomial
 2.Display
 3.Addition
---------------
 Enter Your choice:1

 Enter Max of polynomial:4

 Enter coeffcient:2

 Enter Power:4

 Enter coeffcient:0

 Enter Power:3

 Enter coeffcient:3

 Enter Power:2

 Enter coeffcient:6

 Enter Power:1

 Enter coeffcient:-9

 Enter Power:0

 List Created Successfully.
 Enter Max of polynomial:4
```

```
 Enter coeffcient:3

 Enter Power:4

 Enter coeffcient:1

 Enter Power:3

 Enter coeffcient:0

 Enter Power:2

 Enter coeffcient:-9

 Enter Power:1

 Enter coeffcient:4

 Enter Power:0

 List Created Successfully.
 Do You want to continue:y

 Enter Your choice:2

 Polynomial 1.. -> 2X^4 + 0X^3 + 3X^2 + 6X^1 -9X^0
 Polynomial 2.. -> 3X^4 + 1X^3 + 0X^2 -9X^1 + 4X^0
 Do You want to continue:y

 Enter Your choice:3

 Multiplication of two Polynomial
--------------------------
6X^8 + 2X^7 + 9X^6 + 3X^5 -13X^4 -36X^3 -42X^2 + 105X^1 -36X^0
 Do You want to continue:n
```

==========

5. Singly Linked List (3 Insertions, 3 Deletions, Counting, Display List)

==========

```
#include<stdio.h>
```

```
#include<conio.h>
#include<malloc.h>

struct Stud
{
	int data;
	struct Stud *next;
};

struct Stud * CreateList(struct Stud *first,struct Stud **last )
{
	int n;
	printf("\n Enter number of nodes you want to create:");
	scanf("%d",&n);

	while(n!=0)
	{

		struct Stud *new_node=NULL;

		new_node=(struct Stud *)malloc(sizeof(struct Stud));

		printf("\n Enter data:");
		scanf("%d",&new_node->data);
		new_node->next=NULL;

		if(first==NULL)
		{
			first=new_node;
			*last=first;
		}
		else
		{
			(*last)->next=new_node;
			(*last)=new_node;
		}
		n--;
	}
	printf("\n List Created Successfully.");
	return first;
}
struct Stud * InsertEnd(struct Stud *first,struct Stud **last )
{
	struct Stud *new_node=NULL;
	new_node=(struct Stud *)malloc(sizeof(struct Stud));

	printf("\n Enter data:");
	scanf("%d",&new_node->data);
```

```
	new_node->next=NULL;

	if(first==NULL)
	{
		first=new_node;
		*last=first;
	}
	else
	{
		(*last)->next=new_node;
		(*last)=new_node;
	}
	printf("\n Node Added Successfully.");
	return first;
}
struct Stud * InsertBeg(struct Stud *first,struct Stud *last )
{
	struct Stud *new_node=NULL;
	new_node=(struct Stud *)malloc(sizeof(struct Stud));

	printf("\n Enter data:");
	scanf("%d",&new_node->data);
	new_node->next=NULL;

	if(first==NULL)
	{
		first=new_node;
		last=first;
	}
	else
	{
		new_node->next=first;
		first=new_node;
	}
	printf("\n Node Added Successfully.");
	return first;
}

struct Stud * InsertAnywhere(struct Stud *first,struct Stud **last )
{

	struct Stud *new_node=NULL;

	int N=0,pos=0,cnt=0,insert=0;
	struct Stud *p,*temp;
	p=first;

	while(p!=NULL)			//count the number of nodes
```

```
{
	p=p->next;
	N=N+1;
}
printf("\n cnt is:%d",N);

new_node=(struct Stud *)malloc(sizeof(struct Stud));
printf("\n Enter data:");
scanf("%d",&new_node->data);
new_node->next=NULL;

if(first==NULL)			//if list empty insert 1 node at begning
{
		first=new_node;
		*last=first;
		insert=1;
}
else
{
	printf("\n Enter pogision where you have to add the nodes:");
	scanf("%d",&pos);
}

if(pos>=0 && pos<=(N+1))
{

	if(N==1)		//if only 1 node in the list
	{
		if(pos==1)		//if we have to add at first position
		{
			new_node->next=first;
			first=new_node;
			insert=1;
		}
		else if(pos==2)	//if we have to add at 2nd position
		{
			first->next=new_node;
			(*last)=new_node;
			insert=1;
		}
	}
	else		//there is multiple node in list
	{
		p=first;
		cnt=1;
		while(p!=NULL)
		{
			if(cnt==(pos-1))
```

```
                        {
                                if(pos==(N+1))     //at the end
                                {
                                        (*last)->next=new_node;
                                        (*last)=new_node;
                                        insert=1;
                                }
                                else            //at anywhere
                                {
                                        temp=p->next;
                                        p->next=new_node;
                                        new_node->next=temp;
                                        insert=1;
                                }
                        }
                        p=p->next;
                        cnt++;

                }

        }

        if(insert)
        {
                printf("\n Node Added Successfully.");
        }
    }
    else
    {
        printf("Invalid position");
    }

    return first;
}

struct Stud * DeleteBeg(struct Stud *first,struct Stud **last )
{
    struct Stud *temp;

    if(first==NULL)
    {
        printf("\n List is Empty.");
    }
    else
    {
        if(first==(*last))    //if only one node is exist
        {
                temp=first;
```

```
                free(temp);
                first=NULL;
                (*last)=NULL;
                printf("\n Deleted Successfully.");
        }
        else            //delete node from the begining
        {
                temp=first;
                first=first->next;
                free(temp);
                printf("\n Deleted Successfully.");
        }
    }
    retur  n first;
}

struct Stud * DeleteEnd(struct Stud *first,struct Stud **last )
{
    struct Stud *temp,*p;

    if(first==NULL)
    {
        printf("\n List is Empty.");
    }
    else
    {
        if(first==(*last))    //if only one node is exist
        {
                temp=first;
                free(temp);
                first=NULL;
                (*last)=NULL;
                printf("\n Deleted Successfully.");
        }
        else
        {
                p=first;
                while(p->next->next!=NULL)
                {
                        p=p->next;
                }
                temp=p->next;
                p->next=NULL;
                (*last)=p;
                free(temp);
                printf("\n Deleted Successfully.");
        }
    }
```

```
	return first;
}

struct Stud * DeleteAnywhere(struct Stud *first,struct Stud **last )
{
	struct Stud *temp,*p;
	int N=0,pos=0,cnt=0,found=0;

	p=first;
	while(p!=NULL)			//count the number of nodes
	{
		p=p->next;
		N=N+1;
	}


	if(first==NULL)
	{
		printf("\n List is Empty.");
	}

	if(N==1)	//if only one node into the list
	{
		temp=first;
		free(temp);
		first=NULL;
		(*last)=NULL;
		found=1;
	}
	else
	{
		printf("\n Enter pogision which you want to delete a node:");
		scanf("%d",&pos);
	}

	if(N!=1 && pos>0 && pos<=N)
	{
		if(pos==1)			//delete node from the begining
		{
			temp=first;
			first=first->next;
			free(temp);
			found=1;


		}
		else if(pos==N) //delete from the end
		{
```

```
                p=first;
                while(p->next->next!=NULL)
                {
                        p=p->next;
                }
                temp=p->next;
                p->next=NULL;
                (*last)=p;
                free(temp);
                found=1;

        }
        else            //delete from anywhere
        {
                p=first;
                cnt=1;
                while(p!=NULL)
                {
                        if(cnt==(pos-1))
                        {
                                temp=p->next;
                                p->next=p->next->next;
                                free(temp);

                        }
                        p=p->next;
                        cnt++;

                }
        }
        found=1;
    }


    if(found)
    {
        printf("\n Deleted Successfully.");
    }
    else
    {
        printf("\n Invalid Position..");
    }

    return first;
}
int CountNode(struct Stud *first)
{
    struct Stud *p;
```

```
	int cnt=0;

	p=first;
	while(p!=NULL)			//count the number of nodes
	{
		p=p->next;
		cnt=cnt+1;
	}
	printf("\n Total number of nodes is:%d",cnt);
	return cnt;
}


void Display(struct Stud *first)
{
	struct Stud *p;
	p=first;

	while(p!=NULL)
	{
		printf("%d ",p->data);
		p=p->next;
	}
}
void main()
{
	struct Stud *first=NULL;
	struct Stud *last=NULL;

	int ch,count;
	char choice='n';

	printf("\n---------------");
	printf("\n 1.Create List");
	printf("\n 2.Insert End");
	printf("\n 3.Insert Begning ");
	printf("\n 4.Insert Anywhere");
	printf("\n 5.Delete Begning");
	printf("\n 6.Delete End");
	printf("\n 7.Delete Anywhere");
	printf("\n 8.Count Nodes");
	printf("\n 9.Display");
	printf("\n---------------");

	do
	{
		printf("\n Enter Your choice:");
		scanf("%d",&ch);
```

```
            switch(ch)
            {
                  case 1:
                              first=CreateList(first,&last);
                              break;
                  case 2:
                              first=InsertEnd(first,&last);
                              break;
                  case 3:
                              first=InsertBeg(first,last);
                              break;
                  case 4:
                              first=InsertAnywhere(first,&last);
                              break;
                  case 5:
                              first=DeleteBeg(first,&last);
                              break;
                  case 6:
                              first=DeleteEnd(first,&last);
                              break;
                  case 7:first=DeleteAnywhere(first,&last);
                              break;
                  case 8:
                              count=CountNode(first);
                              break;
                  case 9:
                              Display(first);
                              break;
            }
            printf("\n Do You want to continue:");
            scanf(" %c",&choice);


      }while(choice=='y' || choice=='Y');


      getch();
}

========
OUTPUT
========

---------------
 1.Create List
 2.Insert End
 3.Insert Begning
 4.Insert Anywhere
```

```
 5.Delete Begning
 6.Delete End
 7.Delete Anywhere
 8.Count Nodes
 9.Display
---------------
 Enter Your choice:y

 Do You want to continue:
 Enter Your choice:1

 Enter number of nodes you want to create:3

 Enter data:10

 Enter data:20

 Enter data:30

 List Created Successfully.
 Do You want to continue:y

 Enter Your choice:8

 Total number of nodes is:3
 Do You want to continue:y

 Enter Your choice:9
10 20 30
 Do You want to continue:y

 Enter Your choice:2

 Enter data:89

 Node Added Successfully.
 Do You want to continue:y

 Enter Your choice:3

 Enter data:4

 Node Added Successfully.
 Do You want to continue:y

 Enter Your choice:9
4 10 20 30 89
 Do You want to continue:y
```

```
 Enter Your choice:4

 cnt is:5
 Enter data:3

 Enter pogision where you have to add the nodes:3

 Node Added Successfully.
 Do You want to continue:y

 Enter Your choice:9
4 10 3 20 30 89
 Do You want to continue:y

 Enter Your choice:5

 Deleted Successfully.
 Do You want to continue:y

 Enter Your choice:9
10 3 20 30 89
 Do You want to continue:y

 Enter Your choice:6

 Deleted Successfully.
 Do You want to continue:y

 Enter Your choice:9
10 3 20 30
 Do You want to continue:y

 Enter Your choice:7

 Enter pogision which you want to delete a node:2

 Deleted Successfully.
 Do You want to continue:y

 Enter Your choice:9
10 20 30
 Do You want to continue:n


======================
6. Stack with Array
======================
```

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
int top = -1;


void Push(int stack[],int size,int key)
{
	if(top>=(size-1))
	{
		printf("\n Overflow..");
	}
	else
	{
		top=top+1;
		stack[top]=key;
	}
}

int Pop(int stack[],int size)
{
	int data;

	if(top==-1)
	{
		printf("\n Underflow...");
		//data=-1;
		return NULL;
	}
	else
	{
		data=stack[top];
		top=top-1;
	}

	return data;
}

void Display(int stack[],int size)
{
	int i;

	if(top>-1)
	{
		for(i=top;i>=0;i--)
		{
			printf("%d ",stack[i]);
```

```
		}
	}
	else
	{
		printf("\n Stack is Empty...");
	}
}

void main()
{
	int stack[10],ch,size=3,key,res;
	char choice='n';

	printf("\n---------");
	printf("\n 1.push");
	printf("\n 2.pop");
	printf("\n 3.Display");
	printf("\n---------");
	do
	{
		printf("\n Enter Your Choice:");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1: printf("\n Enter Key:");
				scanf("%d",&key);
				Push(stack,size,key);
				break;
			case 2:
				res=Pop(stack,size);
				printf("\n Element is:%d",res);
				break;
			case 3:
				Display(stack,size);
				break;
		}
		printf("\n Do you want to continue..");
		scanf(" %c",&choice);

	}while(choice=='Y' || choice=='y');

	getch();
}
```

==========
OUTPUT
==========

```
---------
 1.push
 2.pop
 3.Display
---------
 Enter Your Choice:1

 Enter Key:10

 Do you want to continue..y

 Enter Your Choice:1

 Enter Key:20

 Do you want to continue..y

 Enter Your Choice:1

 Enter Key:30

 Do you want to continue..y

 Enter Your Choice:1

 Enter Key:40

 Overflow..
 Do you want to continue..y

 Enter Your Choice:3
30 20 10
 Do you want to continue..y

 Enter Your Choice:2

 Element is:30
 Do you want to continue..y

 Enter Your Choice:2

 Element is:20
 Do you want to continue..y

 Enter Your Choice:2

 Element is:10
```

```
Do you want to continue..y

Enter Your Choice:2

Underflow...
Element is:0
Do you want to continue..
```

==========
7. Stack with Link List
==========

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

struct Stack
{
	int data;
	struct Stack *next;
};

struct Stack *first;


void Push(int key)
{
	struct Stack *new_node=NULL;
	new_node=(struct Stack *)malloc(sizeof(struct Stack));
	new_node->data=key;
	if(first==NULL)			//if list is empty
	{
		new_node->next=NULL;
		first=new_node;
	}
	else
	{
		new_node->next=first;
		first=new_node;
	}
}

int Pop()
```

```
{
	struct Stack *temp;
	int data;

	if(first==NULL)
	{
		printf("\n Underflow..");
		return NULL;
	}
	else
	{
		temp=first;
		first=first->next;
		data=temp->data;
		free(temp);
		return data;
	}

}

void Display()
{
	struct Stack *p;
	p=first;

	while(p!=NULL)
	{
		printf("%d ",p->data);
		p=p->next;
	}
}
void main()
{

	int ch,count,key,res;
	char choice='n';

	printf("\n---------------");
	printf("\n 1.Push ");
	printf("\n 2.Pop");
	printf("\n 3.Display");
	printf("\n---------------");

	do
	{
		printf("\n Enter Your choice:");
		scanf("%d",&ch);
		switch(ch)
```

```
            {
                  case 1:       printf("\n Enter Key:");
                                scanf("%d",&key);

                                Push(key);
                                break;
                  case 2:
                                res=Pop();
                                printf("\n Element is:%d",res);
                                break;
                  case 3:
                                Display();
                                break;
            }
            printf("\n Do You want to continue:");
            scanf(" %c",&choice);

      }while(choice=='y' || choice=='Y');


      getch();
}
```

```
=========
OUTPUT
=========

---------------
 1.Push
 2.Pop
 3.Display
---------------
 Enter Your choice:1

 Enter Key:10

 Do You want to continue:y

 Enter Your choice:1

 Enter Key:20

 Do You want to continue:y

 Enter Your choice:1

 Enter Key:30
```

```
 Do You want to continue:y

 Enter Your choice:3
30 20 10
 Do You want to continue:y

 Enter Your choice:2

 Element is:30
 Do You want to continue:y

 Enter Your choice:3
20 10
 Do You want to continue:y

 Enter Your choice:2

 Element is:20
 Do You want to continue:y

 Enter Your choice:2

 Element is:10
 Do You want to continue:y

 Enter Your choice:2

 Underflow..
 Element is:0
 Do You want to continue:
```

==========
8. Factorial with Recursion
==========

```
#include<stdio.h>
#include<conio.h>

int fact(int n)
{
	if(n==1)
		return 1;
	else
		return n*fact(n-1);
}
```

```
void main()
{
	int no,res;
	printf("\n Enter Number:");
	scanf("%d",&no);

	res=fact(no);

	printf("\n %d",res);
	getch();

}
```

```
=======
OUTPUT
=======

Enter Number:5

 120
```

```
==================================
9. Fibonacci Series with Recursion
==================================
#include<stdio.h>
#include<conio.h>

int Fibo(int n)
{
	if(n==0)
		return 0;
	else if(n==1)
		return 1;
	else
		return (Fibo(n-1) + Fibo(n-2));
}
void main()
{
	int n,res;
	printf("\n Enter Number:");
	scanf("%d",&n);

	for(int i=0;i<n;i++)
	{
		printf("%d ",Fibo(i));
```

```
	}

	getch();

}
```

```
========
OUTPUT
========

Enter Number:10
0 1 1 2 3 5 8 13 21 34
```

```
==========================================
10. Any other Series with Recursion(eg. sine series)
==========================================

#include<stdio.h>
#include<conio.h>
#include<math.h>
double sinx(double,int);
int fact(int);
void main()
{
	int n;
	double x,ans;
	printf("Enter value of x: ");
	scanf("%lf",&x);
	printf("Enter how many terms you want to calculate: ");
	scanf("%d",&n);
	ans=sinx(x,n);
	printf("\nsin(%.2f) = %.4f",x,ans);
	getch();
}

double sinx(double x,int n)
{
	if(n==1)
		return x;
	else if(n%2==0)
			return (sinx(x,n-1) - (pow(x,(2*n-1))/(fact(2*n-1))));
	else
		return (sinx(x,n-1) + (pow(x,(2*n-1))/(fact(2*n-1))));
}
```

```
int fact(int num)
{
	if(num==2)
		return 2;
	else
	{
		num=num*fact(num-1);
		return num;
	}
}
```

```
=========
OUTPUT
=========

Enter value of x: 2
Enter how many terms you want to calculate: 3

sin(2.00) = 0.9333
```

```
==========================================
11. Equation validation with brackets  using Array
==========================================

#include<stdio.h>
#include<conio.h>
#include<malloc.h>
int top = -1;

void Push(char stack[],int size,char key)
{
	if(top>=(size-1))
	{
		printf("\n Overflow..");
	}
	else
	{
		top=top+1;
		stack[top]=key;
	}
}

char Pop(char stack[],int size)
{
```

```
	int data;

	if(top==-1)
	{
		printf("\n Underflow...");
		return NULL;
	}
	else
	{
		data=stack[top];
		stack[top]=NULL;
		top=top-1;
	}

	return data;
}

void Display(char stack[],int size)
{
	int i;

	if(top>-1)
	{
		for(i=top;i>=0;i--)
		{
			printf("%c ",stack[i]);
		}
	}
	else
	{
		printf("\n Stack is Empty...");
	}
}

char stackPop(char stack[])
{
	if(top!=0)
	{
		return stack[top];
	}
	else
		return stack[top];
}
void main()
{
	char str[20],stack[20],key,prv_char;

	int i,cnt=0,size,invalid=0;
```

```
	printf("\n Enter String:");
	scanf("%s",str);

	for(int i=0;str[i]!='\0';i++)
	{
		cnt++;
	}
	size=cnt;
	for(int i=0;str[i]!='\0';i++)
	{

		key=str[i];

		if(key=='(' || key=='{' || key=='[')
		{
			Push(stack,size,key);
		}
		else if(key==')' || key=='}' || key==']')
		{
			prv_char=stackPop(stack);

			if(prv_char=='{' && key=='}' || prv_char=='(' && key==')'
|| prv_char=='[' && key==']')
			{
				Pop(stack,size);
			}
			else
			{
				printf("\n Invalid Expression...");
				invalid=1;
				break;
			}
		}

	}
	if(!invalid)
	{
		printf("Bracket are Balanced.. :)");
	}
	getch();
}
```

```
=======
OUTPUT
=======

 Enter String:{[]()}
Bracket are Balanced.. :)
```

==========
11. Equation validation with brackets using Link List
==========

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

struct Queue
{
	char data;
	struct Queue *next;
};
void Push(struct Queue **front,struct Queue **rear, char key)
	//Insert Beg
{
	struct Queue *new_node=NULL;
	new_node=(struct Queue *)malloc(sizeof(struct Queue));

	new_node->data=key;
	new_node->next=NULL;

	if((*front)==NULL)
	{
		(*front)=new_node;
		(*rear)=(*front);
	}
	else
	{
		new_node->next=(*front);
		(*front)=new_node;
	}

}

char stackPop(struct Queue *front)
{
	 return front->data;
}
char Pop(struct Queue **front,struct Queue **rear )	//delete beg
{
	struct Queue *temp;
	char key;
	if((*front)==NULL)
```

```
	{
		printf("\n List is Empty.");
	}
	else
	{
		if((*front)==(*rear))	//if only one node is exist
		{
			temp=(*front);
			key=temp->data;
			free(temp);
			(*front)=NULL;
			(*rear)=NULL;

		}
		else		//delete node from the begining
		{
			temp=(*front);
			(*front)=(*front)->next;
			key=temp->data;
			free(temp);

		}
	}
	return key;
}


void Display(struct Queue *front)
{
	struct Queue *p;
	p=front;

	if(p==NULL)
	{
		printf("\n Queue is empty..");
	}
	{
		while(p->next!=front)
		{
			printf("%d ",p->data);
			p=p->next;
		}
		printf("%d ",p->data);
	}
}


void main()
```

```
{
	struct Queue *front=NULL;
	struct Queue *rear=NULL;

	char str[20],key,prv_char;

	int i,cnt=0,size,invalid=0;
	printf("\n Enter String:");
	scanf("%s",str);

	for(int i=0;str[i]!='\0';i++)
	{
		cnt++;
	}
	size=cnt;
	for(int i=0;str[i]!='\0';i++)
	{

		key=str[i];

		if(key=='(' || key=='{' || key=='[')
		{
			Push(&front,&rear,key);
		}
		else if(key==')' || key=='}' || key==']')
		{
			prv_char=stackPop(front);

			if(prv_char=='{' && key=='}' || prv_char=='(' && key==')'
|| prv_char=='[' && key==']')
			{
				Pop(&front,&rear);
			}
			else
			{
				printf("\n Invalid Expression...");
				invalid=1;
				break;
			}
		}

	}
	if(!invalid)
	{
		printf("Bracket are Balanced.. :)");
	}
	getch();
}
```

```
=======
OUTPUT
=======

 Enter String:{[]}[]
Bracket are Balanced.. :)
```

```
=========================================
12. Infix to Postfix Conversion Using Link List
=========================================
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
int top = -1;

void Push(char stack[],int size,char key)
{
	if(top>=(size-1))
	{
		printf("\n Overflow..");
	}
	else
	{
		top=top+1;
		stack[top]=key;
	}
}

char Pop(char stack[],int size)
{
	int data;

	if(top==-1)
	{
		printf("\n Underflow...");
		return NULL;
	}
	else
	{

		data=stack[top];
		stack[top]=NULL;
		top=top-1;
```

```
	}

	return data;
}

void Display(char stack[],int size)
{
	int i;

	if(top>-1)
	{
		for(i=top;i>=0;i--)
		{
			printf("%c ",stack[i]);
		}
	}
	else
	{
		printf("\n Stack is Empty...");
	}
}

char stackPop(char stack[])
{
	if(top!=-1)
	{
		return stack[top];
	}
	else
	{
		return stack[0];
	}
}

bool balance_bracket(char str[],char stack[],int size)
{
	char key,prv_char;

	int i,cnt=0,invalid=0;



	for(int i=0;str[i]!='\0';i++)
	{

		key=str[i];
```

```
		if(key=='(' || key=='{' || key=='[')
		{
			Push(stack,size,key);
		}
		else if(key==')' || key=='}' || key==']')
		{
			prv_char=stackPop(stack);

			if(prv_char=='{' && key=='}' || prv_char=='(' && key==')'
|| prv_char=='[' && key==']')
			{
				Pop(stack,size);
			}
			else
			{
				printf("\n Invalid Expression...");
				invalid=1;
				break;
			}
		}

	}

	if(invalid)
	{
		return 0;
	}
	else
		return 1;

}

bool isOperand(char ch)
{

	if(ch>=65 && ch<=90 || ch>=97 && ch<=122)
	{
		return 1;
	}
	else
		return 0;
}
bool isOperator(char ch)
{
	if(ch=='+' || ch=='-' || ch=='/' || ch=='*')
		return 1;
	else
		return 0;
```

```
}

int Precedence(char ch)
{
	if(ch=='+' || ch=='-')
		return 1;
	else if(ch=='*' || ch=='/')
		return 2;
	else if(ch=='(')
		return 0;
	else
		return -1;
}
void infix_to_postfix(char stack[],char str[],int size_str)
{
	int i;
	bool res;
	char stack_val,stack_2[20],ch;

	for(int i=0;str[i]!='\0';i++)
	{
		if(str[i]=='(')
		{
			Push(stack,size_str,str[i]);
		}
		else if(isOperand(str[i]))	//if this is operand then display
		{
			printf("%c",str[i]);
		}
		else if(isOperator(str[i])) //if this is operator then
		{
			//stack_val=stackPop(stack);		// take the stck value
			while(isOperator(str[i]) &&
(Precedence(stack_val=stackPop(stack)) >= Precedence(str[i])))
	//check te priority
			{
				ch=Pop(stack,size_str);
				printf("%c",ch);
			}

			if(str[i]!='(')
			{
				Push(stack,size_str,str[i]);
			}
			else
				Push(stack,size_str,str[i]);
		}
		else if(str[i]==')')
```

```
                {
                        ch=Pop(stack,size_str);
                        while(ch!='(')                  // take the stck value)
                        {
                                printf("%c",ch);
                                ch=Pop(stack,size_str);
                        }
                }
        }
        while(top!=-1)
        {
                ch=Pop(stack,size_str);
                printf("%c",ch);
        }


}
void main()
{
        char str[20],stack[20],stack_2[20],key,prv_char;

        int i,cnt=0,size,invalid=0;
        printf("\n Enter String:");
        scanf("%s",str);

        for(int i=0;str[i]!='\0';i++)
        {
                cnt++;
        }
        size=cnt;
        if(balance_bracket(str,stack,size))
        {
                infix_to_postfix(stack,str,size);
        }
        else
        {
                printf("Invalid Expression...Try Again.");
        }
        getch();
}
```

==========
OUTPUT
==========

```
 Enter String:(a+b)*(a}
```

```
 Invalid Expression...Invalid Expression...Try Again.


 Enter String:A+B*C+D
ABC*+D+


 Enter String:(A+B)*(C+D)
AB+CD+*


 Enter String:A*B+C*D
AB*CD*+


 Enter String:A+B+C+D
AB+C+D+
```

==========
13. Postfix Expression Evaluation
==========

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
int top = -1;

void Push(float stack[],int size,float key)
{
	if(top>=(size-1))
	{
		printf("\n Overflow..");
	}
	else
	{
		top=top+1;
		stack[top]=key;

	}
}

float Pop(float stack[],int size)
{
	float data;

	if(top==-1)
	{
		printf("\n Underflow...");
```

```
		return NULL;
	}
	else
	{

		data=stack[top];
		stack[top]=NULL;
		top=top-1;

	}

	return data;
}

bool isOperand(char ch)
{
	if(ch>=65 && ch<=90 || ch>=97 && ch<=122)
	{
		return 1;
	}
	else
		return 0;
}
bool isOperator(char ch)
{
	if(ch=='+' || ch=='-' || ch=='/' || ch=='*')
		return 1;
	else
		return 0;
}

void Evaluation_postfix(float stack[],char str[],int size_str)
{
	float value,value2,res2,res1;
	int i,result;

	char op,stack_val,ch;

	for(int i=0;str[i]!='\0';i++)
	{
		if(isOperand(str[i]))
		{

			printf("\n Enter Value for %c:",str[i]);
			scanf("%f",&value);


			Push(stack,size_str,value);
```

```
		}
		else if(isOperator(str[i]))
		{
			op=str[i];
			res2=Pop(stack,size_str);
			res1=Pop(stack,size_str);

			if(op=='+')
				result=res1 + res2;
			else if(op=='-')
				result=res1 - res2;
			else if(op=='*')
				result=res1 * res2;
			else if(op=='/')
				result=res1 / res2;

			Push(stack,size_str,result);
		}
	}

	printf("Evalution is:%.2f",Pop(stack,size_str));
}

void main()
{
	char str[20],key,prv_char;

	float stack[20];
	int i,cnt=0,size,invalid=0;
	printf("\n Enter String:");
	scanf("%s",str);

	for(int i=0;str[i]!='\0';i++)
	{
		cnt++;
	}
	size=cnt;

	Evaluation_postfix(stack,str,size);


	getch();
}
```

==========

OUTPUT

```
==========

 Enter String:AB+CD+/

 Enter Value for A:7

 Enter Value for B:8

 Enter Value for C:3

 Enter Value for D:2
Evalution is:3.00

==========
OUTPUT
==========

 Enter String:AB+C/D*E+

 Enter Value for A:100

 Enter Value for B:200

 Enter Value for C:2

 Enter Value for D:5

 Enter Value for E:7
Evalution is:757.00
```

```
==================
14. Queue with Array
==================

#include<stdio.h>
#include<conio.h>


int front =-1;
int rear = -1;

void insert(int key,int size,int queue[])
```

```
{
	if(rear==(size-1))
	{
		printf("\n Overflow");
	}
	else
	{
		if(front==-1)
		{
			front=0;
		}
		rear +=1;
		queue[rear]=key;
	}
}

void del(int queue[])
{
	if(front==-1)
	{
		printf("\n underflow");

	}
	else if(front==rear)
	{
		printf("\n Deleted item is:%d",queue[front]);
		front=-1;
		rear=-1;
	}
	else
	{
		printf("\n Deleted item is:%d",queue[front]);
		front=front+1;
	}
}

void Display(int queue[])
{
	int i;

	if(front ==-1)
	{
		printf("Queue is Empty");
	}
	else
	{

		for(i=front;i<=rear;i++)
```

```
		{
			printf("%d ",queue[i]);
		}
	}

}



void main()
{

	int queue[10],ch,count,key,res,size=5;
	char choice='n';


	printf("\n---------------");
	printf("\n 1.insert ");
	printf("\n 2.delete");
	printf("\n 3.Display");
	printf("\n---------------");

	do
	{
		printf("\n Enter Your choice:");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1:		printf("\n Enter Key:");
					scanf("%d",&key);

					insert(key,size,queue);
					break;
			case 2:
					del(queue);
					break;
			case 3:
					Display(queue);
					break;

		}
		printf("\n Do You want to continue:");
		scanf(" %c",&choice);

	}while(choice=='y' || choice=='Y');
```

```
        getch();
}
```

```
=======
OUTPUT
=======


---------------
 1.insert
 2.delete
 3.Display
---------------
 Enter Your choice:1

 Enter Key:1

 Do You want to continue:y

 Enter Your choice:1

 Enter Key:2

 Do You want to continue:y

 Enter Your choice:3
1 2
 Do You want to continue:y

 Enter Your choice:2

 Deleted item is:1
 Do You want to continue:y

 Enter Your choice:2

 Deleted item is:2
 Do You want to continue:y

 Enter Your choice:3
Queue is Empty
 Do You want to continue:n



========================
15. Queue with Linked List
========================
```

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
static int count =0;
int size=2;

struct Queue
{
	int data;
	struct Queue *next;
};
struct Queue * Insert(struct Queue *front,struct Queue **rear )
	//Insert End
{
	if(count>size-1)
	{
		printf("\n Overflow");

	}
	else
	{
		struct Queue *new_node=NULL;
		new_node=(struct Queue *)malloc(sizeof(struct Queue));

		printf("\n Enter data:");
		scanf("%d",&new_node->data);
		new_node->next=NULL;

		if(front==NULL)
		{
			front=new_node;
			*rear=front;
			count++;
		}
		else
		{
			(*rear)->next=new_node;
			(*rear)=new_node;
			count++;
		}
		printf("\n Node Added Successfully.");
	}
	return front;
}

struct Queue * Dele(struct Queue *front,struct Queue **rear )		//Delete
from begining
{
```

```
	struct Queue *temp;

	if(front==NULL)				//hear front is front and rear is rear
	{
		printf("\n Queue is Empty..Underflow");
	}
	else
	{
		if(front==(*rear))  //if only one node is exist
		{
			temp=front;
			printf("\n Deleted Item is: %d",temp->data);
			free(temp);
			front=NULL;
			(*rear)=NULL;
			count--;

		}
		else		//delete node from the begining
		{
			temp=front;
			printf("\n Deleted Item is: %d",temp->data);
			front=front->next;
			free(temp);
			count--;
		}
	}
	return front;
}


void Display(struct Queue *front)
{
	struct Queue *p;
	p=front;

	while(p!=NULL)
	{
		printf("%d ",p->data);
		p=p->next;
	}
}
void main()
{
	struct Queue *front=NULL;
	struct Queue *rear=NULL;

	int ch,count;
```

```
        char choice='n';

        printf("\n---------------");
        printf("\n 1.Insert");
        printf("\n 2.Delete");
        printf("\n 3.Display");
        printf("\n---------------");

        do
        {
                printf("\n Enter Your choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                        front=Insert(front,&rear);
                                        break;
                        case 2:
                                        front=Dele(front,&rear);
                                        break;
                        case 3:
                                        Display(front);
                                        break;
                }
                printf("\n Do You want to continue:");
                scanf(" %c",&choice);

        }while(choice=='y' || choice=='Y');


        getch();
}
```

==========
OUTPUT
==========

```
---------------
 1.Insert
 2.Delete
 3.Display
---------------
 Enter Your choice:1

 Enter data:20

 Node Added Successfully.
 Do You want to continue:y
```

```
 Enter Your choice:1

 Enter data:45

 Node Added Successfully.
 Do You want to continue:y

 Enter Your choice:1

 Overflow
 Do You want to continue:y

 Enter Your choice:3
20 45
 Do You want to continue:y

 Enter Your choice:2

 Deleted Item is: 20
 Do You want to continue:y

 Enter Your choice:2

 Deleted Item is: 45
 Do You want to continue:y

 Enter Your choice:2

 Queue is Empty..Underflow
 Do You want to continue:y

 Enter Your choice:2

 Queue is Empty..Underflow
 Do You want to continue:n
```

==========

16. Circular Queue with Array

==========

```
#include<stdio.h>
#include<conio.h>
int size =3;
static int count =0;
```

```
int front =-1;
int rear = -1;

void insert(int key,int size,int queue[])
{
	if(count>(size-1))
	{
		printf("\n Overflow");
	}
	else if(front==0 && rear==(size-1))
	{
		printf("\n Overflow");
	}
	else if((rear+1) ==front)
	{
		printf("\n Overflow");
	}
	else if(front!=0 && rear==(size - 1))
	{
		if(front == -1)
		{
			front=0;
		}
			rear=0;
			queue[rear]=key;
		count++;
	}
	else
	{
		if(front == -1)
		{
			front=0;
		}
		rear +=1;
		queue[rear]=key;
		count++;
	}
}

void del(int queue[])
{
	if(front==-1)
	{
		printf("\n underflow");

	}
	else if(front==rear)
```

```
	{
		printf("\n Deleted item is:%d",queue[front]);
		front=-1;
		rear=-1;
		count--;
	}
	else if(front == (size-1) && front>rear)
	{

		printf("\n Deleted item is:%d",queue[front]);
		front=0;
		count--;
	}
	else
	{
		printf("\n Deleted item is:%d",queue[front]);
		front+=1;
		count--;
	}
}

void Display(int queue[])
{
	int i;

	if(front ==-1)
	{
		printf("Queue is Empty");
	}
	else
	{
		if( front <= rear )
		{
			for(i=front;i<=rear;i++)
			{
				printf("%d ",queue[i]);
			}
		}
		else
		{
			for(i=front;i<size;i++)
			{
				printf("%d ",queue[i]);
			}

			for(i=0;i<=rear;i++)
			{
				printf("%d ",queue[i]);
```

```
                }
            }
        }

}


void main()
{

        int queue[10],ch,count,key,res;
        char choice='n';


        printf("\n---------------");
        printf("\n 1.insert ");
        printf("\n 2.delete");
        printf("\n 3.display");
        printf("\n---------------");

        do
        {
            printf("\n Enter Your choice:");
            scanf("%d",&ch);
            switch(ch)
            {
                case 1:     printf("\n Enter Key:");
                            scanf("%d",&key);

                            insert(key,size,queue);
                            break;
                case 2:
                            del(queue);
                            break;
                case 3: Display(queue);
                            break;

            }
            printf("\n Do You want to continue:");
            scanf(" %c",&choice);

        }while(choice=='y' || choice=='Y');


        getch();
}

========
```

OUTPUT

```
========
---------------
 1.insert
 2.delete
 3.display
---------------
 Enter Your choice:1

 Enter Key:10

 Do You want to continue:y

 Enter Your choice:1

 Enter Key:20

 Do You want to continue:y

 Enter Your choice:1

 Enter Key:30

 Do You want to continue:y

 Enter Your choice:3
10 20 30
 Do You want to continue:y

 Enter Your choice:2

 Deleted item is:10
 Do You want to continue:y

 Enter Your choice:2

 Deleted item is:20
 Do You want to continue:y

 Enter Your choice:3
30
 Do You want to continue:y

 Enter Your choice:2

 Deleted item is:30
 Do You want to continue:y
```

```
 Enter Your choice:3
Queue is Empty
 Do You want to continue:n
```

```
================================
17. Circular Queue with Linked List
================================
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

struct Queue
{
	int data;
	struct Queue *next;
};
struct Queue * Insert(struct Queue *front,struct Queue **rear )
	//Insert End
{
	struct Queue *new_node=NULL;
	new_node=(struct Queue *)malloc(sizeof(struct Queue));

	printf("\n Enter data:");
	scanf("%d",&new_node->data);
	new_node->next=NULL;

	if(front==NULL)
	{
		front=new_node;
		*rear=front;
	}
	else
	{
		(*rear)->next=new_node;
		(*rear)=new_node;
		new_node->next=front;
	}

	return front;
}

struct Queue * Dele(struct Queue *front,struct Queue **rear )		//Delete
from begining
{
```

```
	struct Queue *temp;

	if(front==NULL)				//hear front is front and rear is rear
	{
		printf("\n Queue is Empty..Underflow");
	}
	else
	{
		if(front==(*rear))  //if only one node is exist
		{
			temp=front;
			printf("\n Deleted Item is: %d",temp->data);
			free(temp);
			front=NULL;
			(*rear)=NULL;

		}
		else		//delete node from the begining
		{
			temp=front;
			printf("\n Deleted Item is: %d",temp->data);
			front=front->next;
			free(temp);
			(*rear)->next=front;
		}
	}
	return front;
}


void Display(struct Queue *front)
{
	struct Queue *p;
	p=front;

	if(p==NULL)
	{
		printf("\n Queue is empty..");
	}
	{
		while(p->next!=front)
		{
			printf("%d ",p->data);
			p=p->next;
		}
		printf("%d ",p->data);
	}
}
```

```
void main()
{
	struct Queue *front=NULL;
	struct Queue *rear=NULL;

	int ch,count;
	char choice='n';

	printf("\n---------------");
	printf("\n 1.Insert");
	printf("\n 2.Delete");
	printf("\n 3.Display");
	printf("\n---------------");

	do
	{
		printf("\n Enter Your choice:");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1:
					front=Insert(front,&rear);
					break;
			case 2:
					front=Dele(front,&rear);
					break;
			case 3:
					Display(front);
					break;
		}
		printf("\n Do You want to continue:");
		scanf(" %c",&choice);

	}while(choice=='y' || choice=='Y');


	getch();
}
```

```
========
OUTPUT
========

---------------
 1.Insert
 2.Delete
 3.Display
---------------
```

```
Enter Your choice:1

Enter data:10

Do You want to continue:y

Enter Your choice:1

Enter data:20

Do You want to continue:y

Enter Your choice:1

Enter data:30

Do You want to continue:y

Enter Your choice:3
10 20 30
Do You want to continue:y

Enter Your choice:2

Deleted Item is: 10
Do You want to continue:y

Enter Your choice:2

Deleted Item is: 20
Do You want to continue:y

Enter Your choice:3
30
Do You want to continue:y

Enter Your choice:2

Deleted Item is: 30
Do You want to continue:y

Enter Your choice:2

Queue is Empty..Underflow
Do You want to continue:n
```

```
================================================
=
18. Circular Linked List(All Insertions, All Deletions & Display)
================================================
==

#include<stdio.h>
#include<conio.h>
#include<malloc.h>


struct Stud
{
	int data;
	struct Stud *next;
};

struct Stud * InsertEnd(struct Stud *first,struct Stud **last )
{
	struct Stud *new_node=NULL;
	new_node=(struct Stud *)malloc(sizeof(struct Stud));

	printf("\n Enter data:");
	scanf("%d",&new_node->data);
	new_node->next=NULL;

	if(first==NULL)
	{
		first=new_node;
		*last=first;
	//	new_node->next=first;
	}
	else
	{
		(*last)->next=new_node;
		(*last)=new_node;
		new_node->next=first;
	}

	return first;
}

struct Stud * InsertBeg(struct Stud *first,struct Stud *last )
{
	struct Stud *new_node=NULL;
	new_node=(struct Stud *)malloc(sizeof(struct Stud));

	printf("\n Enter data:");
```

```
	scanf("%d",&new_node->data);
	new_node->next=NULL;

	if(first==NULL)
	{
		first=new_node;
		last=first;
	//	last->next=first;

	}
	else
	{

		new_node->next=first;
		first=new_node;
		last->next=first;
	}

	return first;
}
struct Stud * DeleteBeg(struct Stud *first,struct Stud **last )
{
	struct Stud *temp;

	if(first==NULL)
	{
		printf("\n Stud is Empty..Underflow");
	}
	else
	{
		if(first==(*last))    //if only one node is exist
		{
			temp=first;
			printf("\n Deleted Item is: %d",temp->data);
			free(temp);
			first=NULL;
			(*last)=NULL;

		}
		else           //delete node from the begining
		{
			temp=first;
			printf("\n Deleted Item is: %d",temp->data);
			first=first->next;
			free(temp);
			(*last)->next=first;
		}
	}
```

```
	return first;
}


struct Stud * DeleteEnd(struct Stud *first,struct Stud **last )
{
	struct Stud *temp,*p;

	if(first==NULL)
	{
		printf("\n List is Empty.");
	}
	else
	{
		if(first==(*last))    //if only one node is exist
		{
			temp=first;
			free(temp);
			first=NULL;
			(*last)=NULL;

		}
		else
		{
			p=first;
			while(p->next->next!=first)
			{
				p=p->next;
			}
			temp=p->next;
			printf("\n Deleted Item is: %d",temp->data);
			p->next=first;
			(*last)=p;
			free(temp);
		}
	}
	return first;
}

void Display(struct Stud *first,struct Stud *last)
{
	struct Stud *p;
	p=first;

	while(p->next!=first)
	{
		printf("%d ",p->data);
		p=p->next;
```

```
	}
	printf("%d ",p->data);
}
void main()
{
	struct Stud *first=NULL;
	struct Stud *last=NULL;

	int ch,count;
	char choice='n';

	printf("\n---------------");
	printf("\n 1.Insert End");
	printf("\n 2.Insert Begning ");
	printf("\n 3.Delete Begning");
	printf("\n 4.Delete End");
	printf("\n 5.Display");
	printf("\n---------------");

	do
	{
		printf("\n Enter Your choice:");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1:
					first=InsertEnd(first,&last);
					break;
			case 2:
					first=InsertBeg(first,last);
					break;
			case 3:
					first=DeleteBeg(first,&last);
					break;
			case 4:
					first=DeleteEnd(first,&last);
					break;
			case 5:
					Display(first,last);
					break;
		}
		printf("\n Do You want to continue:");
		scanf(" %c",&choice);


	}while(choice=='y' || choice=='Y');
```

```
        getch();
}
```

```
=========
OUTPUT
=========

---------------
 1.Insert End
 2.Insert Begning
 3.Delete Begning
 4.Delete End
 5.Display
---------------
 Enter Your choice:1

 Enter data:10

 Do You want to continue:y

 Enter Your choice:2

 Enter data:20

 Do You want to continue:y

 Enter Your choice:1

 Enter data:40

 Do You want to continue:y

 Enter Your choice:2

 Enter data:60

 Do You want to continue:y

 Enter Your choice:5
60 20 10 40
 Do You want to continue:y

 Enter Your choice:3

 Deleted Item is: 60
 Do You want to continue:y
```

```
 Enter Your choice:5
20 10 40
 Do You want to continue:y

 Enter Your choice:4

 Deleted Item is: 40
 Do You want to continue:y

 Enter Your choice:5
20 10
 Do You want to continue:n
```

ASSIGNMENT -2

==========
1. Doubly Link List.
==========

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

struct Stud
{
	int data;
	struct Stud *next,*prev;
};



struct Stud * CreateList(struct Stud *first,struct Stud **last )
{
	int n;
	printf("\n Enter number of nodes you want to create:");
	scanf("%d",&n);

	while(n!=0)
	{

		struct Stud *new_node=NULL;
		struct Stud *temp=NULL;
```

```
            new_node=(struct Stud *)malloc(sizeof(struct Stud));

            printf("\n Enter data:");
            scanf("%d",&new_node->data);
            new_node->next=NULL;
            new_node->prev=NULL;

            if(first==NULL)
            {
                  first=new_node;
                  new_node->prev=NULL;
                  new_node->next=NULL;
                  *last=first;
            }
            else
            {
                  temp=(*last);
                  (*last)->next=new_node;
                  new_node->prev=temp;
                  new_node->next=NULL;
                  (*last)=new_node;
            }
            n--;
      }
      printf("\n List Created Successfully.");
      return first;
}

struct Stud * InsertEnd(struct Stud *first,struct Stud **last )
{
      struct Stud *new_node=NULL;
      struct Stud *temp=NULL;

      new_node=(struct Stud *)malloc(sizeof(struct Stud));

      printf("\n Enter data:");
      scanf("%d",&new_node->data);
      new_node->next=NULL;
      new_node->prev=NULL;

      if(first==NULL)
      {
            first=new_node;
            new_node->prev=NULL;
            new_node->next=NULL;
            *last=first;
      }
```

```
	else
	{
		temp=(*last);
		(*last)->next=new_node;
		new_node->prev=temp;
		new_node->next=NULL;
		(*last)=new_node;
	}	printf("\n Node Added Successfully.");
	return first;
}

struct Stud * InsertBeg(struct Stud *first,struct Stud *last )
{
	struct Stud *new_node=NULL;
	struct Stud *temp=NULL;

	new_node=(struct Stud *)malloc(sizeof(struct Stud));

	printf("\n Enter data:");
	scanf("%d",&new_node->data);

	new_node->next=NULL;
	new_node->prev=NULL;

	if(first==NULL)
	{
		first=new_node;
		new_node->next=NULL;
		new_node->prev=NULL;
		last=first;
	}
	else
	{
		new_node->next=first;
		new_node->prev=NULL;
		first=new_node;
	}
	printf("\n Node Added Successfully.");
	return first;
}


struct Stud * InsertAnywhereBefore(struct Stud *first,struct Stud **last )
{
	int val;
	struct Stud *new_node=NULL;
	struct Stud *temp=NULL;
	struct Stud *p;
```

```
	new_node=(struct Stud *)malloc(sizeof(struct Stud));

	printf("\n Enter data:");
	scanf("%d",&new_node->data);
	new_node->next=NULL;
	new_node->prev=NULL;

	printf("\n Enter the value befor you want to add:");
	scanf("%d",&val);

	p=first;

	while(p!=NULL && p->data!=val)
	{
		printf("%d ",p->data);
		p=p->next;

	}
		printf("%d ",p->data);


		if(p->prev==NULL)	//at the first
		{
			new_node->next=p;
			new_node->prev=NULL;
			p->prev=new_node;
			first=new_node;
		}
		else
		{
			new_node->next=p;
			new_node->prev=p->prev;
			p->prev->next=new_node;
			new_node->next->prev=new_node;
		}

	return first;
}

struct Stud * InsertAnywhereAfter(struct Stud *first,struct Stud **last )
{
	int val;
	struct Stud *new_node=NULL;
	struct Stud *temp=NULL;
	struct Stud *p;

	new_node=(struct Stud *)malloc(sizeof(struct Stud));
```

```
	printf("\n Enter data:");
	scanf("%d",&new_node->data);
	new_node->next=NULL;
	new_node->prev=NULL;

	printf("\n Enter the value befor you want to add:");
	scanf("%d",&val);

	p=first;

	while(p!=NULL && p->data!=val)
	{
		printf("%d ",p->data);
		p=p->next;

	}
		printf("%d ",p->data);


		if(p->next==NULL)	//at the first
		{

			new_node->prev=p;
			new_node->next=NULL;
			p->next=new_node;
			(*last)=new_node;
		}
		else
		{
			new_node->next=p->next;
			new_node->prev=p;
			p->next=new_node;
			new_node->next->prev=new_node;

		}

	return first;
}

struct Stud * DeleteBeg(struct Stud *first,struct Stud **last )
{
	struct Stud *temp;

	if(first==NULL)
	{
		printf("\n List is Empty.");
	}
```

```
	else
	{
		if(first==(*last))	//if only one node is exist
		{
			temp=first;
			free(temp);
			first=NULL;
			(*last)=NULL;
			printf("\n Deleted Successfully.");
		}
		else		//delete node from the begining
		{
			temp=first;
			first=first->next;
			first->prev=NULL;
			free(temp);
			printf("\n Deleted Successfully.");
		}
	}
	return first;
}

struct Stud * DeleteEnd(struct Stud *first,struct Stud **last )
{
	struct Stud *temp,*p;

	if(first==NULL)
	{
		printf("\n List is Empty.");
	}
	else
	{
		if(first==(*last))	//if only one node is exist
		{
			temp=first;
			free(temp);
			first=NULL;
			(*last)=NULL;
			printf("\n Deleted Successfully.");
		}
		else
		{
			temp=(*last);
			(*last)=(*last)->prev;
			free(temp);
			(*last)->next=NULL;
			printf("\n Deleted Successfully.");
		}
```

```
	}
	return first;
}


void Display(struct Stud *first)
{
	struct Stud *p;
	p=first;

	while(p!=NULL)
	{
		printf("%d ",p->data);
		p=p->next;
	}
}
void main()
{
	struct Stud *first=NULL;
	struct Stud *last=NULL;

	int ch,count;
	char choice='n';

	printf("\n---------------");
	printf("\n 1.Create List");
	printf("\n 2.Insert End");
	printf("\n 3.Insert Begning ");
	printf("\n 4.Insert Anywhere Before");
	printf("\n 5.Insert Anywhere After");
	printf("\n 6.Delete Begning");
	printf("\n 7.Delete End");
	printf("\n 8.Display");
	printf("\n---------------");

	do
	{
		printf("\n Enter Your choice:");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1:
				first=CreateList(first,&last);
				break;
			case 2:
				first=InsertEnd(first,&last);
				break;
			case 3:
```

```
                                first=InsertBeg(first,last);
                                break;
                        case 4:
                                first=InsertAnywhereBefore(first,&last);
                                break;
                        case 5:
                                first=InsertAnywhereAfter(first,&last);
                                break;
                        case 6:
                                first=DeleteBeg(first,&last);
                                break;
                        case 7:
                                first=DeleteEnd(first,&last);
                                break;
                        case 8:
                                Display(first);
                                break;
                }
                printf("\n Do You want to continue:");
                scanf(" %c",&choice);


        }while(choice=='y' || choice=='Y');


        getch();
}
```

```
========
OUTPUT
========

---------------
 1.Create List
 2.Insert End
 3.Insert Begning
 4.Insert Anywhere Before
 5.Insert Anywhere After
 6.Delete Begning
 7.Delete End
 8.Display
---------------
 Enter Your choice:1

 Enter number of nodes you want to create:2

 Enter data:10
```

```
 Enter data:20

 List Created Successfully.
 Do You want to continue:y

 Enter Your choice:2

 Enter data:30

 Node Added Successfully.
 Do You want to continue:y

 Enter Your choice:4

 Enter data:3

 Enter the value befor you want to add:10
10
 Do You want to continue:y

 Enter Your choice:8
3 10 20 30
 Do You want to continue:y

 Enter Your choice:6

 Deleted Successfully.
 Do You want to continue:y

 Enter Your choice:8
10 20 30
 Do You want to continue:y

 Enter Your choice:7

 Deleted Successfully.
 Do You want to continue:y

 Enter Your choice:8
10 20
 Do You want to continue:n


========================
2. Doubly Circuler Link List.
========================
```

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

struct Stud
{
	int data;
	struct Stud *next,*prev;
};

struct Stud * CreateList(struct Stud *first,struct Stud **last )
{
	int n;
	printf("\n Enter number of nodes you want to create:");
	scanf("%d",&n);

	while(n!=0)
	{

		struct Stud *new_node=NULL;
		struct Stud *temp=NULL;

		new_node=(struct Stud *)malloc(sizeof(struct Stud));

		printf("\n Enter data:");
		scanf("%d",&new_node->data);
		new_node->next=NULL;
		new_node->prev=NULL;

		if(first==NULL)
		{
			first=new_node;
			new_node->prev=NULL;
			new_node->next=NULL;
			*last=first;
		}
		else
		{
			temp=(*last);
			(*last)->next=new_node;
			new_node->prev=temp;
			(*last)=new_node;
			(*last)->next=first;
			first->prev=(*last);
		}
		n--;
	}
	printf("\n List Created Successfully.");
```

```
	return first;
}

struct Stud * InsertEnd(struct Stud *first,struct Stud **last )
{
	struct Stud *new_node=NULL;
	struct Stud *temp=NULL;

	new_node=(struct Stud *)malloc(sizeof(struct Stud));

	printf("\n Enter data:");
	scanf("%d",&new_node->data);
	new_node->next=NULL;
	new_node->prev=NULL;

	if(first==NULL)
		{
			first=new_node;
			new_node->prev=NULL;
			new_node->next=NULL;
			*last=first;
		}
		else
		{
			temp=(*last);
			(*last)->next=new_node;
			new_node->prev=temp;
			(*last)=new_node;
			(*last)->next=first;
			first->prev=(*last);
		}
	return first;
}

struct Stud * InsertBeg(struct Stud *first,struct Stud *last )
{
	struct Stud *new_node=NULL;
	struct Stud *temp=NULL;

	new_node=(struct Stud *)malloc(sizeof(struct Stud));

	printf("\n Enter data:");
	scanf("%d",&new_node->data);

	new_node->next=NULL;
	new_node->prev=NULL;

	if(first==NULL)
```

```
	{
		first=new_node;
		new_node->next=NULL;
		new_node->prev=NULL;
		last=first;
	}
	else
	{
		new_node->next=first;
		new_node->next->prev=first;  //new_node->next->prev=first;
change mam says
		first=new_node;
		first->prev=last;
		last->next=first;
	}
	printf("\n Node Added Successfully.");
	return first;
}

struct Stud * InsertAnywhereBefore(struct Stud *first,struct Stud **last )
{
	int val;
	struct Stud *new_node=NULL;
	struct Stud *temp=NULL;
	struct Stud *p;

	new_node=(struct Stud *)malloc(sizeof(struct Stud));

	printf("\n Enter data:");
	scanf("%d",&new_node->data);
	new_node->next=NULL;
	new_node->prev=NULL;

	printf("\n Enter the value befor you want to add:");
	scanf("%d",&val);

	p=first;

	while(p->next!=first && p->data!=val)
	{
		printf("%d ",p->data);
		p=p->next;

	}
		printf(" %d ",p->data);


		if(p->prev==(*last))	//at the first
```

```
		{

			new_node->next=p;
			new_node->prev=(*last);
			p->prev=new_node;
			first=new_node;
			(*last)->next=first;

		}
		else
		{
			new_node->next=p;
			new_node->prev=p->prev;
			p->prev->next=new_node;
			new_node->next->prev=new_node;
		}

	return first;
}

struct Stud * InsertAnywhereAfter(struct Stud *first,struct Stud **last )
{
	int val;
	struct Stud *new_node=NULL;
	struct Stud *temp=NULL;
	struct Stud *p;

	new_node=(struct Stud *)malloc(sizeof(struct Stud));

	printf("\n Enter data:");
	scanf("%d",&new_node->data);
	new_node->next=NULL;
	new_node->prev=NULL;

	printf("\n Enter the value After you want to add:");
	scanf("%d",&val);

	p=first;

	while(p->next!=first && p->data!=val)
	{
		printf("%d ",p->data);
		p=p->next;

	}
		printf("%d ",p->data);
```

```
            if(p->next==first)  //at the last
            {
                  new_node->prev=p;
                  new_node->next=first;
                  p->next=new_node;
                  (*last)=new_node;
                  first->prev=(*last);
            }
            else
            {
                  new_node->next=p->next;
                  new_node->prev=p;
                  p->next=new_node;
                  new_node->next->prev=new_node;
            }

      return first;
}


struct Stud * DeleteBeg(struct Stud *first,struct Stud **last )
{
      struct Stud *temp;

      if(first==NULL)
      {
            printf("\n List is Empty.");
      }
      else
      {
            if(first==(*last))    //if only one node is exist
            {
                  temp=first;
                  free(temp);
                  first=NULL;
                  (*last)=NULL;
                  printf("\n Deleted Successfully.");
            }
            else           //delete node from the begining
            {
                  temp=first;
                  first=first->next;
                  first->prev=(*last);
                  (*last)->next=first;
                  free(temp);
                  printf("\n Deleted Successfully.");
            }
      }
```

```
	return first;
}


struct Stud * DeleteEnd(struct Stud *first,struct Stud **last )
{
	struct Stud *temp,*p;

	if(first==NULL)
	{
		printf("\n List is Empty.");
	}
	else
	{
		if(first==(*last))    //if only one node is exist
		{
			temp=first;
			free(temp);
			first=NULL;
			(*last)=NULL;
			printf("\n Deleted Successfully.");
		}
		else
		{
			temp=(*last);
			(*last)=(*last)->prev;
			(*last)->next=first;
			first->prev=(*last);
			free(temp);
			printf("\n Deleted Successfully.");
		}
	}
	return first;
}


void Display(struct Stud *first)
{
	struct Stud *p;
	p=first;

	if(p==NULL)
	{
		printf("\n List is empty.");
	}
	else
	{
		while(p->next!=first)
```

```
		{
			printf("%d ",p->data);
			p=p->next;
		}
		printf("%d ",p->data);
	}
}
void main()
{
	struct Stud *first=NULL;
	struct Stud *last=NULL;

	int ch,count;
	char choice='n';

	printf("\n---------------");
	printf("\n 1.Create List");
	printf("\n 2.Insert End");
	printf("\n 3.Insert Begning ");
	printf("\n 4.Insert Anywhere Before");
	printf("\n 5.Insert Anywhere After");
	printf("\n 6.Delete Begning");
	printf("\n 7.Delete End");
	printf("\n 8.Display");
	printf("\n---------------");

	do
	{
		printf("\n Enter Your choice:");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1:
				first=CreateList(first,&last);
				break;
			case 2:
				first=InsertEnd(first,&last);
				break;
			case 3:
				first=InsertBeg(first,last);
				break;
			case 4:
				first=InsertAnywhereBefore(first,&last);
				break;
			case 5:
				first=InsertAnywhereAfter(first,&last);
				break;
			case 6:
```

```
                                first=DeleteBeg(first,&last);
                                break;
                        case 7:
                                first=DeleteEnd(first,&last);
                                break;
                        case 8:
                                Display(first);
                                break;
                }
                printf("\n Do You want to continue:");
                scanf(" %c",&choice);


        }while(choice=='y' || choice=='Y');


        getch();
}
```

```
===========
OUTPUT
===========
---------------
 1.Create List
 2.Insert End
 3.Insert Begning
 4.Insert Anywhere Before
 5.Insert Anywhere After
 6.Delete Begning
 7.Delete End
 8.Display
---------------
 Enter Your choice:1

 Enter number of nodes you want to create:2

 Enter data:10

 Enter data:20

 List Created Successfully.
 Do You want to continue:y

 Enter Your choice:2

 Enter data:45
```

```
 Do You want to continue:y

 Enter Your choice:3

 Enter data:7

 Node Added Successfully.
 Do You want to continue:y

 Enter Your choice:8
7 10 20 45
 Do You want to continue:y

 Enter Your choice:4

 Enter data:88

 Enter the value befor you want to add:7

 Do You want to continue:y

 Enter Your choice:5

 Enter data:96

 Enter the value After you want to add:20
88 7 10 20
 Do You want to continue:y

 Enter Your choice:8
88 7 10 20 96 45
 Do You want to continue:y

 Enter Your choice:6

 Deleted Successfully.
 Do You want to continue:y

 Enter Your choice:8
7 10 20 96 45
 Do You want to continue:y

 Enter Your choice:7

 Deleted Successfully.
 Do You want to continue:n
```

==========
3. Bouble Sort
==========

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

void GetData(int num[],int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		printf("\n Enter Value:");
		scanf("%d",&num[i]);
	}

}
void Display(int num[],int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		printf("%d ",num[i]);
	}

}
void Sort(int num[],int n)
{
	int i,j,temp;
	for(i=0;i<n-1;i++)
	{
		for(j=0;j<n-i-1;j++)
		{
			if(num[j]>num[j+1])
			{
				temp=num[j];
				num[j]=num[j+1];
				num[j+1]=temp;
			}
		}
	}

}
void main()
{
```

```
        int num[20],i,n;

        printf("\n Enter value of n:");
        scanf("%d",&n);

        GetData(num,n);
        printf("\n\n Before Sorted..\n");
        Display(num,n);

        Sort(num,n);
        printf("\n\n After Sorted..\n");
        Display(num,n);
        getch();
}
```

```
=========
OUTPUT
=========
Enter value of n:5

 Enter Value:12

 Enter Value:7

 Enter Value:66

 Enter Value:4

 Enter Value:89


 Before Sorted..
12 7 66 4 89

 After Sorted..
4 7 12 66 89
```

```
==================
4. Selection Sort
==================

#include<stdio.h>
#include<conio.h>
#include<malloc.h>

void GetData(int num[],int n)
```

```
{
	int i;
	for(i=0;i<n;i++)
	{
		printf("\n Enter Value:");
		scanf("%d",&num[i]);
	}

}
void Display(int num[],int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		printf("%d ",num[i]);
	}

}

void Selection_Sort(int num[],int n)
{
	int min,temp,index;


	for(int i=0;i<n-1;i++)
	{
		min=i;
		for(int j=i+1;j<n;j++)
		{
			if(num[j]<num[min])
			{
				min=j;
			}
		}
		temp=num[i];
		num[i]=num[min];
		num[min]=temp;
	}
}
void main()
{
	int num[20],i,n;

	printf("\n Enter value of n:");
	scanf("%d",&n);

	GetData(num,n);
	printf("\n\n Before Sorted..\n");
```

```
	Display(num,n);

	Selection_Sort(num,n);
	printf("\n\n After Sorted..\n");
	Display(num,n);

	getch();
}
```

==========
OUTPUT
==========

```
 Enter value of n:5

 Enter Value:45

 Enter Value:0

 Enter Value:42

 Enter Value:12

 Enter Value:2


 Before Sorted..
45 0 42 12 2

 After Sorted..
0 2 12 42 45
```

==========
5. Insertion Sort
==========

```
#include<stdio.h>
#include<conio.h>


void GetData(int num[],int n)
{
	int i;
```

```
	for(i=0;i<n;i++)
	{
		printf("\n Enter Value:");
		scanf("%d",&num[i]);
	}

}
void Display(int num[],int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		printf("%d ",num[i]);
	}

}

void Insertion_Sort(int num[],int n)
{
	int i,j,k,temp;


	for(i=0;i<n;i++)
	{
		for(j=i+1;j<n;j++)
		{
			if(num[j]<num[i])
			{
				temp = num[j];
				for(k=j;k>0;k--)
				{
					num[k] = num[k-1];
				}
				num[i]=temp;
			}

		}
	}
}
void main()
{
	int num[20],i,n;

	printf("\n Enter value of n:");
	scanf("%d",&n);

	GetData(num,n);
	printf("\n\n Before Sorted..\n");
```

```
        Display(num,n);

        Insertion_Sort(num,n);
        printf("\n\n After Sorted..\n");
        Display(num,n);

        getch();
}
```

```
=========
OUTPUT
=========

 Enter value of n:5

 Enter Value:45

 Enter Value:0

 Enter Value:42

 Enter Value:12

 Enter Value:2


 Before Sorted..
45 0 42 12 2

 After Sorted..
0 2 12 42 45
```

```
=============
6. Linear Search
=============

#include<stdio.h>
#include<conio.h>
#include<malloc.h>

void GetData(int num[],int n)
{
        int i;
```

```
	for(i=0;i<n;i++)
	{
		printf("\n Enter Value:");
		scanf("%d",&num[i]);
	}

}
void Display(int num[],int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		printf("%d ",num[i]);
	}

}


void Liner_Search(int num[],int n)
{
	int i,no,found=0;
	printf("\n Enter Value you want to search:");
	scanf("%d",&no);

	for(i=0;i<n;i++)
	{
		if(no==num[i])
		{
			printf("%d is on %d position.",no,i);
			found=1;
			break;
		}
	}
	if(!found)
		printf("\n Element is not present");

}
void main()
{
	int num[20],i,n;

	printf("\n Enter value of n:");
	scanf("%d",&n);

	GetData(num,n);
	printf("\n\n Before Sorted..\n");
	Display(num,n);
```

```
	Liner_Search(num,n);
	getch();
}
```

==========
OUTPUT
==========

```
 Enter value of n:5

 Enter Value:12

 Enter Value:7

 Enter Value:66

 Enter Value:4

 Enter Value:89

12 7 66 4 89


 Enter Value you want to search:66
66 is on 3 position.
```

==========
7. Binary Search
==========

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

void GetData(int num[],int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		printf("\n Enter Value:");
		scanf("%d",&num[i]);
	}

}
void Display(int num[],int n)
{
	int i;
```

```
	for(i=0;i<n;i++)
	{
		printf("%d ",num[i]);
	}

}


void Binary_Search(int num[],int n)
{
	int key,found=0,mid,first,last,index;

	printf("\n Enter Value you want to search:");
	scanf("%d",&key);

	first=0;
	last=n-1;

	while(first<last)
	{
		mid=(first+last)/2;

		if(key == num[mid])
		{
			found=1;
			printf("%d is found at %d position",key,mid);
			index=mid;
			break;
		}
		else if(key<mid)
		{
			last=mid-1;
		}
		else if(key>mid)
		{
			first=mid+1;
		}
	}
	if(!found)
		printf("\n Element is not present");

}
void main()
{
	int num[20],i,n;

	printf("\n Enter value of n:");
	scanf("%d",&n);
```

```
        GetData(num,n);
        printf("\n\n Before Sorted..\n");
        Display(num,n);

        Binary_Search(num,n);
        Binary_Search(num,n);
        Binary_Search(num,n);
        Binary_Search(num,n);
        getch();
}
```

```
========
OUTPUT
========
 Enter value of n:10

 Enter Value:1

 Enter Value:2

 Enter Value:4

 Enter Value:5

 Enter Value:6

 Enter Value:7

 Enter Value:8

 Enter Value:9

 Enter Value:10

 Enter Value:12


 Before Sorted..
1 2 4 5 6 7 8 9 10 12
 Enter Value you want to search:3

 Element is not present
 Enter Value you want to search:2
2 is found at 1 position
 Enter Value you want to search:11
```

```
 Element is not present
 Enter Value you want to search:45

 Element is not present
```

```
================
8 .Expression Tree
================
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int top = -1;

struct Tree
{
	int data;
	struct Tree *next,*prev;
};

void Push(struct Tree *stack[],int size,struct Tree *key)
{
	if(top>=(size-1))
	{
		printf("\n Overflow..");
	}
	else
	{
		top=top+1;
		stack[top]=(key);

	}
}

struct Tree * Pop(struct Tree *stack[],int size)
{
	struct Tree *data;

	if(top==-1)
	{
		printf("\n Underflow...");
		return NULL;
	}
	else
	{
```

```
		data=stack[top];
		stack[top]=NULL;
		top=top-1;
	}

	return data;
}

bool isOperand(char ch)
{
	if(ch>=65 && ch<=90 || ch>=97 && ch<=122)
	{
		return 1;
	}
	else
		return 0;
}
bool isOperator(char ch)
{
	if(ch=='+' || ch=='-' || ch=='/' || ch=='*')
		return 1;
	else
		return 0;
}
struct Tree * CreateTree(struct Tree *root,char key)
{

	struct Tree *newnode=NULL;

	newnode=(struct Tree *)malloc(sizeof(struct Tree));
	newnode->data = key;
	newnode->next=NULL;
	newnode->prev=NULL;

	return newnode;
}



void InOrder(struct Tree *root)
{
	if(root!=NULL)
	{
		InOrder(root->prev);
		printf("%c ",root->data);
		InOrder(root->next);
	}
}
```

```
struct Tree * ExpressionTree(char str[],int size_str,struct Tree * root)
{
	struct Tree *stack[20];

	struct Tree *p,*first,*second;

	for(int i=0;str[i]!='\0';i++)
	{
		if(isOperand(str[i]))
		{
			p = CreateTree(root,str[i]);
			Push(stack,20,p);
		}
		else
		{
			p = CreateTree(root,str[i]);

			first = Pop(stack,20);
			second = Pop(stack,20);

			p->next=first;
			p->prev=second;

			Push(stack,20,p);

		}
	}
	first = Pop(stack,20);

	return first;

}
void main()
{
	struct Tree *first=NULL;
	struct Tree *last=NULL;
	struct Tree *res=NULL;

	int key,ch,count,n;
	char choice='n';

	char str[20],prv_char;


	int i,cnt=0,size,invalid=0;
	printf("\n Enter String:");
```

```
	scanf("%s",str);

	for(int i=0;str[i]!='\0';i++)
	{
		cnt++;
	}
	size=cnt;

	first = ExpressionTree(str,size,first);

	InOrder(first);

	getch();
}
```

```
=========
OUTPUT
=========

 Enter String:AB/C-DE*+
A / B - C + D * E
```

```
=========
OUTPUT
=========

 Enter String:XY+Z+AB+C-*
X + Y + Z * A + B - C
```

```
==============================
9. Binary Search Tree ( Recursive)
==============================
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int top = -1;

struct Tree
{
	int data;
	struct Tree *next,*prev;
```

```
};

struct Tree * SearchTree(struct Tree *root,int key)
{

		if(root==NULL)
			return NULL;
		if(key>root->data)
			return SearchTree(root->next,key);

		if(key < root->data)
			return SearchTree(root->prev,key);
		else
			return root;
}

struct Tree * CreateTree(struct Tree *root,int key)
{

	struct Tree *newnode=NULL;

	newnode=(struct Tree *)malloc(sizeof(struct Tree));
	newnode->data = key;
	newnode->next=NULL;
	newnode->prev=NULL;

	return newnode;
}


struct Tree *Insert(struct Tree *root, int key)
{

	 if(root == NULL)
	 {
		return CreateTree(root,key);
	 }

	 if (key < root->data)
	 {
		root->prev = Insert(root->prev, key);
	 }
	 else
	 {
		root->next = Insert(root->next, key);
	 }
  return root;
}
```

```
void InOrder(struct Tree *root)
{
	if(root!=NULL)
	{
		InOrder(root->prev);
		printf("%d ",root->data);
		InOrder(root->next);
	}
}
void PreOrder(struct Tree *root)
{
	if(root!=NULL)
	{
		printf("%d ",root->data);
		PreOrder(root->prev);
		PreOrder(root->next);
	}
}
void PostOrder(struct Tree *root)
{
	if(root!=NULL)
	{
		PostOrder(root->prev);
		PostOrder(root->next);
		printf("%d ",root->data);
	}
}

void main()
{
	struct Tree *first=NULL;
	struct Tree *last=NULL;
	struct Tree *res=NULL;

	int key,ch,count,n;
	char choice='n';

	printf("\n---------------");
	printf("\n 1.Create Tree");
	printf("\n---------------");

	printf("\n Enter the number of nodes:");
	scanf("%d",&n);
	for(int i=0;i<n;i++)
	{
		printf("\n Enter Key:");
```

```
        scanf("%d",&key);


        first = Insert(first,key);
}


printf("\n\n---------------");
printf("\n 1.Insert Node");
printf("\n 2.InOder");
printf("\n 3.PreOder");
printf("\n 4.PostOder");
printf("\n 5.Search");
printf("\n 7.Exit");
printf("\n---------------");

do
{

        printf("\n\n Enter Your choice:");
        scanf("%d",&ch);
        while(ch<1 || ch>7)
        {
                printf("\n Invalid Choice");
                printf("\n Enter Your choice Again:");
                scanf("%d",&ch);

        }

        switch(ch)
        {
                case 1:
                                printf("\n Enter Key:");
                                scanf("%d",&key);

                                first = Insert(first,key);
                                break;
                case 2:
                                printf("\n");
                                InOrder(first);
                                break;
                case 3:
                                printf("\n");
                                PreOrder(first);
                                break;
                case 4:
                                printf("\n");
                                PostOrder(first);
```

```
                        break;
                case 5:
                        printf("\n Enter Key you want to search:");
                        scanf("%d",&key);

                        res=SearchTree(first,key);
                        if(res==NULL)
                        {
                                printf("\n Node is not found");
                        }
                        else
                        {
                                printf("%d node is found.",res->data);
                        }
                        break;
                case 7:
                        exit(0);
                        break;

                }
        }while(ch!=7);


        getch();
}
```

```
==========
OUTPUT
==========

---------------
 1.Create Tree
---------------
 Enter the number of nodes:11

 Enter Key:36

 Enter Key:20

 Enter Key:15

 Enter Key:17

 Enter Key:29

 Enter Key:25

 Enter Key:31
```

```
 Enter Key:49

 Enter Key:83

 Enter Key:65

 Enter Key:95


---------------
 1.Insert Node
 2.InOder
 3.PreOder
 4.PostOder
 5.Search
 7.Exit
---------------

 Enter Your choice:1

 Enter Key:91


 Enter Your choice:2

15 17 20 25 29 31 36 49 65 83 91 95

 Enter Your choice:3

36 20 15 17 29 25 31 49 83 65 95 91

 Enter Your choice:4

17 15 25 31 29 20 65 91 95 83 49 36

 Enter Your choice:5

 Enter Key you want to search:65
65 node is found.

 Enter Your choice:7
```

```
====================================
10. Binary Search Tree (Non-Recursive)
====================================

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int top = -1;

struct Tree
{
	int data;
	struct Tree *next,*prev;
};

void Push(struct Tree *stack[],int size,struct Tree *key)
{
	if(top>=(size-1))
	{
		printf("\n Overflow..");
	}
	else
	{
		top=top+1;
		stack[top]=(key);

	}
}

struct Tree * Pop(struct Tree *stack[],int size)
{
	struct Tree *data;

	if(top==-1)
	{
		printf("\n Underflow...");
		return NULL;
	}
	else
	{
		data=stack[top];
		stack[top]=NULL;
		top=top-1;
	}

	return data;
}
```

```
struct Tree * Top(struct Tree * stack[])
{
	struct Tree *data;
	data=stack[top];

	if(top!=-1)
		return data;
	else
		return NULL;
}
struct Tree * SearchTree(struct Tree *root, int key)
{

	struct Tree *p;


	p=root;

	while(p!=NULL)
	{
		if(key>p->data)
			p=p->next;
		else if(key < p->data)
			p=p->prev;
		if(key == p->data)
		{
			printf("key is found");
			return p;
			break;
		}
	}

	return NULL;
}

struct Tree * CreateTree(struct Tree *root)
{
	int n;
	printf("\n Enter number of nodes you want to create:");
	scanf("%d",&n);

	while(n!=0)
	{
		struct Tree *newnode=NULL;
		newnode=(struct Tree *)malloc(sizeof(struct Tree));

		printf("\n Enter data:");
```

```
        scanf("%d",&newnode->data);
        newnode->next=NULL;
        newnode->prev=NULL;

        struct Tree *p = NULL;


        if(root==NULL)
        {
                root=newnode;
        }
        else
        {
                p= root;
                while (1)
                {
                        if(newnode->data < p->data)
                        {
                                if(p->prev==NULL)
                                {
                                        p->prev=newnode;
                                        break;
                                }
                                else
                                {
                                        p=p->prev;
                                }
                        }
                        else
                        {
                                if(newnode->data > p->data)
                                {
                                        if(p->next==NULL)
                                        {
                                                p->next=newnode;
                                                break;
                                        }
                                        else
                                        {
                                                p=p->next;
                                        }
                                }
                        }
                }
        }
        n--;
}
return root;
```

```
}
struct Tree * InsertTree(struct Tree *root)
{

		struct Tree *newnode=NULL;
		newnode=(struct Tree *)malloc(sizeof(struct Tree));

		printf("\n Enter data:");
		scanf("%d",&newnode->data);
		newnode->next=NULL;
		newnode->prev=NULL;

		struct Tree *p = NULL;


		if(root==NULL)
		{
			root=newnode;
		}
		else
		{
			p= root;
			while (1)
			{
				if(newnode->data < p->data)
				{
					if(p->prev==NULL)
					{
						p->prev=newnode;
						break;
					}
					else
					{
						p=p->prev;
					}
				}
				else
				{
					if(newnode->data > p->data)
					{
						if(p->next==NULL)
						{
							p->next=newnode;
							break;
						}
						else
						{
							p=p->next;
```

```
                                        }
                                    }
                                }
                            }
                    }

        return root;
}

void inOrder(struct Tree *root)
{
        struct Tree *p;
        p=root;

        struct Tree *stack[20];

        while( top!=-1 || p!=NULL )
        {
                if(p!=NULL)
                {
                        Push(stack,10,p);
                        p=p->prev;
                }
                else
                {
                        p=Pop(stack,20);
                        printf("%d ",p->data);
                        p=p->next;
                }
        }
}
void PreOrder(struct Tree *root)
{
        struct Tree *p;
        p=root;

        struct Tree *stack[20];

        while( top!=-1 || p!=NULL )
        {
                if(p!=NULL)
                {

                        Push(stack,20,p);
                        printf("%d ",p->data);
                        p=p->prev;
                }
                else
```

```
		{
			p=Pop(stack,20);
			p=p->next;
		}
	}
}
void PostOrder(struct Tree *root)
{
	struct Tree *p;
	struct Tree *res;
	p=root;

	struct Tree *stack[20];

	while( top!=-1 || p!=NULL )
	{

		while(p!=NULL)
		{
			Push(stack,10,p);
			p=p->prev;
		}
		while(top!=-1 && (p=Top(stack))->data < 0)
		{
					p=Pop(stack,20);
					printf("%d ",(p->data*-1));
					p->data = (p->data)*-1;   ///for maing value
again positive extra
		}
		if(top!=-1)
		{
			p=Top(stack);
			p->data = (p->data)*-1;
			p=p->next;
		}
		else
		{
			break;
		}
	}
}

int MinValue(struct Tree *root)
{
	struct Tree *p;
	p=root;

	while(p->prev!=NULL)
```

```
	{
		p=p->prev;
	}
	return p->data;
}

int MaxValue(struct Tree *root)
{
	struct Tree *p;
	p=root;

	while(p->next!=NULL)
	{
		p=p->next;
	}
	return p->data;
}

void EmptyStack(struct Tree *stack[])
{
	struct Tree *p;
	while(top!=-1)
	{
		p=Pop(stack,20);
	}
}
void DeleteNode(struct Tree *root)
{
	int key,max,old_key=0;
	struct Tree *p,*add;

	printf("\n Enter Key:");
	scanf("%d",&key);



	struct Tree *stack[20];


	p=root;
	while( top!=-1 || p!=NULL )
	{

		while(p!=NULL)
		{
			Push(stack,10,p);
			if(key!=p->data)
```

```
                {

                        p=p->prev;
                }
                else
                {

                        /// if key is leaf node
                        if(p->prev==NULL && p->next==NULL)  //key is a
leaf node
                        {
                                p=Pop(stack,20);                        //go up
                                p=Pop(stack,20);
                                if(p->prev!=NULL &&  p->prev->data==key)
                //check key is left
                                {
                                        p->prev=NULL;
                                        printf("%d key is deleted",key);
                                        if(p->data < 0)
                                        {
                                                p->data = (p->data)*-1;
                                        }
                                        if(old_key!=0)
                                        {
                                                add=SearchTree(root,old_key);
                                                add->data=key;
                                        }
                                        break;
                                }
                                if(p->next!=NULL && p->next->data==key)
            //check key is right
                                {
                                        p->next=NULL;
                                        printf("%d key is deleted",key);
                                        if(p->data < 0)
                                        {
                                                p->data = (p->data)*-1;
                                        }
                                        if(old_key!=0)
                                        {
                                                add=SearchTree(root,old_key);
                                                add->data=key;
                                        }
                                        break;
                                }
                        }
                        else if(p->next==NULL || p->prev==NULL)
                        {
```

```
                        //if key has a single child
                        if(p->next==NULL )  //key has left single child
                        {
                                if(p->prev->prev==NULL && p->prev-
>next==NULL)
                                {
                                        p->prev->data=p->data;
                                        p->prev=NULL;
                                        printf("%d key is deleted",key);
                                        if(old_key!=0)
                                        {
                                                add=SearchTree(root,old_key);
                                                add->data=key;
                                        }
                                        EmptyStack(stack);
                                        break;
                                }
                        }
                        if(p->prev==NULL)    //key has  rigth single child
                        {
                                if(p->next->prev==NULL && p->next-
>next==NULL)
                                {
                                        p->data=p->next->data;
                                        p->next=NULL;
                                        printf("%d key is deleted",key);
                                        if(old_key!=0)
                                        {
                                                add=SearchTree(root,old_key);
                                                add->data=key;
                                        }
                                        EmptyStack(stack);
                                        break;
                                }
                        }

                        }
                        else
                        {
                                max=MaxValue(p->prev);
                        //      printf("\n max is %d \n",max);
                                old_key=key;
                                key=max;

                        }
```

```
                }
        }

        while(top!=-1 && (p=Top(stack))->data < 0)
        {
                        p=Pop(stack,20);
                        //      printf("%d ",(p->data*-1));
                        p->data = (p->data)*-1;   ///for maing value
again positive extra
        }

        if(top!=-1)
        {
                p=Top(stack);
                p->data = (p->data)*-1;
                p=p->next;
        }
        else
        {
                break;
        }

    }
}
void main()
{
    struct Tree *first=NULL;
    struct Tree *last=NULL;
    struct Tree *res=NULL;

    int ch,count,key;
    char choice='n';

    printf("\n---------------");
    printf("\n 1.Create Tree");
    printf("\n---------------");
    first=CreateTree(first);


    printf("\n\n---------------");
    printf("\n 1.Insert Node");
    printf("\n 2.InOder");
    printf("\n 3.PreOder");
    printf("\n 4.PostOder");
    printf("\n 5.Search");
    printf("\n 6.Min");
    printf("\n 7.Max");
    printf("\n 8.Delete");
```

```
        printf("\n 9.Exit");
        printf("\n---------------");

        do
        {

                printf("\n\n Enter Your choice:");
                scanf("%d",&ch);
                while(ch<1 || ch>9)
                {
                        printf("\n Invalid Choice");
                        printf("\n Enter Your choice Again:");
                        scanf("%d",&ch);

                }

                switch(ch)
                {
                        case 1:
                                        first=InsertTree(first);
                                        break;
                        case 2:
                                        printf("\n");
                                        inOrder(first);
                                        break;
                        case 3:
                                        printf("\n");
                                        PreOrder(first);
                                        break;
                        case 4:
                                        printf("\n");
                                        PostOrder(first);
                                        break;
                        case 5:
                                        printf("\n Enter key:");
                                        scanf("%d",&key);

                                        res=SearchTree(first,key);
                                        if(res==NULL)
                                        {
                                                printf("\n Node is not found");
                                        }
                                        break;
                        case 6:
                                        MinValue(first);
                                        printf("\n Minimum value is:
%d",MinValue(first));
                                        break;
```

```
                    case 7:
                                MaxValue(first);
                                printf("\n Minimum value is:
%d",MaxValue(first));
                                break;
                    case 8:
                                DeleteNode(first);
                                break;
                    case 9:
                                exit(0);
                                break;

                }
        }while(ch!=9);


        getch();
}
```

```
===========
OUTPUT
===========

---------------
 1.Create Tree
---------------
 Enter number of nodes you want to create:8

 Enter data:36

 Enter data:28

 Enter data:16

 Enter data:13

 Enter data:20

 Enter data:59

 Enter data:47

 Enter data:75


---------------
 1.Insert Node
```

```
 2.InOder
 3.PreOder
 4.PostOder
 5.Search
 6.Min
 7.Max
 8.Exit
---------------

 Enter Your choice:1

 Enter data:91


 Enter Your choice:2

13 16 20 28 36 47 59 75 91

 Enter Your choice:3

36 28 16 13 20 59 47 75 91

 Enter Your choice:4

13 20 16 28 47 91 75 59 36

 Enter Your choice:5

 Enter key:45

 Node is not found

 Enter Your choice:5

 Enter key:13
key is found

 Enter Your choice:6

 Minimum value is:13

 Enter Your choice:7

 Minimum value is:91

 Enter Your choice:8
```

```
==============
Output -> Delete
==============

---------------
 1.Create Tree
---------------
 Enter number of nodes you want to create:17

 Enter data:29

 Enter data:18

 Enter data:35

 Enter data:10

 Enter data:12

 Enter data:16

 Enter data:23

 Enter data:19

 Enter data:25

 Enter data:6

 Enter data:8

 Enter data:30

 Enter data:67

 Enter data:54

 Enter data:90

 Enter data:51

 Enter data:57


---------------
 1.Insert Node
 2.InOder
```

```
 3.PreOder
 4.PostOder
 5.Search
 6.Min
 7.Max
 8.Delete
 9.Exit
---------------

 Enter Your choice:2

6 8 10 12 16 18 19 23 25 29 30 35 51 54 57 67 90

 Enter Your choice:8

 Enter Key:67

57 key is deleted

 Enter Your choice:2

6 8 10 12 16 18 19 23 25 29 30 35 51 54 57 90

 Enter Your choice:8

 Enter Key:18

16 key is deleted

 Enter Your choice:2

6 8 10 12 16 19 23 25 29 30 35 51 54 57 90

 Enter Your choice:8

 Enter Key:16

12 key is deleted

 Enter Your choice:2

6 8 10 12 19 23 25 29 30 35 51 54 57 90

 Enter Your choice:8

 Enter Key:44
```

Enter Your choice:2

6 8 10 12 19 23 25 29 30 35 51 54 57 90

Enter Your choice:8

ASSIGNMENT -3

==========
1)UnWeighted Graph Shortest Path
==========

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

int front =-1;
int rear = -1;

struct node{
        int data;
        struct node * next;
};

struct table{
```

```
        int node_name_value;
        int known;
        int distance;
        int  path;
};
struct node * InsertEnd(struct node **first,struct node **last )
{
        struct node *new_node=NULL;
        new_node=(struct node *)malloc(sizeof(struct node));

        printf("\n Enter Adjacent:");
        scanf("%d",&new_node->data);
        new_node->next=NULL;

        if((*first)==NULL)
        {
                (*first)=new_node;
                (*last)=*first;
        }
        else
        {
                (*last)->next=new_node;
                (*last)=new_node;
        }

        return (*first);
}

void display(struct node n1[], int n)
{
        struct node *p;
        int i;
        for(int i=0;i<n;i++)
        {
                printf("\n %d --->",n1[i].data);

                p=n1[i].next;

                while(p!=NULL)
                {
                        printf("%d ",p->data);
                        p=p->next;
                }
        }

}

void display_table(struct table t1[], int total_vertex)
```

```
{
	printf("\n V\t Known\t Dv\t Pv");
	for(int i=0;i<total_vertex;i++)
	{
		printf("\n %d\t %d\t %d\t
%d",t1[i].node_name_value,t1[i].known,t1[i].distance,t1[i].path);
	}
}

void insert(int key,int size,int queue[])
{
	if(rear==(size-1))
	{
		printf("\n Overflow");
	}
	else
	{
		if(front==-1)
		{
			front=0;
		}
		rear +=1;
		queue[rear]=key;
	}
}

void del(int queue[])
{
	if(front==-1)
	{
		printf("\n underflow");

	}
	else if(front==rear)
	{
		//printf("\n Deleted item is:%d",queue[front]);
		front=-1;
		rear=-1;
	}
	else
	{
		//printf("\n Deleted item is:%d",queue[front]);
		front=front+1;
	}
}


void sortest_path(struct table t1[], int total_vertex,struct node n1[],int start)
```

```
{
	int reachable,V;
	int cur_dis=0;
	struct node *W;   //for recahble
	int i;
	int queue[10];
	int size=10;


	insert(start,size,queue);

	while(front!=-1)
	{
		V = queue[front];

		//	V =n1[i].data;	// data

				W = n1[V-1].next; //address of list


				t1[V-1].known = 1;
				del(queue);

				while(W!=NULL)
				{
					reachable = W->data;
					if(t1[reachable-1].distance == -1)
					{
						t1[reachable-1].distance = t1[V-
1].distance + 1;
						t1[reachable-1].path =V;
						insert(reachable,size,queue);
					}

					W=W->next;
				}


		printf("\n \n ");
		display_table(t1,total_vertex);
	}
}


void main()
{
	struct node n1[7];
	struct node *p;
```

```
int n,n2,total_vertex,start;

printf("\n Enter how many nodes you create n:");
scanf("%d",&total_vertex);              //total vertex = total V1,V2...V7

for(int i=0;i<total_vertex;i++)
{
        printf("\n Enter graph Node::");
        scanf("%d",&n1[i].data);

        n1[i].next =NULL;
        struct node *first =NULL;
        struct node *last =NULL;


        printf("\n Enter total number of adjacent of %d ::",n1[i].data);
        scanf("%d",&n2);


        for(int j=0;j<n2;j++)
        {
                first=InsertEnd(&first,&last);
                n1[i].next =first;
        }
}

display(n1,total_vertex);

struct table t1[7];

//INITIAL TABLE

printf("\n Enter starting node:");
scanf("%d",&start);



for(int i=0;i<total_vertex;i++)
{
        t1[i].node_name_value = i+1;
        t1[i].known =0;

        if(i==(start-1))
        {
                t1[i].distance = 0;
        }
        else
        {
```

```
                t1[i].distance = -1;
            }
            t1[i].path = 0;

        }


        display_table(t1,total_vertex);  //initial table
        //DISPLAY
        sortest_path(t1,total_vertex,n1,start);


        getch();
}
```

==========
OUTPUT
==========

```
 Enter how many nodes you create n:7

 Enter graph Node::1

 Enter total number of adjacent of 1 ::2

 Enter Adjacent:2

 Enter Adjacent:4

 Enter graph Node::2

 Enter total number of adjacent of 2 ::2

 Enter Adjacent:4

 Enter Adjacent:5

 Enter graph Node::3

 Enter total number of adjacent of 3 ::2

 Enter Adjacent:1

 Enter Adjacent:6

 Enter graph Node::4
```

```
Enter total number of adjacent of 4 ::3

Enter Adjacent:6

Enter Adjacent:7

Enter Adjacent:5

Enter graph Node::5

Enter total number of adjacent of 5 ::1

Enter Adjacent:7

Enter graph Node::6

Enter total number of adjacent of 6 ::0

Enter graph Node::7

Enter total number of adjacent of 7 ::1

Enter Adjacent:6

1 --->2 4
2 --->4 5
3 --->1 6
4 --->6 7 5
5 --->7
6 --->
7 --->6
V      Known   Dv      Pv
1      0       -1      0
2      0       -1      0
3      0       0       0
4      0       -1      0
5      0       -1      0
6      0       -1      0
7      0       -1      0
Enter starting node:3


V      Known   Dv      Pv
1      0       1       3
2      0       -1      0
3      1       0       0
4      0       -1      0
```

5	0	-1	0
6	0	1	3
7	0	-1	0

V	Known	Dv	Pv
1	1	1	3
2	0	2	1
3	1	0	0
4	0	2	1
5	0	-1	0
6	0	1	3
7	0	-1	0

V	Known	Dv	Pv
1	1	1	3
2	0	2	1
3	1	0	0
4	0	2	1
5	0	-1	0
6	1	1	3
7	0	-1	0

V	Known	Dv	Pv
1	1	1	3
2	1	2	1
3	1	0	0
4	0	2	1
5	0	3	2
6	1	1	3
7	0	-1	0

V	Known	Dv	Pv
1	1	1	3
2	1	2	1
3	1	0	0
4	1	2	1
5	0	3	2
6	1	1	3
7	0	3	4

V	Known	Dv	Pv
1	1	1	3
2	1	2	1

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

V	Known	Dv	Pv
1	1	1	3
2	1	2	1
3	1	0	0
4	1	2	1
5	1	3	2
6	1	1	3
7	1	3	4

==========
OUTPUT 2
==========

Enter how many nodes you create n:5

Enter graph Node::1

Enter total number of adjacent of 1 ::2

Enter Adjacent:2

Enter Adjacent:3

Enter graph Node::2

Enter total number of adjacent of 2 ::1

Enter Adjacent:4

Enter graph Node::4

Enter total number of adjacent of 4 ::1

Enter Adjacent:5

Enter graph Node::3

Enter total number of adjacent of 3 ::1

```
Enter Adjacent:2

Enter graph Node::5

Enter total number of adjacent of 5 ::1

Enter Adjacent:3

1 --->2 3
2 --->4
4 --->5
3 --->2
5 --->3
Enter starting node:1

V      Known   Dv     Pv
1      0       0      0
2      0       -1     0
3      0       -1     0
4      0       -1     0
5      0       -1     0


V      Known   Dv     Pv
1      1       0      0
2      0       1      1
3      0       1      1
4      0       -1     0
5      0       -1     0


V      Known   Dv     Pv
1      1       0      0
2      1       1      1
3      0       1      1
4      0       2      2
5      0       -1     0


V      Known   Dv     Pv
1      1       0      0
2      1       1      1
3      1       1      1
4      0       2      2
5      0       2      3


V      Known   Dv     Pv
```

1	1	0	0
2	1	1	1
3	1	1	1
4	1	2	2
5	0	2	3

V	Known	Dv	Pv
1	1	0	0
2	1	1	1
3	1	1	1
4	1	2	2
5	1	2	3

==========

2)Dijkastra's Weighted Graph Shortest Path

==========

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

int front =-1;
int rear = -1;

struct node{
        int data;
        int weight;
        struct node * next;
};

struct table{
        int node_name_value;
        int known;
        int distance;
        int  path;
};
struct node * InsertEnd(struct node **first,struct node **last )
{
        struct node *new_node=NULL;
        new_node=(struct node *)malloc(sizeof(struct node));

        printf("\n Enter Adjacent:");
        scanf("%d",&new_node->data);
```

```
        printf("\n Enter Weight:");
        scanf("%d",&new_node->weight);

        new_node->next=NULL;

        if((*first)==NULL)
        {
                (*first)=new_node;
                (*last)=*first;
        }
        else
        {
                (*last)->next=new_node;
                (*last)=new_node;
        }

        return (*first);
}

void display(struct node n1[], int n)
{
        struct node *p;
        int i;
        for(int i=0;i<n;i++)
        {
                printf("\n %d --->",n1[i].data);

                p=n1[i].next;

                while(p!=NULL)
                {
                        printf("%d ",p->data);
                        p=p->next;
                }
        }

}

void display_table(struct table t1[], int total_vertex)
{
        printf("\n V\t Known\t Dv\t Pv");
        for(int i=0;i<total_vertex;i++)
        {
                printf("\n %d\t %d\t %d\t
%d",t1[i].node_name_value,t1[i].known,t1[i].distance,t1[i].path);
        }
}
```

```
void insert(int key,int size,int queue[])
{
	if(rear==(size-1))
	{
		printf("\n Overflow");
	}
	else
	{
		if(front==-1)
		{
			front=0;
		}
		rear +=1;
		queue[rear]=key;
	}
}

void del(int queue[])
{
	if(front==-1)
	{
		printf("\n underflow");

	}
	else if(front==rear)
	{
		//printf("\n Deleted item is:%d",queue[front]);
		front=-1;
		rear=-1;
	}
	else
	{
		//printf("\n Deleted item is:%d",queue[front]);
		front=front+1;
	}
}


int MIN(struct table t1[], int total_vertex)
{
	int min,i,j,index=0,flag=0;

	for(i=0;i<total_vertex;i++)
	{
		if(t1[i].known==0 &&  t1[i].distance!=-1)
		{
			min=t1[i].distance;
```

```
                index = i;

                for(j=i+1;j<total_vertex;j++)
                {
                        if(t1[j].known==0 &&  t1[j].distance!=-1)
                        {
                                if(min > t1[j].distance)
                                {
                                        min = t1[j].distance;
                                        index = j;
                                        flag=1;
                                }
                        }
                }
                if(flag==1){i=j;}
        }
    }
    return index;
}



void sortest_path(struct table t1[], int total_vertex,struct node n1[])
{
    int reachable,V;
    int cur_dis=0;
    struct node *W;   //for recahble
    int i,start;
    int queue[10];
    int size=10,min,index;



    for(i=0;i<total_vertex;i++)
    {
        min = MIN(t1 , total_vertex);


        V = n1[min].data;
        W = n1[min].next;

                t1[min].known = 1;

                        while(W!=NULL)
                        {
                                reachable = W->data;

                                if(t1[reachable-1].distance == -1)
```

```
                                {
                                        t1[reachable-1].distance = t1[V-
1].distance + W->weight;
                                t1[reachable-1].path =V;
                                }
                                else
                                {
                                if(t1[V-1].distance + W->weight <=
t1[reachable-1].distance)
                                //if(t1[reachable-1].distance == -1)
                                {
                                        t1[reachable-1].distance = t1[V-
1].distance + W->weight;
                                        t1[reachable-1].path =V;
                                }

                                }
                                W=W->next;

                        }

                printf("\n \n ");
                display_table(t1,total_vertex);
        }
}


void main()
{
        struct node n1[7];
        struct node *p;
        int n,n2,total_vertex,start;

        printf("\n Enter how many nodes you create n:");
        scanf("%d",&total_vertex);              //total vertex = total V1,V2...V7

        for(int i=0;i<total_vertex;i++)
        {
                printf("\n Enter graph Node::");
                scanf("%d",&n1[i].data);

                n1[i].next =NULL;
                struct node *first =NULL;
                struct node *last =NULL;


                printf("\n Enter total number of adjacent of %d ::",n1[i].data);
                scanf("%d",&n2);
```

```
        for(int j=0;j<n2;j++)
        {
                first=InsertEnd(&first,&last);
                n1[i].next =first;
        }
}

display(n1,total_vertex);

printf("\n Enter starting node:");
scanf("%d",&start);



struct table t1[7];

//INITIAL TABLE


for(int i=0;i<total_vertex;i++)
{
        t1[i].node_name_value = i+1;
        t1[i].known =0;

        if(i==(start-1))
        {
                t1[i].distance = 0;
        }
        else
        {
                t1[i].distance = -1;
        }
        t1[i].path = 0;

}


display_table(t1,total_vertex);  //initial table
//DISPLAY
sortest_path(t1,total_vertex,n1);


getch();
}
```

OUTPUT

```
 Enter how many nodes you create n:5

 Enter graph Node::1

 Enter total number of adjacent of 1 ::2

 Enter Adjacent:2

 Enter Weight:6

 Enter Adjacent:3

 Enter Weight:1

 Enter graph Node::2

 Enter total number of adjacent of 2 ::1

 Enter Adjacent:4

 Enter Weight:2

 Enter graph Node::3

 Enter total number of adjacent of 3 ::1

 Enter Adjacent:2

 Enter Weight:2

 Enter graph Node::4

 Enter total number of adjacent of 4 ::2

 Enter Adjacent:3

 Enter Weight:1

 Enter Adjacent:5

 Enter Weight:2

 Enter graph Node::5
```

```
Enter total number of adjacent of 5 ::1

Enter Adjacent:3

Enter Weight:2

1 --->2 3
2 --->4
3 --->2
4 --->3 5
5 --->3
Enter starting node:1

V      Known   Dv     Pv
1      0       0      0
2      0       -1     0
3      0       -1     0
4      0       -1     0
5      0       -1     0


V      Known   Dv     Pv
1      1       0      0
2      0       6      1
3      0       1      1
4      0       -1     0
5      0       -1     0


V      Known   Dv     Pv
1      1       0      0
2      0       3      3
3      1       1      1
4      0       -1     0
5      0       -1     0


V      Known   Dv     Pv
1      1       0      0
2      1       3      3
3      1       1      1
4      0       5      2
5      0       -1     0


V      Known   Dv     Pv
1      1       0      0
2      1       3      3
```

```
3       1       1       1
4       1       5       2
5       0       7       4


V       Known   Dv      Pv
1       1       0       0
2       1       3       3
3       1       1       1
4       1       5       2
5       1       7       4



============
OUTPUT
============



Enter how many nodes you create n:6

Enter graph Node::1

Enter total number of adjacent of 1 ::3

Enter Adjacent:2

Enter Weight:10

Enter Adjacent:4

Enter Weight:2

Enter Adjacent:5

Enter Weight:7

Enter graph Node::2

Enter total number of adjacent of 2 ::0

Enter graph Node::3

Enter total number of adjacent of 3 ::2

Enter Adjacent:2
```

```
Enter Weight:3

Enter Adjacent:6

Enter Weight:4

Enter graph Node::4

Enter total number of adjacent of 4 ::2

Enter Adjacent:3

Enter Weight:2

Enter Adjacent:5

Enter Weight:3

Enter graph Node::5

Enter total number of adjacent of 5 ::1

Enter Adjacent:6

Enter Weight:2

Enter graph Node::6

Enter total number of adjacent of 6 ::0

1 --->2 4 5
2 --->
3 --->2 6
4 --->3 5
5 --->6
6 --->
Enter starting node:1

V       Known   Dv      Pv
1       0       0       0
2       0       -1      0
3       0       -1      0
4       0       -1      0
5       0       -1      0
6       0       -1      0


V       Known   Dv      Pv
```

1	1	0	0
2	0	10	1
3	0	-1	0
4	0	2	1
5	0	7	1
6	0	-1	0

V	Known	Dv	Pv
1	1	0	0
2	0	10	1
3	0	4	4
4	1	2	1
5	0	5	4
6	0	-1	0

V	Known	Dv	Pv
1	1	0	0
2	0	7	3
3	1	4	4
4	1	2	1
5	0	5	4
6	0	8	3

V	Known	Dv	Pv
1	1	0	0
2	0	7	3
3	1	4	4
4	1	2	1
5	1	5	4
6	0	7	5

V	Known	Dv	Pv
1	1	0	0
2	0	7	3
3	1	4	4
4	1	2	1
5	1	5	4
6	1	7	5

V	Known	Dv	Pv
1	1	0	0
2	1	7	3
3	1	4	4

```
4     1     2     1
5     1     5     4
6     1     7     5
```

==========
3)Priority Queue Using MIN Heap
==========

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int size = 15;
static int index=0;

void swap(int arr[],int index1,int index2)
{
	int temp;

	temp=arr[index1];
	arr[index1] = arr[index2];
	arr[index2] =temp;
}

void InsertPriorityQueue(int arr[],int value)		// Prioity queue insert / Min heap Insert
{
	int parent,child,i;
	index++;
	i=index;

	if(index >size+1)
	{
		printf("\n Heap is full");
	}
	else
	{
		arr[index] = value;

		while(i>1)
		{
			parent =i/2;
			child = arr[i];

			if(arr[parent] > child)
			{
```

```
                        swap(arr,i/2,i);
                        i=parent;
                }
                else
                {
                        break;
                }
        }

    }


}

void Display(int arr[],int n)
{

    if(n==0)
    {
        printf("Queue is Empty");
    }
    else
    {
        printf("\n");
        for(int i=1;i<=n;i++)
        {
            printf("%d ",arr[i]);
        }
    }
}
void DeletePriorityQueue(int arr[], int* N) //n is total members in queue
{
    //hear *N is used for make n value  n-1  after the function
    int i=1,left,right,last;
    int n=*N;


    if(n==0)
    {
        printf("\n queue is empty");
    }
    else
    {
        printf("\n Deleted item is : %d",arr[i]);

        while(i<n)
        {
            left = 2*i;
```

```
                right = 2*i + 1;
                last = n;
                if(left<n && right<n)
                {
                        if(arr[left]>arr[right] )
                        {
                                if(arr[right]<arr[last])
                                {
                                        arr[i]=arr[right];
                                        i=right;
                                }
                                else
                                {
                                        arr[i] = arr[last];
                                        i=last;
                                }
                        }
                        else if(arr[left]<arr[right] )
                        {
                                if( arr[left]<arr[last])
                                {
                                        arr[i] = arr[left];
                                        i=left;
                                }
                                else
                                {
                                        arr[i]=arr[last];
                                        i=last;
                                }
                        }
                }
                else
                {
                        break;
                }
        }
        if(i!=1)   // if i==1 then there is last node so we cant give any
value to it
        {
                arr[i] = arr[last];
        }
        n--;
        *N = n;
    }
}

void main()
{
```

```
        int n=0,i,ch,size,arr[20],value;
        char choice = 'n';

        printf("\n----------");
        printf("\n 1.Insert");
        printf("\n 2.Delete");
        printf("\n 3.Display");
        printf("\n 4.exit");
        printf("\n----------");

        do
        {
                printf("\n Enter your choice:");
                scanf("%d",&ch);

                switch(ch)
                {
                        case 1:
                                        n++;
                                        printf("\n Enter node :");
                                        scanf("%d",&value);
                                        InsertPriorityQueue(arr,value);

                                        break;
                        case 2:
                                        DeletePriorityQueue(arr,&n);
                                        break;
                        case 3:
                                        Display(arr,n);
                                        break;
                        case 4:
                                        exit(0);
                                        break;
                }
        }while(ch!=4);
        getch();
}

==============
OUTPUT
==============
----------
 1.Insert
 2.Delete
 3.Display
 4.exit
----------
 Enter your choice:1
```

```
Enter node :12

Enter your choice:1

Enter node :16

Enter your choice:1

Enter node :14

Enter your choice:1

Enter node :29

Enter your choice:1

Enter node :74

Enter your choice:1

Enter node :38

Enter your choice:1

Enter node :49

Enter your choice:38

Enter your choice:1

Enter node :39

Enter your choice:3

12 16 14 29 74 38 49 39
Enter your choice:2

Deleted item is : 12
Enter your choice:3

14 16 38 29 74 39 49
Enter your choice:2

Deleted item is : 14
Enter your choice:3

16 29 38 49 74 39
```

```
 Enter your choice:2

 Deleted item is : 16
 Enter your choice:3

29 39 38 49 74
 Enter your choice:2

 Deleted item is : 29
 Enter your choice:3

38 39 74 49
 Enter your choice:2

 Deleted item is : 38
 Enter your choice:3

39 49 74
 Enter your choice:2

 Deleted item is : 39
 Enter your choice:3

39 49
 Enter your choice:2

 Deleted item is : 39
 Enter your choice:3

39
 Enter your choice:2

 Deleted item is : 39
 Enter your choice:3
Queue is Empty
 Enter your choice:2

 queue is empty
 Enter your choice:4
```

==========

4)MAX Heap

==========

```
#include<stdio.h>
```

```
#include<conio.h>
#include<stdlib.h>
int size = 15;
static int index=0;

void swap(int arr[],int index1,int index2)
{
	int temp;

	temp=arr[index1];
	arr[index1] = arr[index2];
	arr[index2] =temp;
}

void Insert_MAX_Heap(int arr[],int value)		// MAX heap Insert
{
	int parent,child,i;
	index++;
	i=index;

	if(index >size+1)
	{
		printf("\n Heap is full");
	}
	else
	{
		arr[index] = value;

		while(i>1)
		{
			parent =i/2;
			child = arr[i];

			if(arr[parent] < child)
			{
				swap(arr,i/2,i);
				//index = parent;
				i=parent;
			}
			else
			{
				break;
			}
		}

	}
```

```
}

void Display(int arr[],int n)
{

	if(n==0)
	{
		printf("Queue is Empty");
	}
	else
	{
		printf("\n");
		for(int i=1;i<=n;i++)
		{
			printf("%d ",arr[i]);
		}
	}
}
void Delete_MAX_Heap(int arr[], int* N) //n is total members in queue
{
	//hear *N is used for make n value  n-1  after the function
	int i=1,left,right,last;
	int n=*N;


	if(n==0)
	{
		printf("\n queue is empty");
	}
	else
	{
		printf("\n Deleted item is : %d",arr[i]);

		while(i<n)
		{
			left = 2*i;
			right = 2*i + 1;
			last = n;
			if(left<n && right<n)
			{
				if(arr[left]<arr[right] )
				{
					if(arr[right]>arr[last])
					{
						arr[i]=arr[right];
						i=right;
					}
					else
```

```
                                {
                                        arr[i] = arr[last];
                                        i=last;
                                }
                        }
                        else if(arr[left]>arr[right] )
                        {
                                if( arr[left]>arr[last])
                                {
                                        arr[i] = arr[left];
                                        i=left;
                                }
                                else
                                {
                                        arr[i]=arr[last];
                                        i=last;
                                }
                        }
                }
                else
                {
                        break;
                }
        }
        if(i!=1)   // if i==1 then there is last node so we cant give any
value to it
        {
                arr[i] = arr[last];
        }
        n--;
        *N = n;
    }
}

void main()
{
    int n=0,i,ch,size,arr[20],value;
    char choice = 'n';

    printf("\n----------");
    printf("\n 1.Insert");
    printf("\n 2.Delete");
    printf("\n 3.Display");
    printf("\n 4.exit");
    printf("\n----------");

    do
    {
```

```
		printf("\n Enter your choice:");
		scanf("%d",&ch);

		switch(ch)
		{
			case 1:
					n++;
					printf("\n Enter node :");
					scanf("%d",&value);
					Insert_MAX_Heap(arr,value);

					printf("\n After Insert");
					for(int j=1;j<=n;j++)
					{
						printf("%d ",arr[j]);
					}
					break;
			case 2:
					Delete_MAX_Heap(arr,&n);
					break;
			case 3:
					Display(arr,n);
					break;
			case 4:
					exit(0);
					break;
		}
	}while(ch!=4);
	getch();
}
```

```
==============
OUTPUT
==============
----------
 1.Insert
 2.Delete
 3.Display
 4.exit
----------
 Enter your choice:1

 Enter node :31

 After Insert31
 Enter your choice:1

 Enter node :41
```

```
After Insert41 31
Enter your choice:1

Enter node :59

After Insert59 31 41
Enter your choice:1

Enter node :26

After Insert59 31 41 26
Enter your choice:1

Enter node :53

After Insert59 53 41 26 31
Enter your choice:1

Enter node :58

After Insert59 53 58 26 31 41
Enter your choice:1

Enter node :97

After Insert97 53 59 26 31 41 58
Enter your choice:2

Deleted item is : 97
Enter your choice:3

59 53 58 26 31 41
Enter your choice:2

Deleted item is : 59
Enter your choice:3

58 53 41 26 31
Enter your choice:2

Deleted item is : 58
Enter your choice:3

53 31 41 26
Enter your choice:2

Deleted item is : 53
```

```
 Enter your choice:3

41 31 26
 Enter your choice:2

 Deleted item is : 41
 Enter your choice:3

41 31
 Enter your choice:2

 Deleted item is : 41
 Enter your choice:3

41
 Enter your choice:2

 Deleted item is : 41
 Enter your choice:3
Queue is Empty
 Enter your choice:2

 queue is empty
 Enter your choice:4
```

==========
5)Heap Sort
==========

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>


void swap(int *num1,int *num2)
{
	int *temp;

	*temp=*num1;
	*num1 = *num2;
	*num2 =*temp;
}
void heapify(int arr[],int n,int i)
```

```
{
	int largest,l,r,temp;

	largest = i;
	l = 2*i+ 1 ;
	r = 2*i + 2;

	if(l<n && arr[l] > arr[largest])
	{
		largest = l;
	}
	if(r<n && arr[r] > arr[largest])
	{
		largest = r;
	}
	if(largest!=i)  //here we check that is largest is changed
	{
		//swap(&arr[i],&arr[largest]);

		temp = arr[i];
		arr[i]=arr[largest];
		arr[largest]=temp;

		heapify(arr,n,largest);
	}
}

void HeapSort(int arr[],int n)
{
	int temp;

	for(int i=n/2;i>=0;i--)
	{
		heapify(arr,n,i);
	}

	for(int i=n;i>=0;i--)
	{
		temp = arr[0];
		arr[0] = arr[i];
		arr[i] = temp;

		heapify(arr,i,0);

	}
}
```

```
void Display(int arr[],int n)
{

		for(int i=1;i<=n;i++)
		{
			printf("%d ",arr[i]);
		}

}

void main()
{
	int n,i,ch,size,arr[20],value;
	char choice = 'n';

	printf("\n Enter total elements");
	scanf("%d",&n);

	for(i=1;i<=n;i++)
	{
		printf("\n Enter value : ");
		scanf("%d",&arr[i]);
	}

	HeapSort(arr,n);

	printf("\n final \n");
	 for (int i=1; i<=n; ++i)
	  printf("%d ",arr[i]);


	getch();
}
```

```
========
OUTPUT
=======

 Enter total elements7

 Enter value : 30

 Enter value : 17

 Enter value : 15

 Enter value : 1
```

```
 Enter value : 5

 Enter value : 10

 Enter value : 20

 final
1 5 10 15 17 20 30

========
OUTPUT
=======

 Enter total elements
10

 Enter value : 1

 Enter value : 10

 Enter value : 2

 Enter value : 3

 Enter value : 4

 Enter value : 1

 Enter value : 2

 Enter value : 100

 Enter value : 23

 Enter value : 2

 final
1 1 2 2 2 3 4 10 23 100
```

```
====================
6)Quick Sort
====================
```

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

void GetData(int num[],int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		printf("\n Enter Value:");
		scanf("%d",&num[i]);
	}

}
void Display(int num[],int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		printf("%d ",num[i]);
	}

}




void quicksort(int number[25],int first,int last){
   int i, j, pivot, temp;

   if(first<last){
      pivot=first;
      i=first;
      j=last;

      while(i<j){
         while(number[i]<=number[pivot]&&i<last)
            i++;
         while(number[j]>number[pivot])
            j--;
         if(i<j){
            temp=number[i];
            number[i]=number[j];
            number[j]=temp;
         }
      }
```

```
    temp=number[pivot];
    number[pivot]=number[j];
    number[j]=temp;
    quicksort(number,first,j-1);
    quicksort(number,j+1,last);

  }
}
void main()
{
       int num[20],i,n;

       printf("\n Enter value of n:");
       scanf("%d",&n);

       GetData(num,n);
       printf("\n\n Before Sorted..\n");
       Display(num,n);

       quicksort(num,0,n-1);
       printf("\n\n After Sorted..\n");
       Display(num,n);
       getch();
}
```

==========
OUTPUT
==========

```
 Enter value of n:7

 Enter Value:6

 Enter Value:4

 Enter Value:5

 Enter Value:1

 Enter Value:2

 Enter Value:7

 Enter Value:3


 Before Sorted..
6 4 5 1 2 7 3
```

```
 After Sorted..
1 2 3 4 5 6 7
```

```
=======================
7)Radix Sort
=======================

#include<stdio.h>
#include<conio.h>
#include <math.h>
#include<malloc.h>


struct ArrayNode{
	int data;
	struct node * next;
	struct node * inerrNodeLast;
};

struct node {
	int data;
	struct node *next;
};

struct node * InsertEnd(struct node **first,struct node **last ,int data)
{

	struct node *new_node=NULL;
	new_node=(struct node *)malloc(sizeof(struct node));

	new_node->data = data;
	new_node->next=NULL;

	if((*first)==NULL)
	{
		(*first)=new_node;
		(*last)=*first;
	}
	else
	{
		(*last)->next=new_node;
		(*last)=new_node;
	}
```

```
	return (*first);
}

int max(int arr[], int n)
{
	int i,max,digit=0;

	max=arr[0];

	for(i=1;i<n;i++)
	{
		if(max<arr[i])
			max =arr[i];
	}

	return max;
}

void Display(struct ArrayNode struct_arr[])
{
		struct node *p;

	for(int i=0;i<10;i++)
	{
		printf("\n %d --->",struct_arr[i].data);

		p=struct_arr[i].next;

		while(p!=NULL)
		{
			printf("%d ",p->data);
			p=p->next;
		}


	}
}
void main()
{
	int i,j,array_data[20];
	int n,data,rem,max_digit, modValue;

	int index=0,array_data2[20],w,m,k;

	struct ArrayNode struct_arr[10];
	struct node *p;

	printf("\n Enter n:");
```

```
scanf("%d",&n);

for(i=0;i<n;i++)
{
	printf("\n Enter data:");
	scanf("%d",&array_data[i]);
}

max_digit = max(array_data,n);

for(w=1;max_digit/w>0;w*=10)
{

	for( k=0;k<10;k++)
	{
		struct_arr[k].data = k;
		struct_arr[k].next = NULL;
		struct_arr[k].inerrNodeLast =NULL;
	}


	for(i=0;i<n;i++)
	{

		rem = (array_data[i] /w) % 10;

		for(j=0;j<10;j++)
		{
			struct node *first = NULL;
			struct node *last= NULL;

			if(rem == j)  //struct_arr[k].data
			{
				if(struct_arr[j].next == NULL)
				{

first=InsertEnd(&first,&last,array_data[i]);
					struct_arr[j].next = first;
					struct_arr[j].inerrNodeLast = last;
				}
				else
				{
					first = struct_arr[j].next;
					last = struct_arr[j].inerrNodeLast;

first=InsertEnd(&first,&last,array_data[i]);
					struct_arr[j].next = first;
					struct_arr[j].inerrNodeLast = last;
```

```
                                }
                        }//if
                }//end of for j

        }// end of for i
        index=0;
        for(int o=0;o<10;o++)
        {

                //printf("\n %d --->",o);

                p=struct_arr[o].next;

                while(p!=NULL)
                {
                        array_data[index]=p->data;
                //      printf("--> %d ",p->data);
                        p=p->next;
                        index++;

                }

        }

        printf("\n Array\n----------\n");
        for( m =0 ;m<n;m++)
        {
                printf("%d ",array_data[m]);
        }
    }
    getch();
}
```

==========
OUTPUT
==========

```
 Enter n:10

 Enter data:15

 Enter data:1

 Enter data:321

 Enter data:10

 Enter data:802
```

```
 Enter data:2

 Enter data:123

 Enter data:90

 Enter data:109

 Enter data:11

 Array
------------
10 90 1 321 11 802 2 123 15 109
 Array
------------
1 802 2 109 10 11 15 321 123 90
 Array
------------
1 2 10 11 15 90 109 123 321 802
```

==========
8) Shell Sort
==========

```
#include<stdio.h>
#include<conio.h>
#include <math.h>

void swap(int arr[],int index1,int index2)
{
	int temp;

	temp=arr[index1];
	arr[index1] = arr[index2];
```

```
	arr[index2] =temp;
}
void main()
{
	int arr[20] ; //= {81,94,11,96,12,35,17,95,28,58,41,75,15};

	int n; // = 13;
	printf("\n Enter n:");
	scanf("%d",&n);

	for(int i=0;i<n;i++)
	{
		printf("\n Enter Value:");
		scanf("%d",&arr[i]);
	}



	int increment =5;
	int tempIndex;

	printf("\n \nbefore Sorting");
	for(int j=0;j<n;j++)
	{
		printf("%d ",arr[j]);
	}
	printf("\n ------------");

	while(increment!=0)
	{
		printf("\nFor Increment %d",increment);
		printf("\n--------------------------");
		for(int i=0;i<n;i++)
		{
			if(i + increment<n)
			{
				tempIndex = i + increment;
				if(arr[i] > arr[tempIndex])
				{
					swap(arr,i,tempIndex);
				}
			}

			if((i - increment )>=0)
			{
				tempIndex = i - increment;
				if(arr[i] < arr[tempIndex])
				{
```

```
                                swap(arr,i,tempIndex);
                        }
                }

                printf("\n");

                for(int j=0;j<n;j++)
                {
                        printf("%d ",arr[j]);
                }
        }
        increment =(increment/2);
    }


    printf(" \n \nAfter Sorting");
    for(int j=0;j<n;j++)
    {
        printf("%d ",arr[j]);
    }

    getch();
}
```

==========
OUTPUT
==========

Enter n:13

Enter Value:81

Enter Value:94

Enter Value:11

Enter Value:96

Enter Value:12

Enter Value:35

Enter Value:17

Enter Value:95

Enter Value:28

```
 Enter Value:58

 Enter Value:41

 Enter Value:75

 Enter Value:15


before Sorting81 94 11 96 12 35 17 95 28 58 41 75 15


For Increment 5
--------------------------
35 94 11 96 12 81 17 95 28 58 41 75 15
35 17 11 96 12 81 94 95 28 58 41 75 15
35 17 11 96 12 81 94 95 28 58 41 75 15
35 17 11 28 12 81 94 95 96 58 41 75 15
35 17 11 28 12 81 94 95 96 58 41 75 15
35 17 11 28 12 41 94 95 96 58 81 75 15
35 17 11 28 12 41 75 95 96 58 81 94 15
35 17 11 28 12 41 75 15 96 58 81 94 95
35 17 11 28 12 41 75 15 96 58 81 94 95
35 17 11 28 12 41 75 15 96 58 81 94 95
35 17 11 28 12 41 75 15 96 58 81 94 95
35 17 11 28 12 41 75 15 96 58 81 94 95
For Increment 2
--------------------------
11 17 35 28 12 41 75 15 96 58 81 94 95
11 17 35 28 12 41 75 15 96 58 81 94 95
11 17 12 28 35 41 75 15 96 58 81 94 95
11 17 12 28 35 41 75 15 96 58 81 94 95
11 17 12 28 35 41 75 15 96 58 81 94 95
11 17 12 15 35 28 75 41 96 58 81 94 95
11 17 12 15 35 28 75 41 96 58 81 94 95
11 17 12 15 35 28 75 41 96 58 81 94 95
11 17 12 15 35 28 75 41 81 58 96 94 95
11 17 12 15 35 28 75 41 81 58 96 94 95
11 17 12 15 35 28 75 41 81 58 95 94 96
11 17 12 15 35 28 75 41 81 58 95 94 96
11 17 12 15 35 28 75 41 81 58 95 94 96
For Increment 1
--------------------------
11 17 12 15 35 28 75 41 81 58 95 94 96
11 12 17 15 35 28 75 41 81 58 95 94 96
11 12 15 17 35 28 75 41 81 58 95 94 96
```

```
11 12 15 17 35 28 75 41 81 58 95 94 96
11 12 15 17 28 35 75 41 81 58 95 94 96
11 12 15 17 28 35 75 41 81 58 95 94 96
11 12 15 17 28 35 41 75 81 58 95 94 96
11 12 15 17 28 35 41 75 81 58 95 94 96
11 12 15 17 28 35 41 58 75 81 95 94 96
11 12 15 17 28 35 41 58 75 81 95 94 96
11 12 15 17 28 35 41 58 75 81 94 95 96
11 12 15 17 28 35 41 58 75 81 94 95 96
11 12 15 17 28 35 41 58 75 81 94 95 96

After Sorting11 12 15 17 28 35 41 58 75 81 94 95 96


==========================
9) Merge Sort
==========================


#include<stdio.h>
#include<conio.h>
#include<malloc.h>

void GetData(int num[],int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		printf("\n Enter Value:");
		scanf("%d",&num[i]);
	}

}
void Display(int num[],int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		printf("%d ",num[i]);
	}

}


void merge_sort(int i, int j, int a[], int temp[]) {
   if (j <= i) {
      return;
   }
```

```
   int mid = (i + j) / 2;

   merge_sort(i, mid, a, temp);
   merge_sort(mid + 1, j, a, temp);

   int left = i;
   int right = mid + 1;
   int k;

   for (k = i; k <= j; k++) {
      if (left == mid + 1) {      // left pointer has reached the limit
         temp[k] = a[right];
         right++;
      } else if (right == j + 1) {       // right pointer has reached the limit
         temp[k] = a[left];
         left++;
      } else if (a[left] < a[right]) {       // pointer left points to smaller element
         temp[k] = a[left];
         left++;
      } else {       // pointer right points to smaller element
         temp[k] = a[right];
         right++;
      }
   }

   for (k = i; k <= j; k++) {      // copy the elements from temp[] to a[]
      a[k] = temp[k];
   }
}

void main()
{
       int num[20],i,n,temp[100];

       printf("\n Enter value of n:");
       scanf("%d",&n);

       GetData(num,n);
       printf("\n\n Before Sorted..\n");
       Display(num,n);

       merge_sort(0, n - 1, num, temp);
       printf("\n\n After Sorted..\n");
       Display(num,n);
       getch();
}

============
```

OUTPUT
==========

Enter value of n:7

Enter Value:6

Enter Value:4

Enter Value:5

Enter Value:1

Enter Value:2

Enter Value:7

Enter Value:3

Before Sorted..
6 4 5 1 2 7 3

After Sorted..
1 2 3 4 5 6 7

==========
10) BFS , DFS
==========

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>

int front =-1;
int rear = -1;


void insert(int key,int size,int queue[])
{
        if(rear==(size-1))
        {
```

```
		printf("\n Overflow");
	}
	else
	{
		if(front==-1)
		{
			front=0;
		}
		rear +=1;
		queue[rear]=key;
	}
}

void del(int queue[])
{
	if(front==-1)
	{
		printf("\n underflow");

	}
	else if(front==rear)
	{
		//printf("\n Deleted item is:%d",queue[front]);
		front=-1;
		rear=-1;
	}
	else
	{
		//printf("\n Deleted item is:%d",queue[front]);
		front=front+1;
	}
}




struct node{
	int data;
	struct node * next;
};

struct table{
	int node_name_value;
	int known;
	int distance;
	int  path;
};
```

```
struct node * InsertEnd(struct node **first,struct node **last )
{
	struct node *new_node=NULL;
	new_node=(struct node *)malloc(sizeof(struct node));

	printf("\n Enter Adjacent:");
	scanf("%d",&new_node->data);
	new_node->next=NULL;

	if((*first)==NULL)
	{
		(*first)=new_node;
		(*last)=*first;
	}
	else
	{
		(*last)->next=new_node;
		(*last)=new_node;
	}

	return (*first);
}


void display_table(struct table t1[], int total_vertex)
{
	printf("\n V\t Known\t Dv\t Pv");
	for(int i=0;i<7;i++)
	{
		printf("\n %d\t %d\t %d\t
%d",t1[i].node_name_value,t1[i].known,t1[i].distance,t1[i].path);
	}
}




void sortest_path(struct table t1[], int total_vertex,struct node n1[])  // this is
for BFS
{
	int reachable,V;
	int cur_dis=0;
	struct node *W;   //for recahble
	int i,start;
	int queue[10];
	int size=10;

	int bfsIndex = 0;  // bfs index
```

```
	int BFS[10]; //breath first search



	printf("\n Enter starting node:");
	scanf("%d",&start);

	insert(start,size,queue);
	BFS[bfsIndex] = start;
	bfsIndex++;

	while(front!=-1)
	{
		V = queue[front];

				W = n1[V-1].next; //address of list


				t1[V-1].known = 1;
				del(queue);

				while(W!=NULL)
				{
					reachable = W->data;
					if(t1[reachable-1].distance == -1)
					{
						t1[reachable-1].distance = t1[V-
1].distance + 1;
						t1[reachable-1].path =V;
						insert(reachable,size,queue);
						BFS[bfsIndex] = reachable;
						bfsIndex++;
					}

					W=W->next;
				}
	}

	printf("\n-----BFS-----\n");
	for(int i=0;i<bfsIndex;i++)
	{
		printf("V%d ",BFS[i]);
	//	bfsIndex++;

	}
}

int visited[10];
```

```
void DFS(struct table t1[], int total_vertex,struct node n1[],int start)
{
	int reachable,V;
	int cur_dis=0;
	struct node *W;   //for recahble
	int i;
	int stack[10],size=10;

	visited[start-1]=1;
	printf("V%d  ",start);

	W = n1[start-1].next;	//address of list


				while(W!=NULL)
				{
					reachable = W->data;
					if(visited[reachable-1] !=1)
					{
						DFS(t1,total_vertex,n1,reachable);
					}
					W=W->next;
				}

}



void main()
{
	struct node n1[7];
	struct node *p;
	int n,n2,total_vertex,start;

	printf("\n Enter how many nodes you create n:");
	scanf("%d",&total_vertex);		//total vertex = total V1,V2...V7

	for(int i=0;i<total_vertex;i++)
	{
		printf("\n Enter graph Node::");
		scanf("%d",&n1[i].data);

		n1[i].next =NULL;
		struct node *first =NULL;
		struct node *last =NULL;
```

```
		printf("\n Enter total number of adjacent of %d ::",n1[i].data);
		scanf("%d",&n2);



		for(int j=0;j<n2;j++)
		{
			first=InsertEnd(&first,&last);
			n1[i].next =first;
		}
	}


	struct table t1[7];

	//INITIAL TABLE

	printf("\n Enter starting node:");
	scanf("%d",&start);


	for(int i=0;i<total_vertex;i++)
	{
		t1[i].node_name_value = i+1;
		t1[i].known =0;

		if(i==(start -1))
		{
			t1[i].distance = 0;
		}
		else
		{
			t1[i].distance = -1;
		}
		t1[i].path = 0;

	}

	//BFS
	sortest_path(t1,total_vertex,n1);

	printf("\n---------\n   DFS   \n---------\n");
	DFS(t1,total_vertex,n1,start);

	getch();
}
```

==========
OUTPUT
==========

```
 Enter how many nodes you create n:7

 Enter graph Node::1

 Enter total number of adjacent of 1 ::2

 Enter Adjacent:2

 Enter Adjacent:4

 Enter graph Node::2

 Enter total number of adjacent of 2 ::2

 Enter Adjacent:4

 Enter Adjacent:5

 Enter graph Node::3

 Enter total number of adjacent of 3 ::2

 Enter Adjacent:1

 Enter Adjacent:6

 Enter graph Node::4

 Enter total number of adjacent of 4 ::3

 Enter Adjacent:6

 Enter Adjacent:7

 Enter Adjacent:5

 Enter graph Node::5

 Enter total number of adjacent of 5 ::1

 Enter Adjacent:7

 Enter graph Node::6
```

```
 Enter total number of adjacent of 6 ::0

 Enter graph Node::7

 Enter total number of adjacent of 7 ::1

 Enter Adjacent:6

 Enter starting node:1

 Enter starting node:1

-----BFS-----
V1 V2 V4 V5 V6 V7

---------
  DFS
---------
V1  V2  V4  V6  V7  V5
```

```
================================
11)Prims Minimum Spanning Tree
================================

#include<stdio.h>
#include<conio.h>
#include<malloc.h>

int front =-1;
int rear = -1;

struct node{
        int data;
        int weight;
        struct node * next;
};

struct table{
        int node_name_value;
        int known;
        int distance;
        int  path;
};
struct node * InsertEnd(struct node **first,struct node **last )
```

```
{
	struct node *new_node=NULL;
	new_node=(struct node *)malloc(sizeof(struct node));

	printf("\n Enter Adjacent:");
	scanf("%d",&new_node->data);


	printf("\n Enter Weight:");
	scanf("%d",&new_node->weight);

	new_node->next=NULL;

	if((*first)==NULL)
	{
		(*first)=new_node;
		(*last)=*first;
	}
	else
	{
		(*last)->next=new_node;
		(*last)=new_node;
	}

	return (*first);
}

void display(struct node n1[], int n)
{
	struct node *p;
	int i;
	for(int i=0;i<n;i++)
	{
		printf("\n %d --->",n1[i].data);

		p=n1[i].next;

		while(p!=NULL)
		{
			printf("%d ",p->data);
			p=p->next;
		}
	}

}

void display_table(struct table t1[], int total_vertex)
{
```

```
	printf("\n V\t Known\t Dv\t Pv");
	for(int i=0;i<total_vertex;i++)
	{
		printf("\n %d\t %d\t %d\t
%d",t1[i].node_name_value,t1[i].known,t1[i].distance,t1[i].path);
	}
}

void insert(int key,int size,int queue[])
{
	if(rear==(size-1))
	{
		printf("\n Overflow");
	}
	else
	{
		if(front==-1)
		{
			front=0;
		}
		rear +=1;
		queue[rear]=key;
	}
}

void del(int queue[])
{
	if(front==-1)
	{
		printf("\n underflow");

	}
	else if(front==rear)
	{
		//printf("\n Deleted item is:%d",queue[front]);
		front=-1;
		rear=-1;
	}
	else
	{
		//printf("\n Deleted item is:%d",queue[front]);
		front=front+1;
	}
}


int MIN(struct table t1[], int total_vertex)
{
```

```
	int min,i,j,index=0,flag=0;

	for(i=0;i<total_vertex;i++)
	{
		if(t1[i].known==0 &&  t1[i].distance!=-1)
		{
			min=t1[i].distance;
			index = i;

			for(j=i+1;j<total_vertex;j++)
			{
				if(t1[j].known==0 &&  t1[j].distance!=-1)
				{
					if(min > t1[j].distance)
					{
						min = t1[j].distance;
						index = j;
						flag=1;
					}
				}
			}
			if(flag==1){i=j;}
		}
	}
	return index;
}



void sortest_path(struct table t1[], int total_vertex,struct node n1[])
{
	int reachable,V;
	int cur_dis=0;
	struct node *W;   //for recahble
	int i,start;
	int queue[10];
	int size=10,min,index;



	for(i=0;i<total_vertex;i++)
	{
		min = MIN(t1 , total_vertex);


		V = n1[min].data;
		W = n1[min].next;
```

```
                t1[min].known = 1;

                        while(W!=NULL)
                        {
                                reachable = W->data;

                                if(t1[reachable-1].distance == -1)
                                {
                                        t1[reachable-1].distance = t1[V-
1].distance;
                                    t1[reachable-1].path =V;
                                }
                        /*      else
                                {
                                        if(t1[V-1].distance <= t1[reachable-
1].distance)
                                        //if(t1[reachable-1].distance == -1)
                                        {
                                                t1[reachable-1].distance = t1[V-
1].distance;
                                                t1[reachable-1].path =V;
                                        }

                                }
                                */
                                W=W->next;

                        }

                printf("\n \n ");
                display_table(t1,total_vertex);
        }
}


void main()
{
        struct node n1[7];
        struct node *p;
        int n,n2,total_vertex,start;

        printf("\n Enter how many nodes you create n:");
        scanf("%d",&total_vertex);              //total vertex = total V1,V2...V7

        for(int i=0;i<total_vertex;i++)
        {
                printf("\n Enter graph Node::");
                scanf("%d",&n1[i].data);
```

```
        n1[i].next =NULL;
        struct node *first =NULL;
        struct node *last =NULL;


        printf("\n Enter total number of adjacent of %d ::",n1[i].data);
        scanf("%d",&n2);


        for(int j=0;j<n2;j++)
        {
                first=InsertEnd(&first,&last);
                n1[i].next =first;
        }
}

display(n1,total_vertex);

printf("\n Enter starting node:");
scanf("%d",&start);



struct table t1[7];

//INITIAL TABLE


for(int i=0;i<total_vertex;i++)
{
        t1[i].node_name_value = i+1;
        t1[i].known =0;

        if(i==(start-1))
        {
                t1[i].distance = 0;
        }
        else
        {
                t1[i].distance = -1;
        }
        t1[i].path = 0;

}


display_table(t1,total_vertex);  //initial table
```

```
        //DISPLAY
        sortest_path(t1,total_vertex,n1);


        getch();
}
```

OUTPUT

```
 Enter how many nodes you create n:5

 Enter graph Node::1

 Enter total number of adjacent of 1 ::2

 Enter Adjacent:2

 Enter Weight:6

 Enter Adjacent:3

 Enter Weight:1

 Enter graph Node::2

 Enter total number of adjacent of 2 ::1

 Enter Adjacent:4

 Enter Weight:2

 Enter graph Node::3

 Enter total number of adjacent of 3 ::1

 Enter Adjacent:2

 Enter Weight:2

 Enter graph Node::4

 Enter total number of adjacent of 4 ::2

 Enter Adjacent:3

 Enter Weight:1
```

```
Enter Adjacent:5

Enter Weight:2

Enter graph Node::5

Enter total number of adjacent of 5 ::1

Enter Adjacent:3

Enter Weight:2

1 --->2 3
2 --->4
3 --->2
4 --->3 5
5 --->3
Enter starting node:1

V      Known   Dv      Pv
1      0       0       0
2      0       -1      0
3      0       -1      0
4      0       -1      0
5      0       -1      0


V      Known   Dv      Pv
1      1       0       0
2      0       0       1
3      0       0       1
4      0       -1      0
5      0       -1      0


V      Known   Dv      Pv
1      1       0       0
2      0       0       1
3      1       0       1
4      0       -1      0
5      0       -1      0


V      Known   Dv      Pv
1      1       0       0
2      1       0       1
3      1       0       1
```

```
4       0       0       2
5       0       -1      0


V       Known   Dv      Pv
1       1       0       0
2       1       0       1
3       1       0       1
4       1       0       2
5       0       0       4


V       Known   Dv      Pv
1       1       0       0
2       1       0       1
3       1       0       1
4       1       0       2
5       1       0       4
```

==========
12)Kruskals Minimum Spanning Tree
==========

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int size = 15;
static int index=0;

struct node{
        int data;
        int weight;
        struct node * next;
};


struct node * InsertEnd(struct node **first,struct node **last )
{
        struct node *new_node=NULL;
        new_node=(struct node *)malloc(sizeof(struct node));

        printf("\n Enter Adjacent:");
        scanf("%d",&new_node->data);
```

```
	printf("\n Enter Weight:");
	scanf("%d",&new_node->weight);

	new_node->next=NULL;

	if((*first)==NULL)
	{
		(*first)=new_node;
		(*last)=*first;
	}
	else
	{
		(*last)->next=new_node;
		(*last)=new_node;
	}

	return (*first);
}

void swap(int arr[][20],int index1,int index2)
{
	int temp;

	temp=arr[index1][2];
	arr[index1][2] = arr[index2][2];
	arr[index2][2] =temp;
}

void BuildHeap(int arr[][20],int size)		// Prioity queue insert / Min heap
Insert
{
	int temp[3] = {0};
	for(int row=0;row<size;row++){
		for(int row2=row+1;row2<size;row2++){
			if(arr[row][2]>arr[row2][2]){
				temp[0] = arr[row][0];
				temp[1] = arr[row][1];
				temp[2] = arr[row][2];

				arr[row][0] = arr[row2][0];
				arr[row][1] = arr[row2][1];
				arr[row][2] = arr[row2][2];

				arr[row2][0] = temp[0];
				arr[row2][1] =temp[1];
				arr[row2][2] =temp[2];
			}
```

```
		}
	}

}


void Display(int arr[],int n)
{

	if(n==0)
	{
		printf("Queue is Empty");
	}
	else
	{
		printf("\n");
		for(int i=1;i<=n;i++)
		{
			printf("%d ",arr[i]);
		}
	}
}
void DeleteMin(int arr[][20], int* N) //n is total members in queue
{
	//hear *N is used for make n value  n-1  after the function
	int i=1,left,right,last;
	int n=*N;


	if(n==0)
	{
		printf("\n queue is empty");
	}
	else
	{
		printf("\n Deleted item is : %d",arr[i]);

		while(i<n)
		{
			left = 2*i;
			right = 2*i + 1;
			last = n;
			if(left<n && right<n)
			{
				if(arr[left][2]>arr[right][2] )
				{
					if(arr[right][2]<arr[last][2])
					{
```

```
                                        arr[i][2]=arr[right][2];
                                        i=right;
                                }
                                else
                                {
                                        arr[i][2] = arr[last][2];
                                        i=last;
                                }
                        }
                        else if(arr[left][2]<arr[right][2] )
                        {
                                if( arr[left][2]<arr[last][2])
                                {
                                        arr[i][2] = arr[left][2];
                                        i=left;
                                }
                                else
                                {
                                        arr[i][2]=arr[last][2];
                                        i=last;
                                }
                        }
                }
                else
                {
                        break;
                }
        }
        if(i!=1)   // if i==1 then there is last node so we cant give any
value to it
        {
                arr[i][2] = arr[last][2];
        }
        n--;
        *N = n;
    }
}

void Initialize(int S[] ,int totalNodes)
{

    for(int k=0;k <(totalNodes + 2);k++)
    {
        S[k] = 0;
        printf(" -> %d",k);
    }

}
```

```
int Find(int x,int S[])
{
	if(S[x]==0)
	{
		return x;
	}
	else
		return Find(S[x],S);
}

void SetUnion(int S[], int root1,int root2)
{
	S[root2] = root1;
}
int ReadGraphIntoHeapArray(struct node graph[],int HeapArray[][20], int
totalNodes)
{
	struct node *p;
	int i=0,j,res;
	for(int j=0;j<totalNodes;j++)
	{
		printf("\n %d --->",graph[j].data);

		res=graph[j].data;
		p=graph[j].next;


		while(p!=NULL)
		{
			HeapArray[i][0]=res;

			HeapArray[i][1]=p->data;
			HeapArray[i][2]=p->weight;
			i++;
			printf("%d ",p->data);
			p=p->next;

		}
	}
	return i;

}
void Kruskals(struct node graph[] ,int totalNodes){

	int S[20];  // s is set
```

```
	int HeapArray[20][20];  // array for heap
	int edgesAcepted;
	int Uset,Vset,U,V;
	int flag=0;
	//Initialize(S,totalNodes);  //initialize array/set with zero



	size = ReadGraphIntoHeapArray(graph,HeapArray,totalNodes);
	for(int k=0;k<(totalNodes +1);k++)
	{
		S[k] = 0;
	}
	printf("\n Before Sorting");
	for(int i=0;i<size;i++)
	{
		printf(" \n %d %d %d",HeapArray[i][0],HeapArray[i]
[1],HeapArray[i][2]);
	}

	BuildHeap(HeapArray,size);  // sorting array
	printf("\n After Sorting");

	for(int i=0;i<size;i++)
	{
		printf(" \n\n %d %d %d",HeapArray[i][0],HeapArray[i]
[1],HeapArray[i][2]);
	}
	int index2 =0;
	edgesAcepted =0;

	while(index2 < size )  //total nodes
	{

		U = HeapArray[index2][0];
		V = HeapArray[index2][1];
		printf("\n U-> %d   V-> %d   ",U,V);
		Uset = Find(U,S);
		Vset = Find(V,S);
		//printf("Ust-> %d   Vset-> %d   ",Uset,Vset);
		flag=0;
		if(Uset !=Vset)
		{
			flag=1;
			edgesAcepted++;
			SetUnion(S,Uset,Vset);
		}
		index2++;
```

```
		if(flag==1)
			printf("-----> Accepted");
		else
			printf("------> NotAccepted");
	}

	printf("\n------------------\n");
	for(int k=1;k<(totalNodes+1);k++)
	{
		printf("  %d" , S[k]);
	}
}

void main()
{
	struct node n1[7];   //n1 is the graph
	struct node *p;
	int n,n2,total_vertex,start;



	printf("\n Enter how many nodes you create n:");
	scanf("%d",&total_vertex);              //total vertex = total V1,V2...V7

	for(int i=0;i<total_vertex;i++)
	{
		printf("\n Enter graph Node::");
		scanf("%d",&n1[i].data);

		n1[i].next =NULL;
		struct node *first =NULL;
		struct node *last =NULL;


		printf("\n Enter total number of adjacent of %d ::",n1[i].data);
		scanf("%d",&n2);


		for(int j=0;j<n2;j++)
		{
			first=InsertEnd(&first,&last);
			n1[i].next =first;
		}
	}



	Kruskals(n1,total_vertex);
```

```
        getch();
}
```

```
============
OUTPUT
============
```

```
 Enter how many nodes you create n:5

 Enter graph Node::1

 Enter total number of adjacent of 1 ::2

 Enter Adjacent:2

 Enter Weight:6

 Enter Adjacent:3

 Enter Weight:1

 Enter graph Node::2

 Enter total number of adjacent of 2 ::1

 Enter Adjacent:4

 Enter Weight:2

 Enter graph Node::3

 Enter total number of adjacent of 3 ::1

 Enter Adjacent:2

 Enter Weight:2

 Enter graph Node::4

 Enter total number of adjacent of 4 ::2

 Enter Adjacent:5

 Enter Weight:2

 Enter Adjacent:3
```

```
 Enter Weight:1

 Enter graph Node::5

 Enter total number of adjacent of 5 ::1

 Enter Adjacent:3

 Enter Weight:2

 1 --->2 3
 2 --->4
 3 --->2
 4 --->5 3
 5 --->3
Before Sorting
 1 2 6
 1 3 1
 2 4 2
 3 2 2
 4 5 2
 4 3 1
 5 3 2
 After Sorting

 1 3 1

 4 3 1

 3 2 2

 4 5 2

 2 4 2

 5 3 2

 1 2 6
 U-> 1   V-> 3   -----> Accepted
 U-> 4   V-> 3   -----> Accepted
 U-> 3   V-> 2   -----> Accepted
 U-> 4   V-> 5   -----> Accepted
 U-> 2   V-> 4   ------> NotAccepted
 U-> 5   V-> 3   ------> NotAccepted
 U-> 1   V-> 2   ------> NotAccepted
-------------------
  4  4  1  0  4
```