DATE
PAGE No

Assignment

1.) Consider array exam [-3:2, 4:7]. Find address of exam [-1,7]. If array is row major and column major. Base address = 4000. w = 4 bytes

Ans Base Address = 4000
w = 4 bytes

Exam $[L_1 : U_1, L_2 : U_2]$
$L_1 = -3 \quad U_1 = 2$
$L_2 = 4 \quad U_2 = 7$
To Find Exam[i][j]. i = -1, j = 7

(1) Row major

Exam[-1][7] = $4000 + [((i-L_1) \times (U_2 - L_2 + 1) + (j - L_2)) \times w]$
$= 4000 + [((-1+3) \times (7-4+1) + (7-4)) \times 4]$
$= 4000 + 11 \times 4$
$= 4000 + 44$
$= 4044$

(2) Column major

Exam[-1][7] = $4000 + [((j-L_2) \times (U_1 - L_1 + 1) + (i - L_1)) \times w]$
$= 4000 + [((7-4) \times (2+3+1) + (-1+3)) \times 4]$
$= 4000 + 20 \times 4$
$= 4000 + 80$
$= 4080$

DATE
PAGE No

2.) Given 2d array $z_1[3:10, 10:20]$ stored in row major order. Base address = 200, w = 4. Find address $z_1[5, 15]$

Ans Base Address = 200
w = 4 bytes

$z_1[L_1 : U_1, L_2 : U_2]$
$L_1 = 3 \quad U_1 = 10$
$L_2 = 10 \quad U_2 = 20$
To find $z_1[i][j]$. $i = 5, j = 15$

(1) Row Major

$$z_1[5][15] = 200 + ((i - L_1) \times (U_2 - L_2 + 1) + (j - L_2)) \times w)$$
$$= 200 + ((15 - 3) \times (20 - 10 + 1) + (15 - 10)) \times 4)$$
$$= 200 + ((12 \times 11 + 5) \times 4)$$
$$= 200 + 27 \times 4$$
$$= 200 + 108$$
$$= 308$$

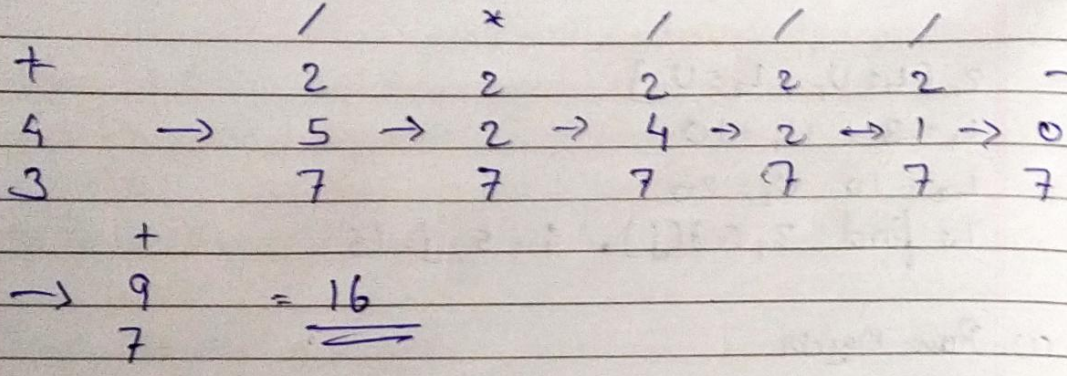
3) Evaluate with the help of stack.

34 + 5 2 / 2 * 2 / 2 / 2 / - 9 +

+		2		2
4	→	5	→	5
3		7		7

3.] Evaluate with the help of stack

34+52/2*2/2/2/-9+



4.] Convert infix to prefix using stack

(A^B)/D*E / (F+A)*(D*E)-C

	Stack	Prefix exp.
C		
-	-	CED*AF+EDBAA/
)	-,)	*/*-
E		
*	-,), *	
D		
(	-, *	
*		
)	-, *,)	

19CE138

DATE
PAGE No

A	
+	-, *,), +
F	
(	
/	-, *, /
E	
*	-, *, /, *
D	
/	-, *, /, *, /
)	-, *, /, *, /,)
B	
^	-, *, /, *, /,), ^
(	

* Final Prefix expression

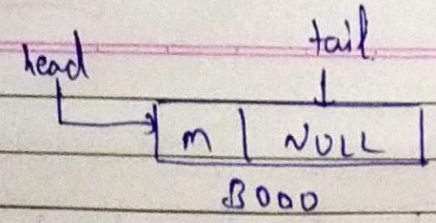
- * / * / ^ A B D E + F A * D E C

5) Evaluate with the help of stack.
ABC + * CBA - + *

A=6, B=12, C=-2
6 12 -2 + * -2 12 6 - + *

+				6		+					
-2		*		12		6		*			
12	→	10	→	-2	→	-2	→	4	→	240	
6		6		60		60		60			

6)

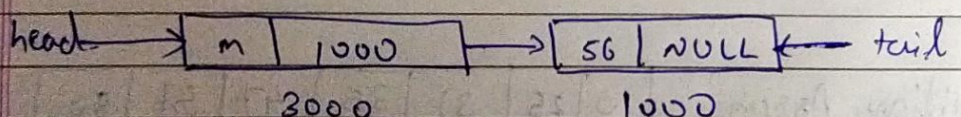


-> For inserting a new node at end.

- We insert a new node which contains data 56 and address after first node.
- suppose we have two pointer head and tail. They are pointing at the first node
- For inserting a new node at end, first of all we create a new node.

```
struct node newnode = (struct node *) malloc
                      (sizeof (struct node))
```

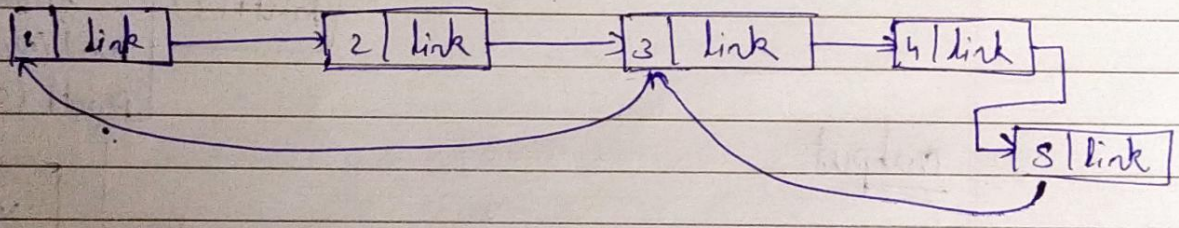
- Then for insert we move our tail pointer
  tail → next = new node
- we assign address of new node in first node then we move tail pointer to new node
- In last, the scenario is -



7)
```
void fun1 (struct node * head)
{
    if (head == NULL)
        return;
    fun1 (head -> next -> next);
    printf("%d", head -> data);
}
```

So, the given function prints the alternate node (i.e. odd nodes) present in linked list in reverse manner.

For example, we have linked list containing 1 → 2 → 3 → 4 → 5 then it will give output 5 3 1



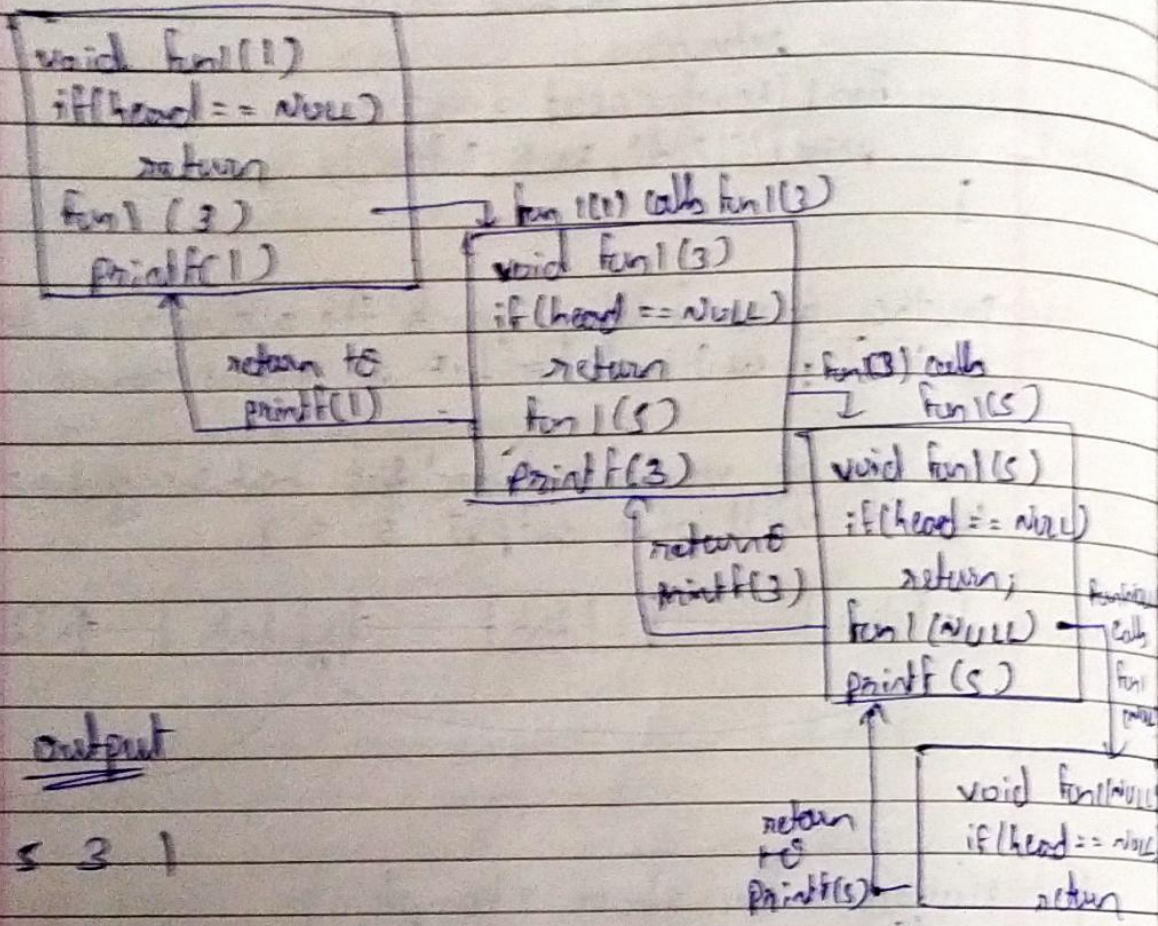
Recursion for above example

1 → 2 → 3 → 4 → 5

head

```
void fun1(1)
if (head == NULL)
    return;
    fun1 (3)
    printf(1)
```

Recursion function example



output

5 3 1

Q) What are advantages of circular queue over simple queue? Explain pseudocode of enqueue() and dequeue() operation

→ In circular queue, the front end and the rear end are next to each other. As a result, if the rear end is full even when the front end has space, data can be stored in the later section until there is an overflow.

→ Thus free space is effectively utilized.

→ However in normal queue the front and rear ends are at opposite corners. Thus, even if the queue has a full rear end and only partially filled front end, the space cannot be used to store data as insertion starts from rear end which is full.

* Enqueue Algorithm

enqueue (Q, R, F, x, N)

1 if (R == N)
    "Overflow"
  else
    R = R+1

2 Q[R] = x

3 if (F == -1)
F = 0

Step-1 check if the queue is full.

Step-2 If the queue is full, produce overflow error and exit.

Step-3 If the queue is not full, increment R to point next empty space.

Step-4 Add data element to the queue location where R is pointing.

Step 5 Increment F to 0 if F is -1.

4 Dequeue Algorithm.

Dequeue (Q, F, R)

1 if (F == -1)
"underflow"
return;

2 Y ← Q[F]

3 if F == R
then F ← R ← -1
else
F ← F + 1

4 return

DATE
PAGE No.

step-1 Check if queue is empty

step-2 If queue is empty, produce underflow and exit.

step-3 If the queue is not empty, access the data where front element is pointing.

Step-4 Increment the front pointer to next data element.

Step-5 If F==R, assign F and R to -1 (initial)

Step-6 Return.

DATE
PAGE No.

Q1 Sort the following and show each step
25, 56, 47, 35, 10, 90, 80, 31

(a) Bubble Sort

Pass 1 25 56 47 35 10 90 80 31

25 47 56 35 10 90 80 31

25 47 35 56 10 90 80 31

25 47 35 10 56 90 80 31

25 47 35 10 56 80 90 31

25 47 35 10 56 80 31 90

Pass 2 25 47 35 10 56 80 31 90

25 35 47 10 56 80 31 90

25 35 10 47 56 80 31 90

25 35 10 47 56 31 80 90

Pass 3 25 35 10 47 56 31 80 90

25 10 35 47 56 31 80 90

25 10 35 47 31 56 80 90

Pass 4 25 10 35 47 31 56 80 90

10 25 35 47 31 56 80 90

10 25 35 31 47 56 80 90

Pass 5 10 25 35 31 47 56 80 90

10 25 31 35 47 56 80 90

Sorted Array :-
10, 25, 31, 35, 47, 56, 80, 90

(b) Selection sort

25 56 47 35 10 90 80 31

Step-1 10 56 47 35 25 90 80 31

Step-2 10 25 47 35 56 90 80 31

Step-3 10 25 31 35 56 90 80 47

Step-4 10 25 31 35 47 90 80 56

Step-5 10 25 31 35 47 56 80 90

Sorted Array:-
10, 25, 31, 35, 47, 56, 80, 90

(c) Insertion sort

25 56 47 35 10 90 80 31

Step 1 25 56 47 35 10 90 80 31

Step 2 25 47 56 35 10 90 80 31

Step 3 25 47 35 56 10 90 80 31

25 35 47 56 10 90 80 31

DATE
PAGE NO

Step 4 25 35 47 10 56 90 80 31

25 35 10 47 56 90 80 31

25 10 35 47 56 90 80 31

10 25 35 47 56 90 80 31

Step 5 10 25 35 47 56 90 80 31

Step 6 10 25 35 47 56 80 90 31

Step 7 10 25 35 47 56 80 31 90

10 25 35 47 56 31 80 90

10 25 35 47 31 56 80 90

10 25 35 31 47 56 80 90

10 25 31 35 47 56 80 90

Sorted Array:-
10, 25, 31, 35, 47, 56, 80, 90

(d) Quick Sort

Step-1 25 56 47 35 10 90 80 31
Pivot P Q q

Step-2 25 10 47 35 56 90 80 31
Pivot Q P q
Left sub Array Right sub Array

Step-3 [10] 25 [47 35 56 90 80 31]
Pivot P q

Step-4 [10] 25 [47 35 31 90 80 56]
Pivot Q P q

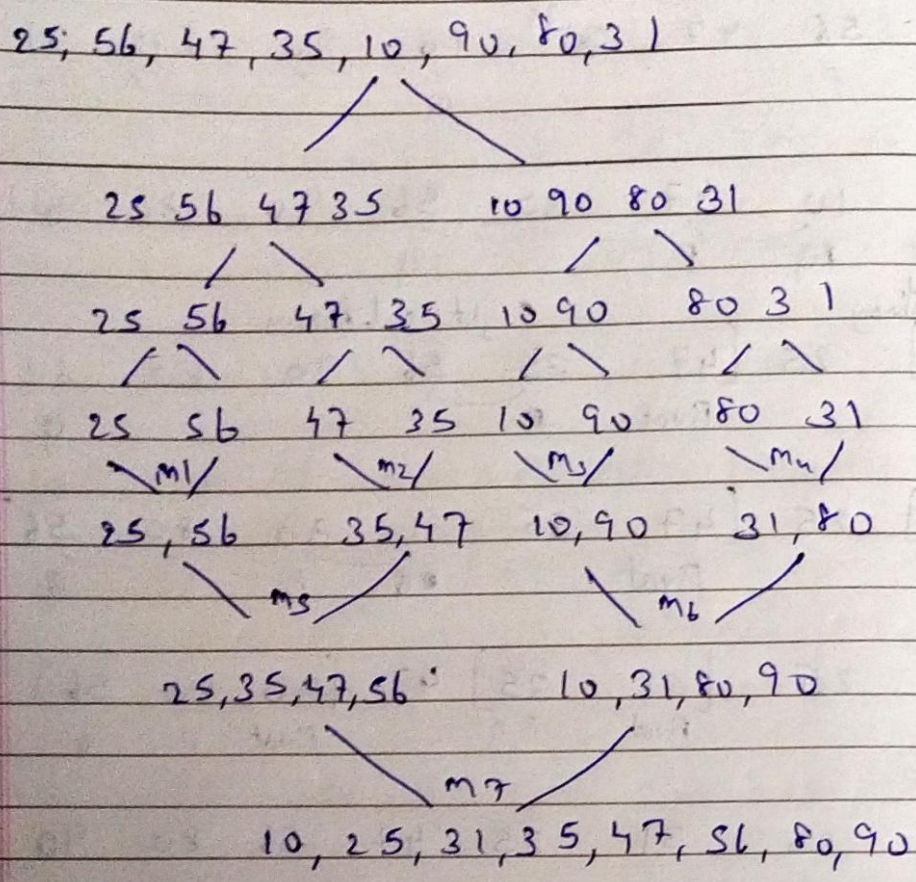
Step-5 10 25 [31 35] 47 [90 80 56]
Pivot P Q Pivot P Q

Step-6 10 25 31 35 47 56 80 90

Sorted Array :-
10, 25, 31, 35, 47, 56, 80, 90

DATE
PAGE No.

(e) merge sort



Sorted Array:-
10, 25, 31, 35, 47, 56, 80, 90

DATE
PAGE No

(f) Radix Sort

Pass-1 Input Array

25	56	47	35	10	90	80	31
0	1	2	3	4	5	6	7

Count Array

3	1	0	0	0	2	1	1	0	0
0	1	2	3	4	5	6	7	8	9

-1	2	3	3	3	3	5	6	7	7
0	1	2	3	4	5	6	7	8	9

Auxiliary Array

10	90	80	31	25	35	56	47
0	1	2	3	4	5	6	7

Pass-2 Input Array

10	90	80	31	25	35	56	47
0	1	2	3	4	5	6	7

Count Array

0	1	1	2	1	1	0	0	1	1
0	1	2	3	4	5	6	7	8	9

-1	0	0	1	2	3	4	5	5	6
0	1	2	3	4	5	6	7	8	9

Auxiliary Array

10	25	31	35	47	56	80	90
0	1	2	3	4	5	6	7

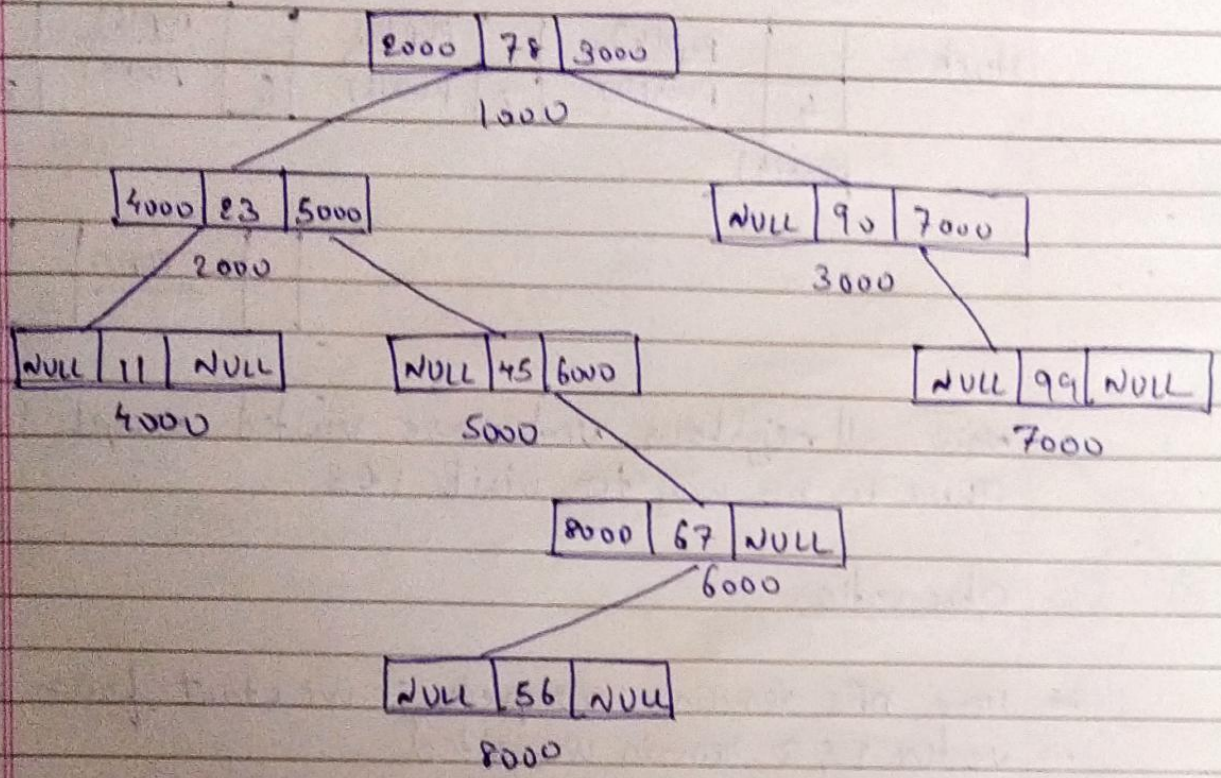
Pass-3 Input Array

10	25	31	35	47	56	80	90
0	1	2	3	4	5	6	7

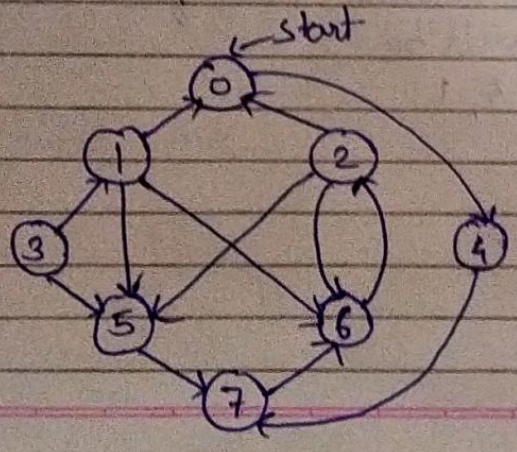
Sorted Array:- 10, 25, 31, 35, 47, 56, 80, 90

10) BST

78, 23, 11, 45, 67, 90, 99, 56



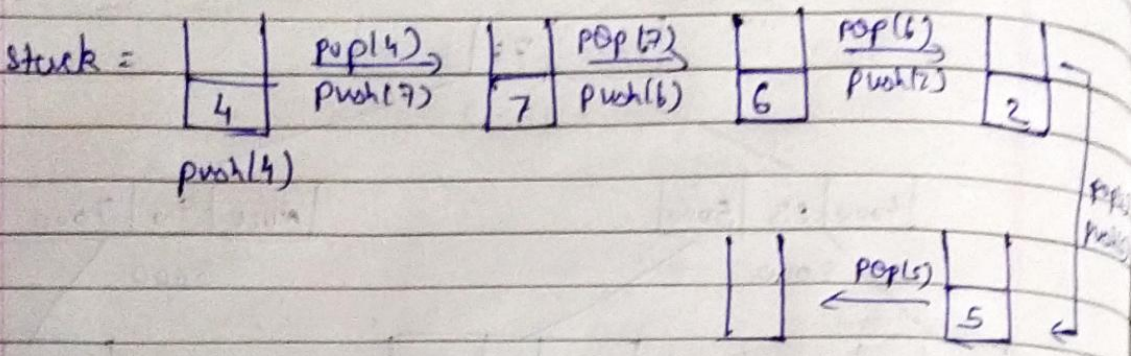
11) DFS sequence (start vertex is 0)



Vertex	Visited
0	✓
1	×
2	✓
3	×
4	✓
5	✓
6	✓
7	✓

DFS uses stack data structure.

sequence → 0 4 7 6 2 5



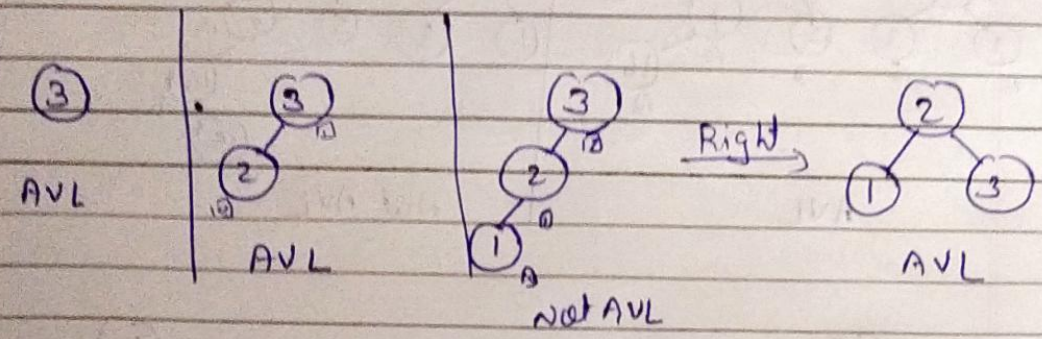
Now, all neighbour vertex are visited except 1 & 3
There is no way to visit 1 & 3.

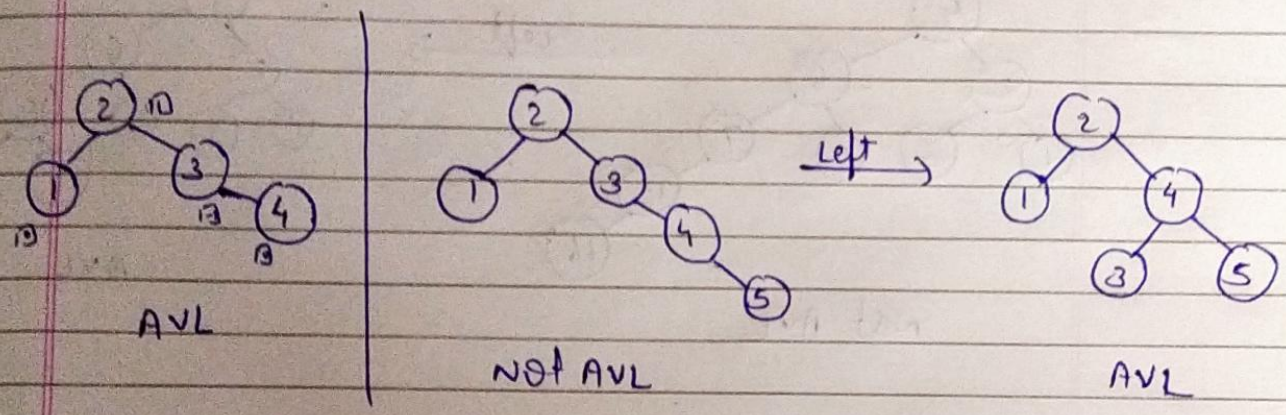
Observation

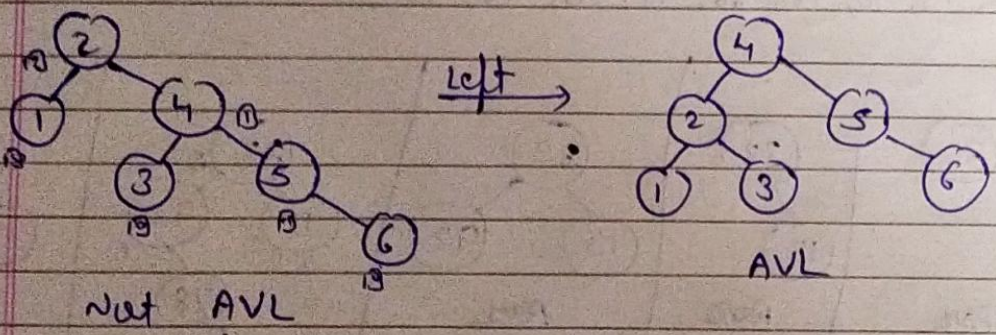
→ Here DFS remains incomplete if we start from vertex 0
→ vertex 1 & 3 remain unvisited
→ In this graph, there is loop of (5 → 7 → 6 → 2 → 0 → 4 → 7) so, there is no way to visit 1 & 3 if we start from vertex 0.
→ For complete DFS, we should start from vertex 3 then DFS sequence will be
3, 5, 7, 6, 2, 0, 4, 1

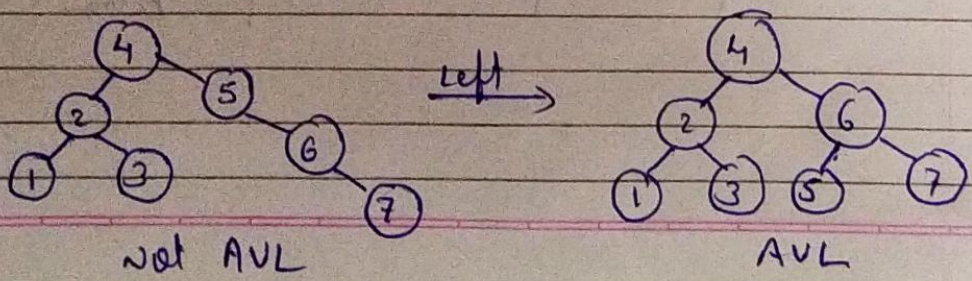
12) AVL Tree

3 2 1 4 5 6 7 16 15

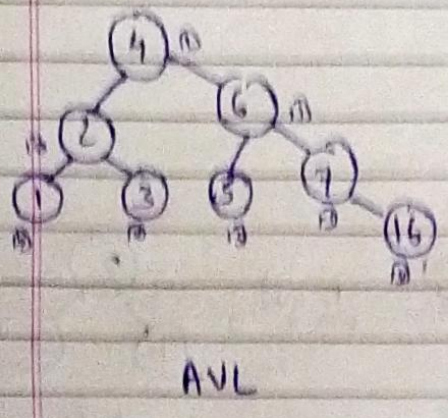




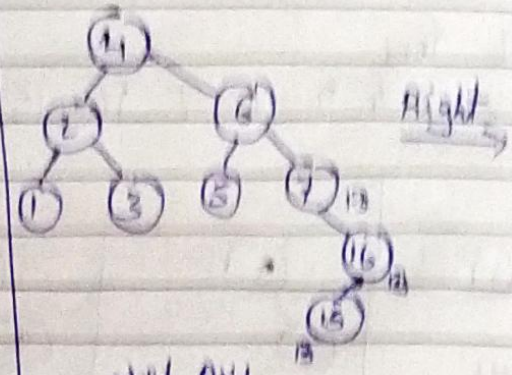




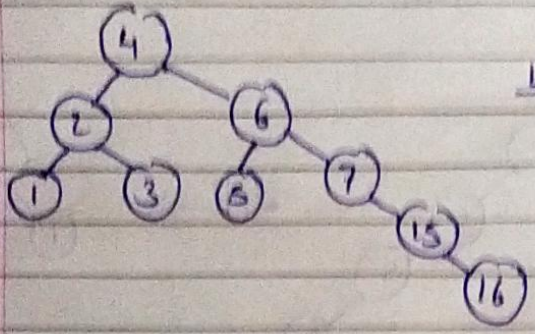
19CE138



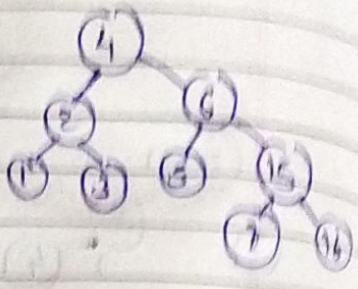
AVL



NOT AVL

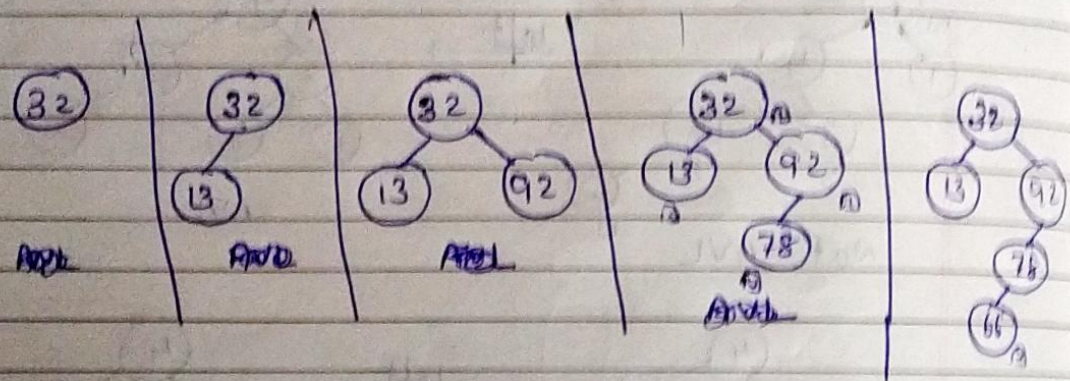


NOT AVL



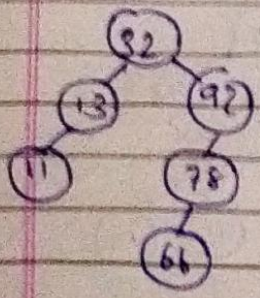
AVL

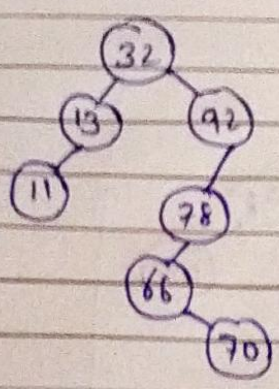
13) 32, 13, 92, 78, 66, 11, 70, 98, 108, 17, 45, 12

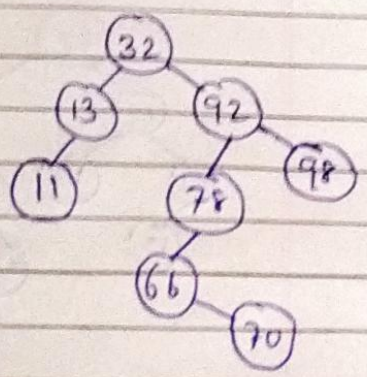


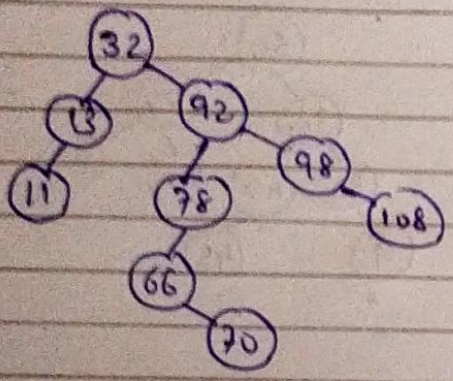
19CE138

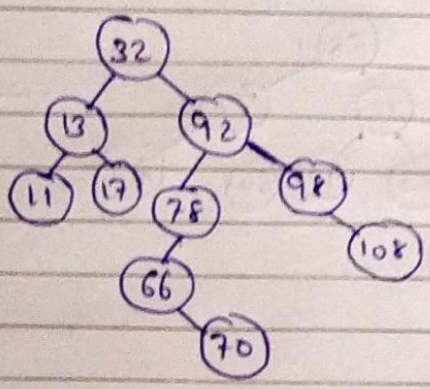
DATE
PAGE No.

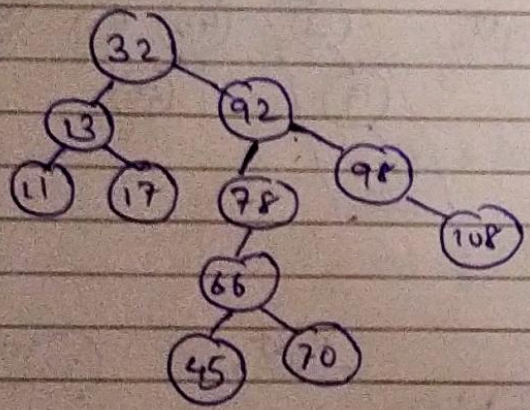


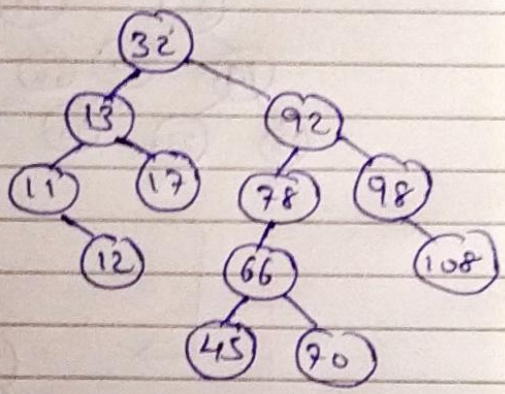




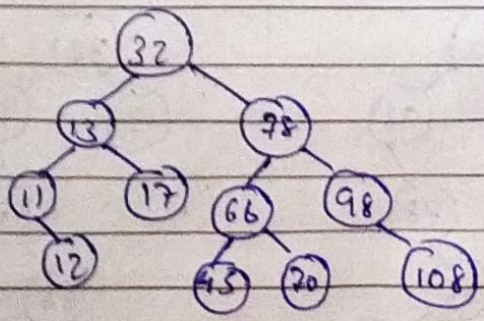


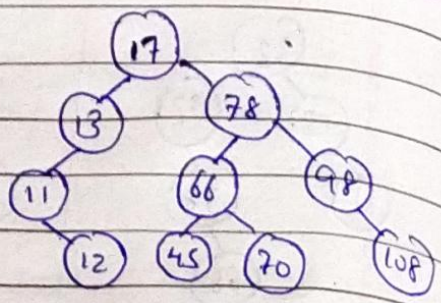




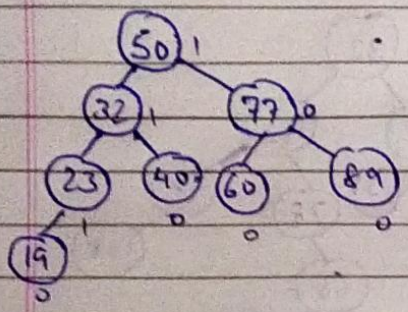


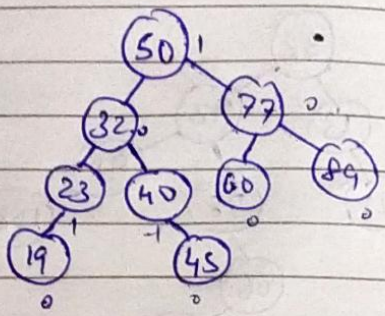
19CE138

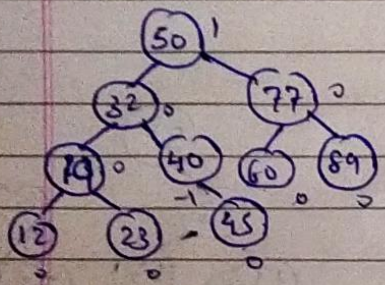




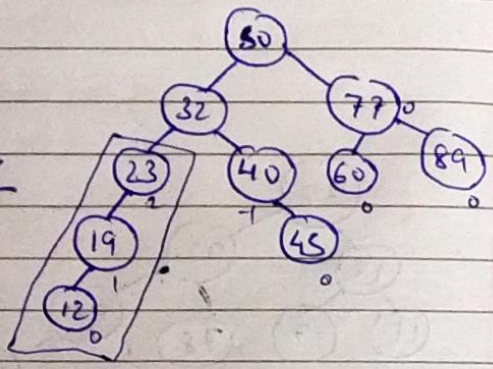
14)

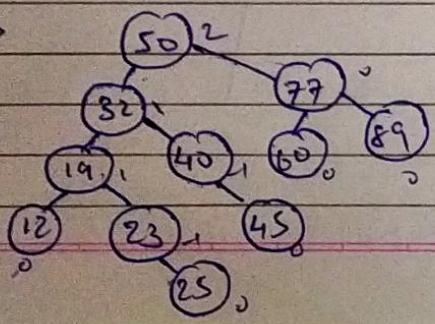




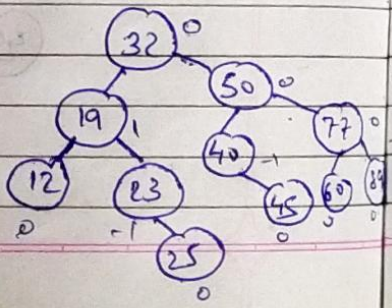


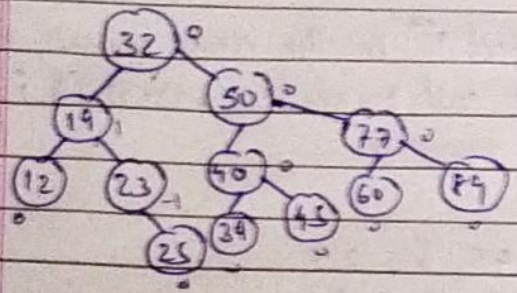
Right





Right





15) Suppose you are given an implementation of a queue of integers. The operations that can be performed on the queue are:

(i) isEmpty(Q): returns true if the queue is empty, false otherwise

(ii) delete(Q): delete the element at the front of the queue and returns its value.

(iii) insert(Q, i): inserts the integer i at the rear of the queue.

Consider the following function:

```
void f(queue Q)
{
    int i;
    if(!isEmpty(Q))
    {
        i = delete(Q);
        f(Q);
        insert(Q,i);
    }
}
```

what operation is performed by the above function f?

Ans Reverses the order of the elements in the queue Q.

As, it is recursive call, and removing from front while inserting from end, that means last element will be deleted

at last and will be inserted 1st in the new queue. And like it will continue till first call executes insert (Q, i) function. So, the queue will be in reverse.

16) Data: 50, 47, 19, 21, 11, 45, 6, 25
Hash f^n H(x) = (x mod 5) + 3

Index	Data
0	45
1	6
2	25
3	50
4	21
5	47
6	11
7	19

Hashing: 1) (50 mod 5) + 3 = 3
2) (47 mod 5) + 3 = 5
3) (19 mod 5) + 3 = 7
4) (21 mod 5) + 3 = 4
5) (11 mod 5) + 3 = 4 collision
(4+0) mod 8 = 4 collision
(4+1) mod 8 = 4 collision
(4+2) mod 8 = 6 ✓

6) (45 mod 5) + 3 = 2 collision
(3+0) mod 8 = 3 collision
(3+1) mod 8 = 4 collision
(3+2) mod 8 = 5 collision

(3+3) mod 8 = 6 collision
(3+4) mod 8 = 7 collision
(3+5) mod 8 = 0 ✓

7) (6 mod 5) + 3 = 4 collision
(4+0) mod 8 = 4 collision
(4+1) mod 8 = 5 collision
(4+2) mod 8 = 6 collision
(4+3) mod 8 = 7 collision
(4+4) mod 8 = 0 collision
(4+5) mod 8 = 1 ✓

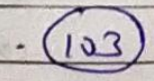
8) (25 mod 5) + 3 = 3 collision
(3+0) mod 8 = 4 collision
(3+1) mod 8 = 5 collision
(3+2) mod 8 = 6 collision
(3+3) mod 8 = 7 collision
(3+4) mod 8 = 8 collision
(3+5) mod 8 = 0 collision
(3+6) mod 8 = 1 collision
(3+7) mod 8 = 2 ✓

So, final array using hashing is

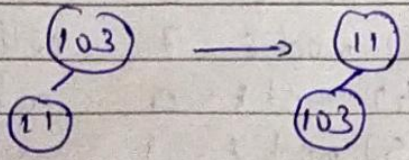
45	6	25	50	21	47	11	19
0	1	2	3	4	5	6	7

17) [103, 11, 101, 110, 111, 119, 19, 91]

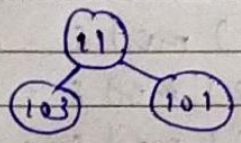
Step 1) Insert 103



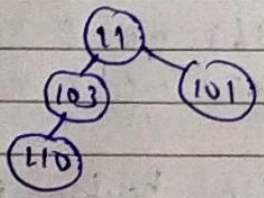
2) Insert 11 (11 < 103) so swap 11 & 103



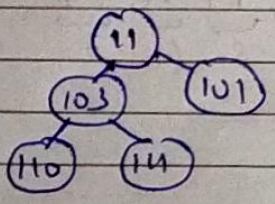
3) Insert 101



4) Insert 110 at bottom (start from left)

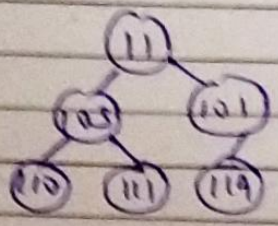


5) Insert 111 (at bottom)

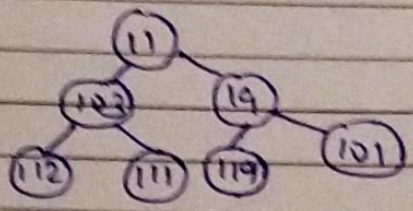


DATE
PAGE No.

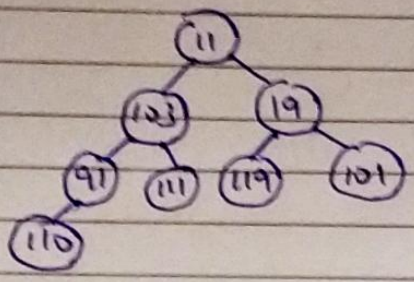
1) Insert 119



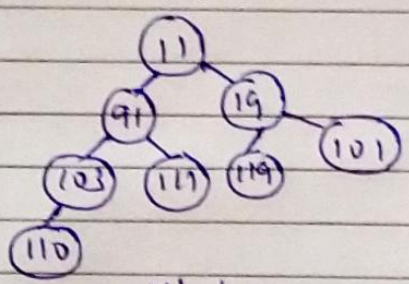
2) Insert 19 (19<101) so swap 19 & 101



3) Insert 91 (91<110) so swap 91 & 110



4) (91<103) so swap 91 & 103



so, min. heap will be → 11, 91, 19, 103, 111, 119, 101, 110