Winter: 2019

Q.1

(a) Discuss Various types of Data structure.

→ The data type represent type of data.
→ The data type can be numeric or character type.
→ It is categorized into two group Primitive and non-primitive.
→ There are basic four types of data types
(1) Integer data type
(2) Float data type
(3) Character data type
(4) Double data type

→(1) Integer data type :

Format = %d
Memory = 2 byte

It is used to have integer value in variable. That value should not be contain any fractional data.

Ex. int a ;
a = 5 ; [Allowed]
a = 0.5 ; [Not allowed]

→(2) Float data type : This is used to store the real value in the variable.

Format = %f
Memory = 2 byte

The value may contain some fractional data.

Ex. float a;
a = 5; [Allowed but it will be interpreted as]
a = 5.0;

a = 3.79; [Perfectly Allowed]

→ (3) Character data type:

Formate = %c
Memory = 1 byte

It is used to contain some text or alphabatical information in a variable. Character is always written in single quotes.

Ex. Char Status;
Status = 'T';

→ (4) Double data type:

Format = %.If
Memory = 8 byte

It is used to store numerical information in variable. It is same as float data type but capacity is larger than float data type.

Ex. double val;
val = 3.45;

String
→ ~~Primitive~~ data type

(b) What is hash function. Give Example

→ Hashing : It is a technique to convert a range of key value into a rang of index of array.

→ Hash function is a function which is used to put the data in the hash table. Hence one can use the same hash function to retrive the data from the hash table. Thus has function is used to implement the has table.

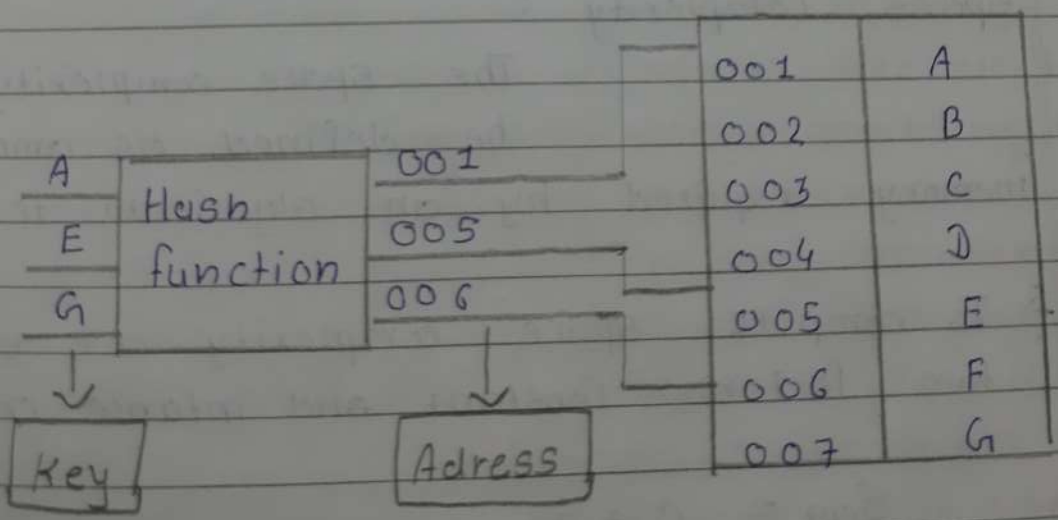
→ The integer returned by has function is called hash key.

→ Hashing involves less key comparision and searching can perfomed in constant time.

→ The goal of hash search is to find the largest data in only one test.

O(1) { complexity }

Ex.



→ Each key has only one adress.

(c) What is time and space analysis? State and explain time analysis for linear and binary search method.

→ Time Complexity :

Time complexity is the ammount of time taken by the program for execution.

As it is difficult to measure the time complexity in terms of clock units, we will measure the time complexity using frequency count.

Frequency count :

The efficiency of program is measured by inserting a counter in the algorithm in order to count the number of times the basic operation is executed. This is a method of measuring time complexity of program.

→ Space Complexity :

The space complexity can be defined as ammount of memory required by an algorithm to run.

→ To compute space complexity we used two factors : Constant and intance characteristic

$$S(P) = C + S_P$$

→ Where, C is constant and it denotes space of input and output.
→ This space is amount of space taken by instructions variable and identifires.
→ Sp is a space dependent upon instance characterstics.

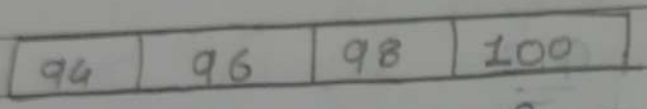
Q. 2

(a) Compare array and link list.

Array	Link list
→ It is a collection of elements of similar data type.	A link list is collection of an object known as node
→ It is store in a contiguous memory location.	It is store anywhere in the memory or randomly stored
→ Array works with a static memory.	It is work with dynamic memory.
→ Memory size is fixed can not change at run time.	Memory size can be change according to our requirments.
→ Array elements are independent of each other.	It is dependent on each other.
→ Array takes more time while perfoming any operation	It takes less time while perfoming any operation
→ Array used in binary and linear search tree.	It is used in linear search tree.
→ Less memory required.	More memory required.

(b) State disadvantages of Simple queue. How to overcome it?

→ Queue is a linear data structure in which insertion of element in the queue is from one end called rear and deletion of element from the other end called front.



→ Simple queue is linear in nature having boundary between front and rear.

→ In simple queue there can be wastage of space as front and rear are away from each other.

→ Disadvantages : The simple queue is linear in nature. Hence after deletion of some element the corresponding space can go waste.
On the other hand in circular queue the intermediate space that is left in between can be utilized.

→ It is possible to move from rear end to front end easily.

(c)

(c) INSERT and DELETION Algorithm in circular Queue.

=> INSERT (F, R, Q, N, Y)

- Given F and R are pointors of the front and rear element of circular queue.
Circular queue Q containing N elemnts.
The Procedure of insert Y at rear end of circular queue.

[1] Reset rear pointer
if, R = N
Then R ← 1
Else R ← R+1

[2] Overflow
if, F = R
Then, write ('overflow')
Return

[3] Insert Element
Q[R] ← Y

[4] Is front pointor is properly set?
if, F = 0
Then F ← 1
Return

→ DELETE (F, R, Q, N)

- Given F and R are pointers of front and rear element of circular queue. Circular queue containing N elements. This function deleted an element from front end queue.

[1] Underflow ?
if, F = 0
Then write ['Underflow']
Return [0]

[2] Delete element
Y ← Q[F]

[3] Queue empty ?
if, F = R
Then F ← R ← 0
Return (Y)

[4] Incremeant front pointor
if, F = N
Then, F ← 1
Else F ← F + 1
Return (Y)

Q.3

(a) Discuss height balance tree

→ An empty tree is height balance tree if T is a non-empty binary tree will T_L and T_R as its left and Right sub-trees.

→ The tree is height balance tree if and if only
(i) T_L and T_R are height balanced
(ii) $|h_L - h_R| \leq 1$ (Balance factor)

Where,

T_L = Left Subtree
T_R = Right Subtree
h_L = Height of left subtree
h_R = Height of Right subtree

→ BF(T) = -1, 0, 1

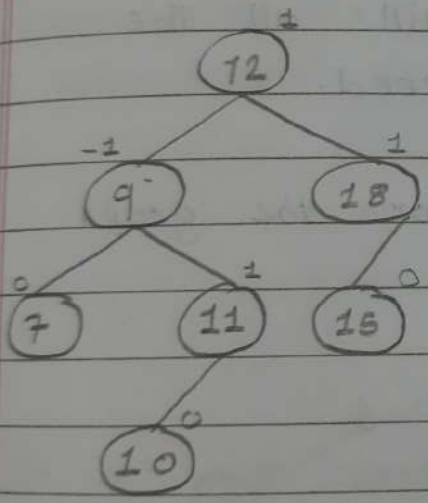
→ There are four types of Rotation
(1) Insertion of node into left subtree of left child (LL).
(2) Insertion of node into right subtree of left child (LR)
(3) Insertion of node into left subtree of right child (RL)
(4) Insertion of node into right subtree of right child (RR)

→ The AVL tree follows the property of binary search tree. Infact AVL trees are basically binary search tree with balance factor as -1, 0 or +1.

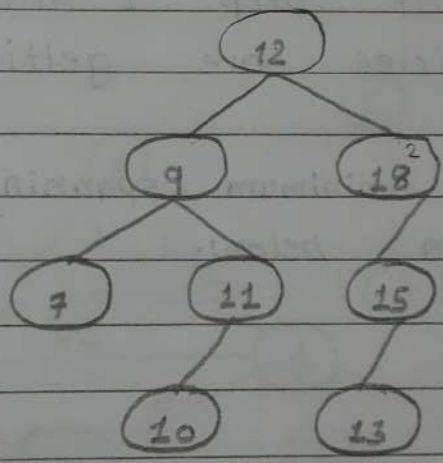
-> After insertion of any node in an AVL tree if the balance factor of any node becomes other than -1, 0 or 1 then it is said that AVL property is violated. Then we have to restore the destroyed balance condition. Blance factor is denoted at right top corner inside the node.

-> After an insertion of node if balance condition get destroyed, then the node on path need to be readjusted. That means only affected subtree is to be rebalanced.

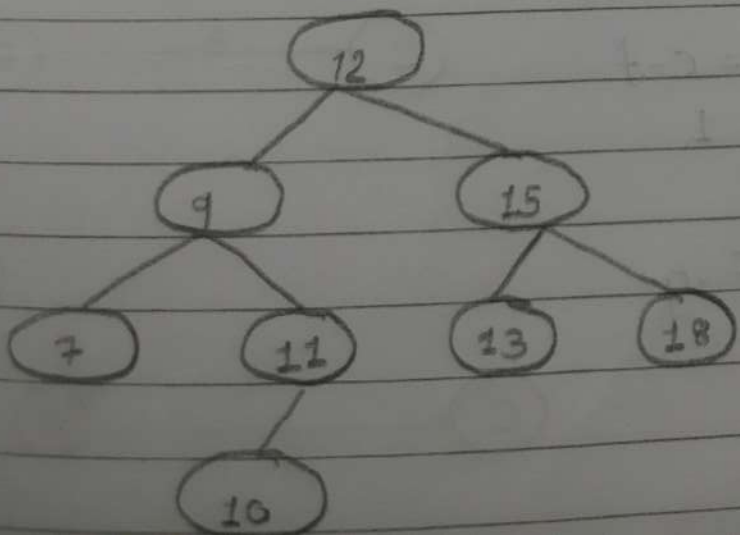
-> Original AVL



-> Insert : 13



-> Restoring AVL property and Adjusting node



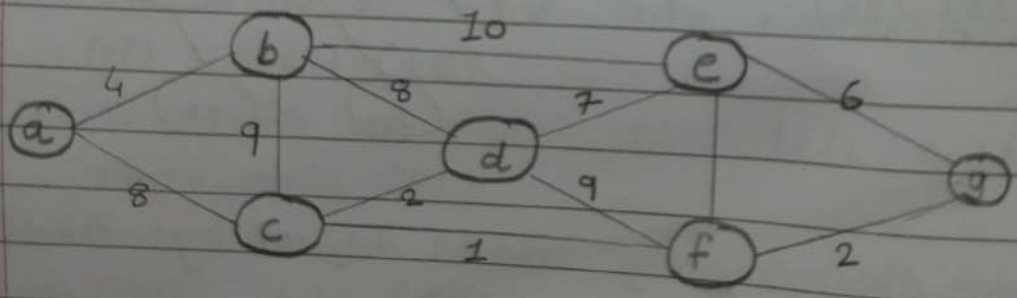
(b) Discuss minimum spanning tree.

→ The minimum spanning tree of a weighted connected graph G is called minimum spanning tree if its ~~weig~~ weight is minimum.

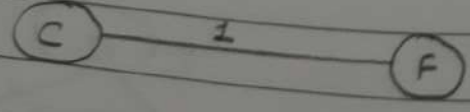
* Prim's Algorithm

Step 1: Select the pair with minimum weight
Step 2: Select the adjacent ~~and~~ vertex and select the minimum weighted edge using this adjacent vertex. The selected adjacent vertex should not form the circuit.
Step 3: Repeat Step : 1 and 2 untill all the verticies are getting covered.

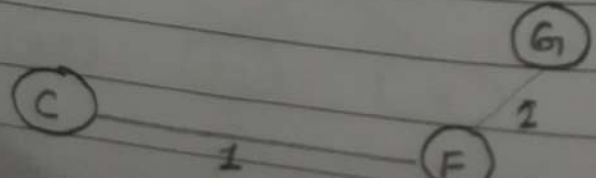
Ex. Find minimum spanning tree for the graph shown below.



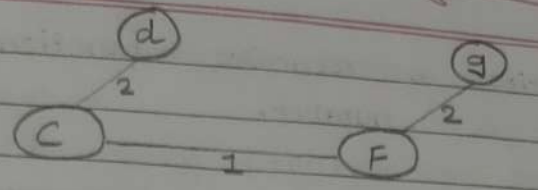
→(1) Select an edge = c-f
path length = 1



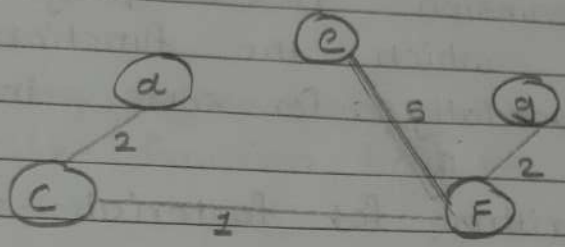
(1) Select an edge = f-g
path length = 3



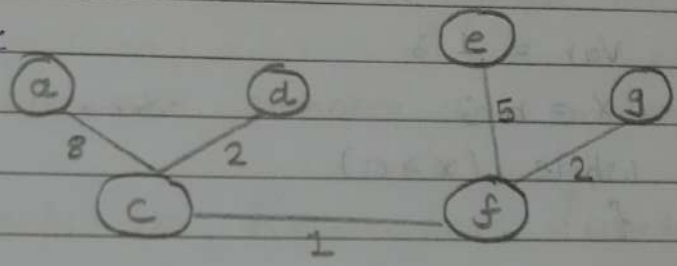
(3) Select an edge = c-d
path length = 5



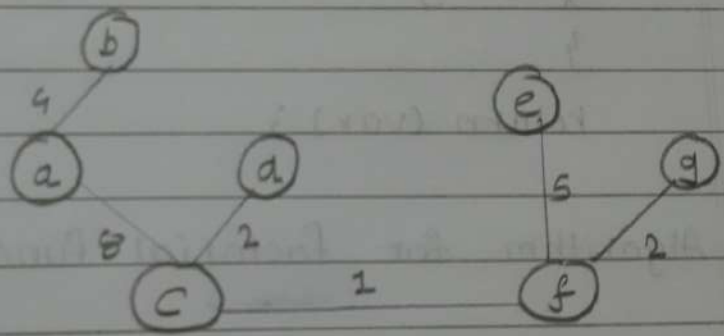
(4) Select an edge = f-e
path length = 10



(5) Select an edge = a-c
path length = 18



(6) Select an edge = a-b
path length = 22



→ As all the vertices are visited, we obtain minimum spanning tree with path length 22.

(c) Write a recursive function to compute factorial of a number.

→ Recursion is a programming technique in which the function call itself repeadatly for some input.

→ Algorithm for factorial function using itreative:

```
Var = 1 ;
x = n ;
while (x > 0)
{
Var = Var * x
x -- ;
}
return (var) ;
```

→ Algorithm for factorial function using Recursive

```
If (n == 0)
fact = 1 ;
else
{
x = x - 1 ;
y = value of x ;
fact = n * y ;
}
```

→ Advantage : (1) Recursive method being compactness in program.
(2) There is no need of using programing constructs such as for, while, do-while.

→ Disadvantage : (1) Memory utilization is more in recursive function because all pending operation must be preserved.
(2) Recursive method less efficient than iterative method.
(3) Recursive method are complex to implement.

→ Usage of Stack in Recursive function

```
int fact (int num)
{
 int a, b ;
 if (num == 0)
    return 1;
    else
    {
     a = num - 1;
     b = fact(a);
     f = a * b;
     return f;
    }
}
```

Q. 3

(b) Write an algorithm insert a node in circular link list.

→ Algorithm :

Q INSERT (QUEUE [MAXSIZE], Iteam)

Step 1 : If (front == (Rear %. 1) %. maxsize)]
write queue is overflow and exite.

Else
Rear = [(Rear %. 1) %. maxsize]

[Assign value] QUEue [Rear] = Value
[End if]

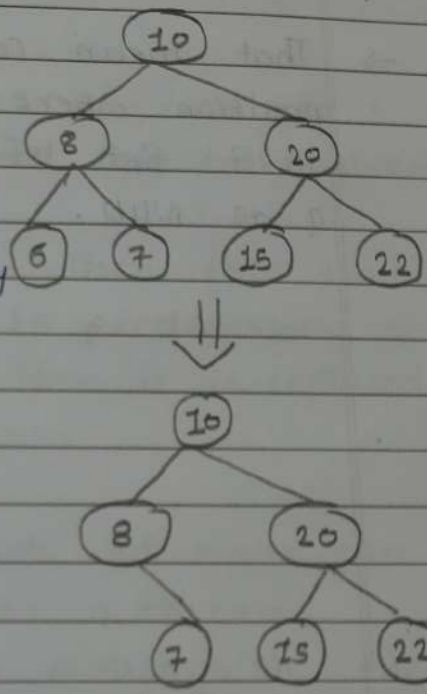
step 2 : Exit

(c) Write an algorithm for DELETE operation in a Binary search tree.

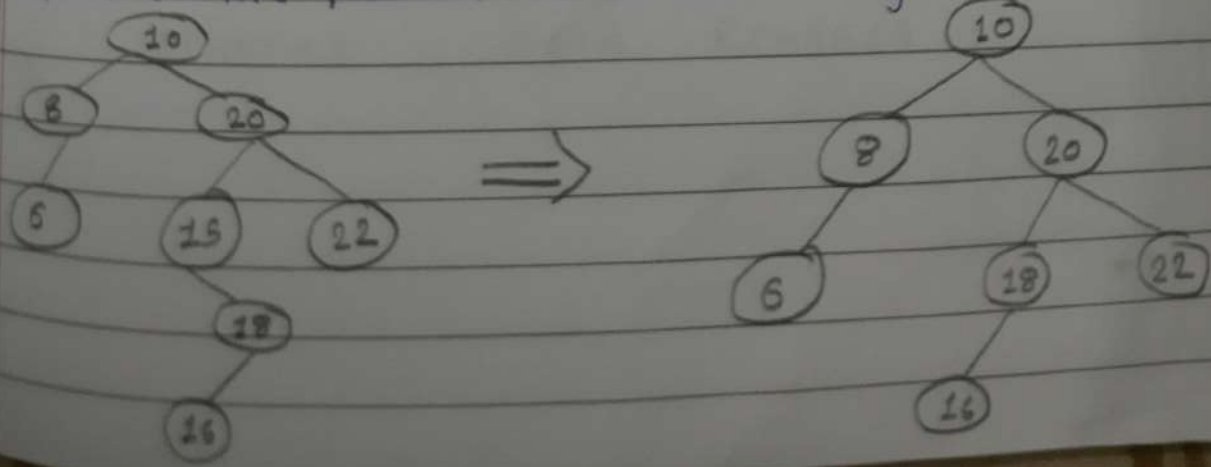
→ There are 3 cases
(1) Deletion of leaf node
(2) Deletion of node having one child
(3) Deletion of node having two child

→(1) Deletion of leaf node.

- This is simplest deletion, in which we set the left or right pointer of parent node as NULL.
- We want to delete the node having value 6 then we will set left pointer of its parent node as NULL. That is left pointer of node having value 8 is set to be NULL

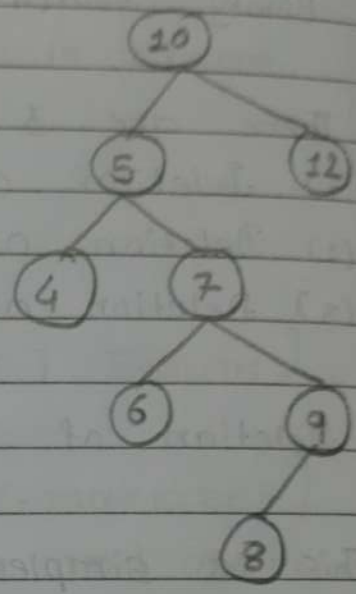


(2) Deletion of node having one child
- If we want to delete node 15, then we will simply copy node 18 at place of 15 and then set the node free. The inorder ~~is always~~ successor is always copied at the position of node being deleted.

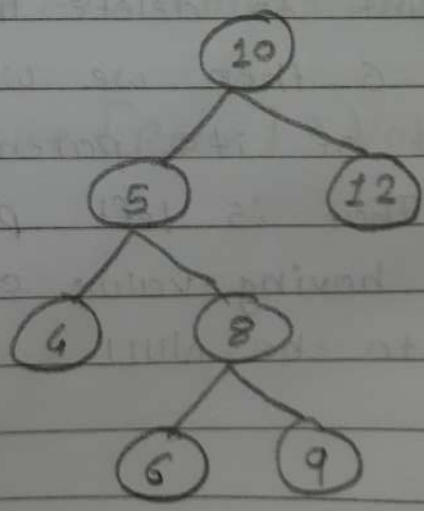


(3) The node having two children

-> We want to delete node having value 7. We will then find out the inorder succesor of node 7. The inorder successor will be simply copied at location of node 7.



-> That mean copy 8 at the position where value of node is 7. Set left pointer of 9 as NULL.



Q.4

(a) Discuss Threaded Binary Tree.

→ The wasted NULL links in the binary tree storage representation can be replaced by threads.

→ A binary tree is threaded according to particular traversal order.

→ Threads for the inorder traversal of tree are pointers to its higher nodes, for this traversal order.

→ If left link of node P is NULL- then this link is replaced by the address of its predecessor. If Right link of node P is NULL. then it is replaced by the address of its successor.

→ Because the left and Right link of node can denote either structual link or a thread, we must somehow be able to distinguish them.

→ Method 1: Represent thread with -Ve address
Method 2: To have seprate boolean flag for each of left or right pointers.

LTHREAD DATA RTHREAD

- LTHREAD = TRUE = Denotes leaft thread link
- LTHREAD = FALSE = Denotes leaft structual link
- RTHREAD = TRUE = Denotes Right thread link
- RTHREAD = FALSE = Denotes Right structual link.

→ Advantages :

Inorder traversal are faster than unthread version.

- Effectively determines the proccessor and successor for inorder traversal.
- It is possible to genrate succesor or predecessor of any node without having over head.

→ Disadvantage :

Threaded tree are unable to share common subtree.

(b) Pre-Order Traversal for Binary Search tree.

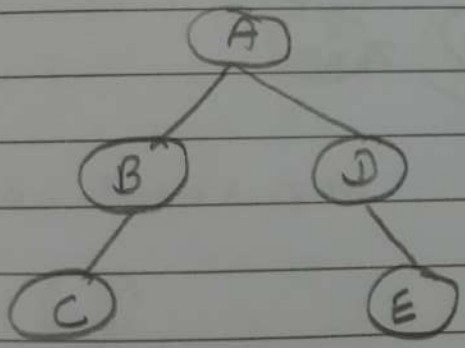
→ Definition : Traversing the tree means visit the each node exactly once.

→ There are 3 types of traversing
(1) PreOrder (2) InOrder (3) PostOrder

→ L - means move to the left child
R - means move to the Right child
D - means the ROOT / PARANT node.

* PreOrder :

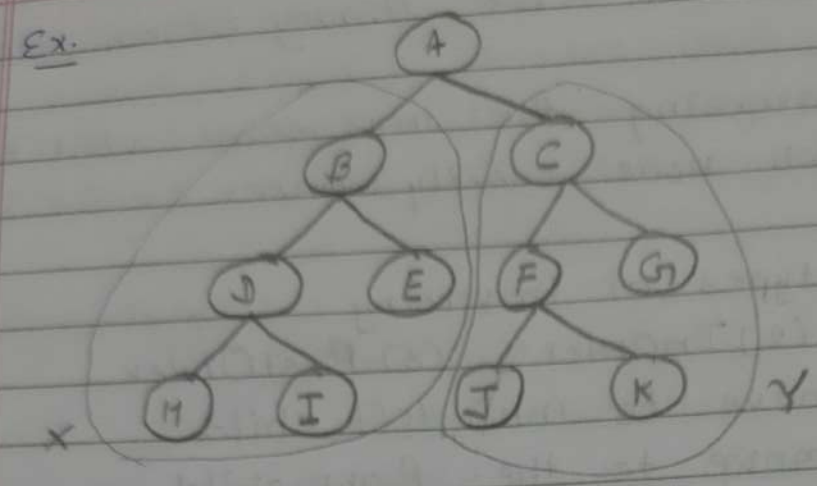
→ Step 1 : Traverse the root (D)
Step 2 : Traverse left sub tree (L)
Step 3 : Traverse Right sub tree (R)



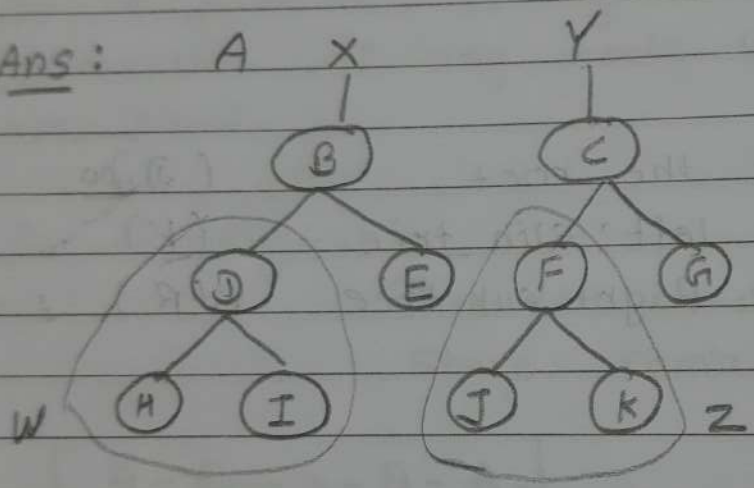
[A - B - C - D - E]

→ A - B - C - D - E is the pre order traversal
→ First we go to root the towards the left B then C and then towards the right subtree D then Finally E.
→ The basic principal of is to preorder traversal is Root then left sub tree. then Right subtree.

→ Ex.

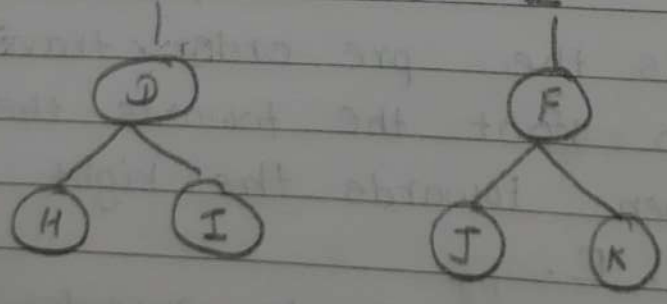


→ Ans: A X Y



A B W E C Z G

→ A B W E C Z G

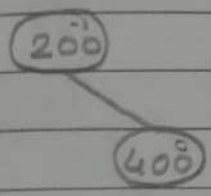


→ Ans: A B D H I E C F J K G

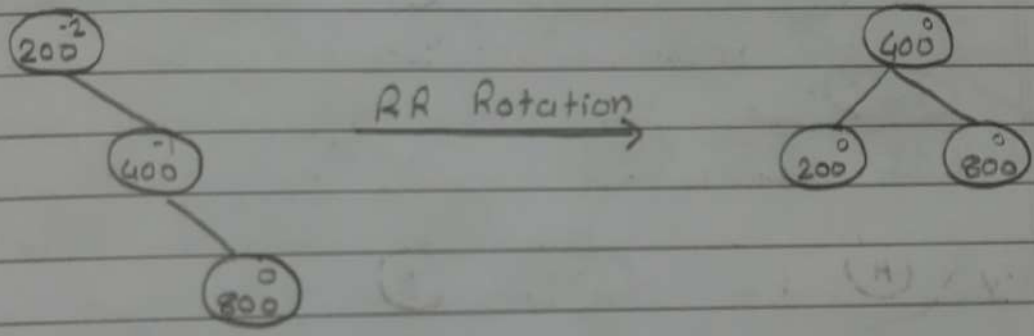
(c) Create AVL tree for the following sequence of numbers. 200, 400, 800, 900, 850, 700, 950, 100, 150

-> Step 1: Insert 200

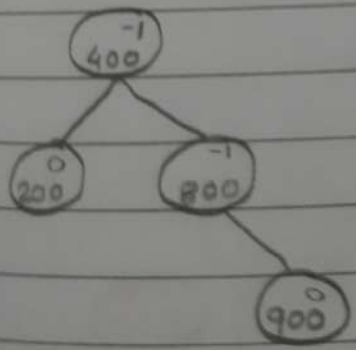
-> Step 2: Insert 400



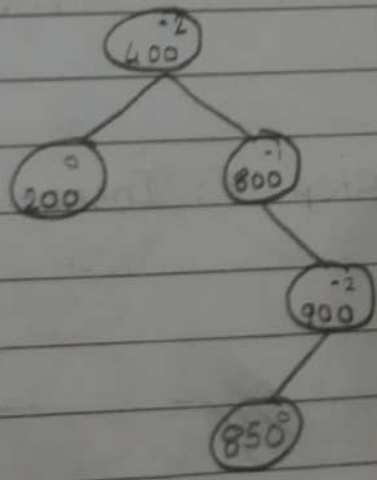
-> Step 3: Insert 800

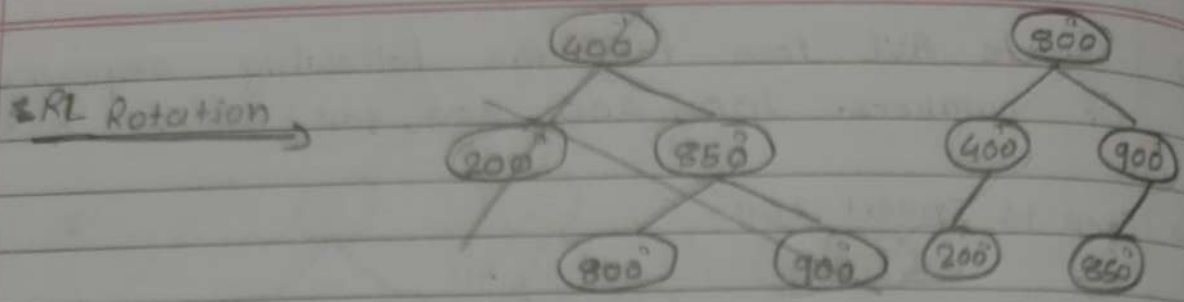


-> Step 4: Insert 900

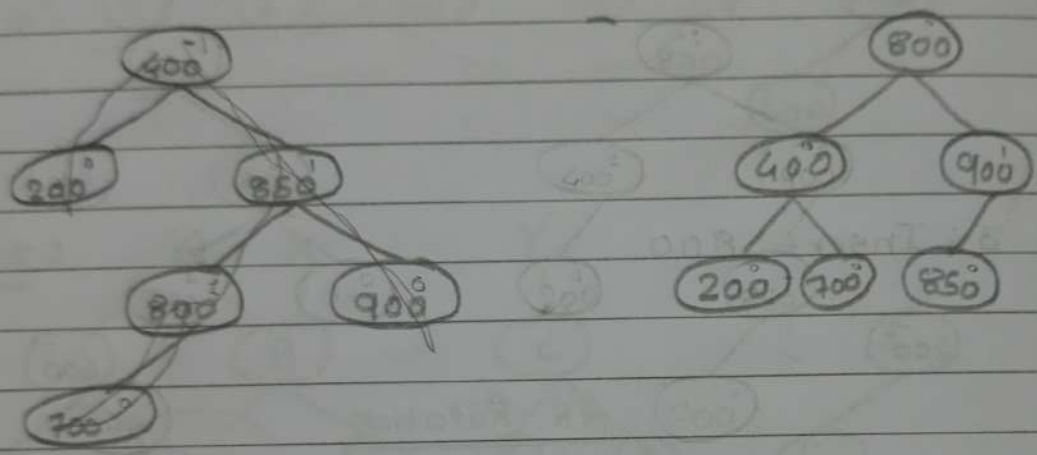


-> Step 5: Insert 850

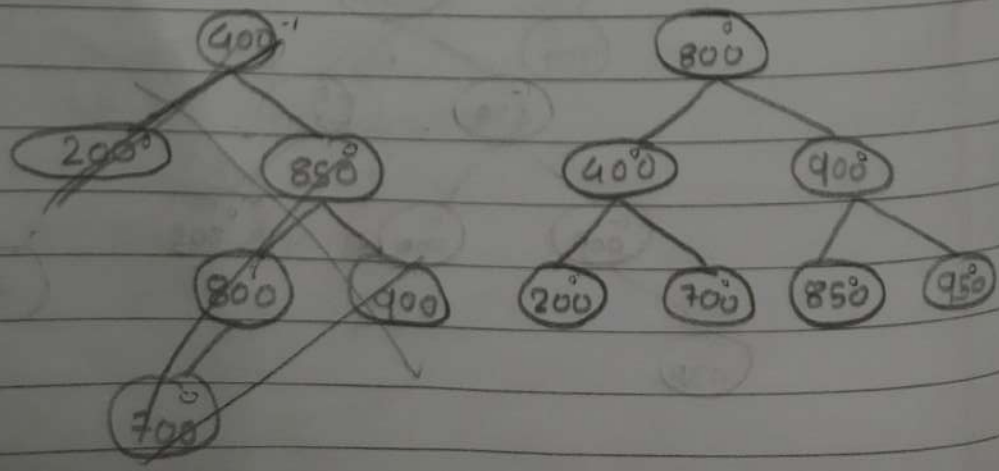




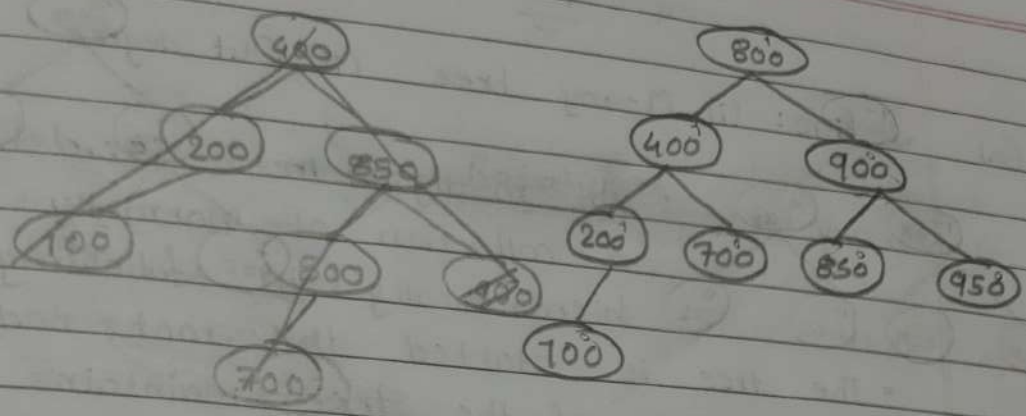
→ Step 6: Insert 700



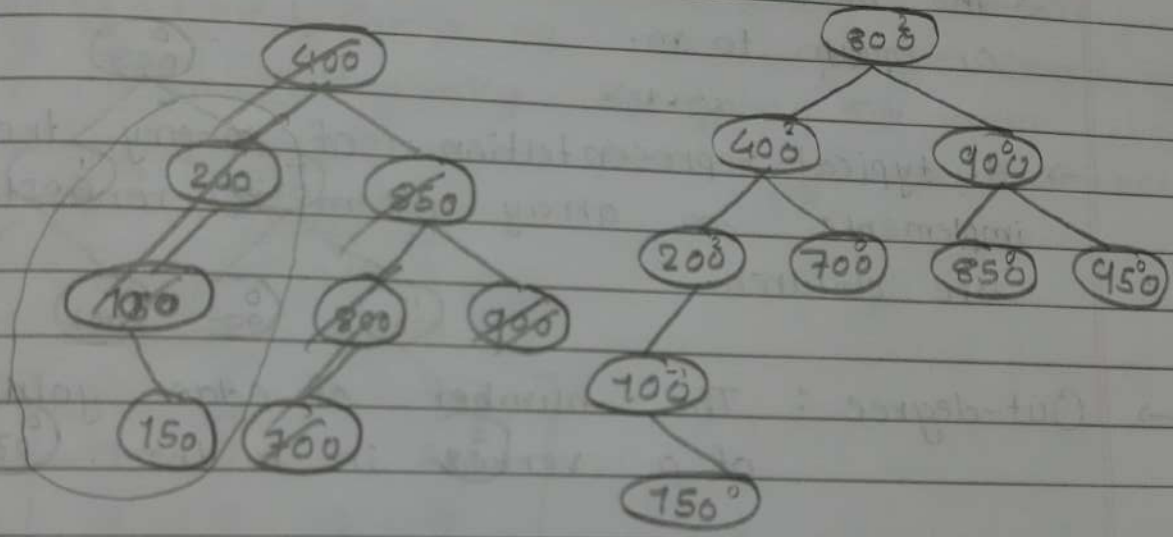
→ Step 7: Insert 950



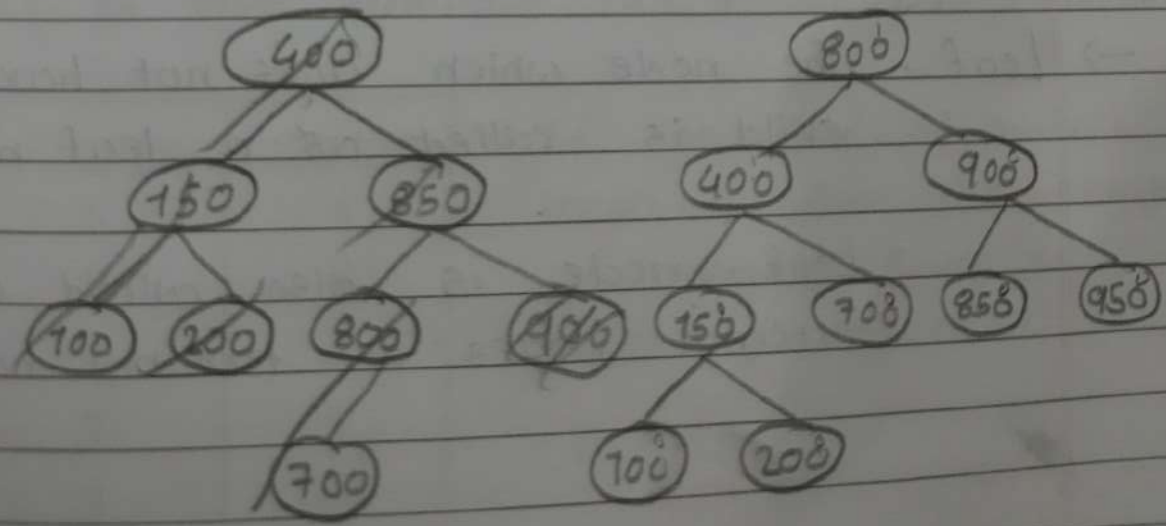
-> Step 8 : Insert 100



-> Step 9 : Insert 150



LR Rotation



Q.4

(a) Define: (i) M-ary tree (ii) Out degree (iii) Leaf

→ M-ary tree : An m-ary tree is defined as a collection of normally represented hierarchically in following manner.

- The tree is started the root node.
- Each node of the tree maintains a list of pointers to its child nodes.
- The number of child nodes is less than or equal to m.

→ A typical representation of m-ary tree implements an array of m reference to store children.

→ Out-degree : The number of edges going out of a vertex in a tree.

- Degree of node is the total number of children that node.

→ Leaf : The node which does not have any child is called as a leaf node.

- Leaf node is also called as external nodes or terminal nodes.

(b)

(b) State atleast one efficient representation of a sparse matrix.

→ A matrix is a two-dimensional data object made of (m) rows and (n) colums, therefore having total m x n values. If most of the elements of the matrix have 0 value, then it is called sparse matrix.

→ Storage: There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.

→ Computing-time: Computing time can be saved by logically designing a data structure traversing only non-zero element.

Ex —

	0	1	2	3
0	0	4	0	5
1	0	0	3	6
2	0	0	2	0
3	2	3	0	0
4	0	0	0	0

5x4

→ There are 13 zero elements and 7 non-zero elements.

Row	Column	Value
0	1	4
0	3	5
1	2	3
1	3	6
2	2	2
3	0	2

→ If the size of matrix increased then the wastage of memory space will also be increased.

(c) Discuss algorithm of BFS traversal for a graph.

-> BFS produce a spanning tree as a final result is a graph without loop.
-> We use queue data structure with maximum size of total number vertices in the graph to implement BFS of a graph.

-> Algorithm:

Step 1: Define a queue of size total no. of vertices in the graph.

Step 2: Select any vertex as starting point and traversal. Visit that vertex and insert into queue.

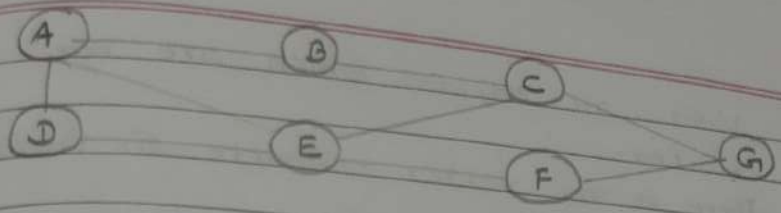
Step 3: Visit all the adjacent vertices of the vertex which is at front of the queue. which is not visited insert into the queue.

Step 4: When there is no vertex visited from the vertex at front then delete the vertex from the queue.

Step 5: Visit step 3 and 4 untill the queue becomes empty.

Step 6: When Queue becomes empty produce final spanning tree by removing unused edges from the graph.

Ex.



→ Step 1: Select A as a starting point.
Insert A into queue.

A						

→ Step 2: Visit all adj(A) which are not visited.
Insert D, E, B into the queue and delete A.

	D	E	B			

→ Step 3: Visit all adj(D) which are not visited.
There is no vertex delete D from the queue.

		E	B			

→ Step 4: Visit all adj(E) which are not visited.
Insert C, F into the queue and delete E.

			B	C	F	

→ Step 5: Visit all adj(B) which are not visited.
There is no vertex delete B from the queue.

				C	F	

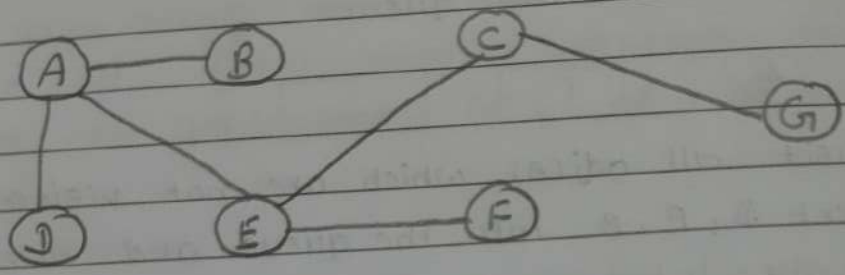
→ Step 6: Visit all adj(C) which are not visited.
Insert G into the Queue. and delete C.

					F	G

→ Step 7: Visit all adj(F) which are not visited.
There is no vertex delete F from the queue.

						G

→ Step 8 : Visit all adj(G) which are not visited.
There is no vertex . Delete G from queue



→ Final result of BFS spanning tree by removing unused edges.

(a)

→

Ex.

* Algorithm for bubble sort

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

Q.5

(a) Write a algorithm for bubble sort method.

→ Algorithm:

(1) Read the total number of element say n.
(2) Store the elements in the array.
(3) Set , i = 0
(4) Compare the adjacent element.
(5) Repeat step 4 for all n elements
(6) Increment the value of i by 1 and repeat step 4, 5 for i < n
(7) Print the sorted list of element.
(8) Stop.

Ex. Consider 5 unsorted elements.
45, -40, 190, 99, 11

0	45
1	-40
2	190
3	99
4	11

→ Pass 1 :

- Compare 45 and -40
  ∴ Interchange.

0	45	-40
1	-40	45
2	190	190
3	99	99
4	11	11

- Compare a[1] and a[2]
  45 < 90 ∴ No interchange

- Compare a[2] and a[3]
  190 > 99 ∴ Interchange

0	-40	-40
1	45	45
2	190	99
3	99	190
4	11	11

- Compare a[3] and a[4]
  190 > 11 ∴ Interchange.

0	-40
1	45
2	99
3	190
4	11

0	-40
1	45
2	99
3	11
4	190

→ Pass : 2

- Compare a[0] and a[1]
  -40 < 45 ∴ No Interchange

- Compare a[1] and a[2]
  45 < 99 ∴ No Interchange

- Compare a[2] and a[3]
  99 > 11 ∴ Interchange.

- Compare a[3] and a[4]
  99 < 190 ∴ No interchange.

0	-40
1	45
2	99
3	11
4	190

0	-40
1	45
2	11
3	99
4	190

→ Pass : 3

- Compare a[0] and a[1]
  -40 < 45 ∴ No Interchange

- Compare a[1] and a[2]
  45 > 11 ∴ Interchange

- Compare a[2] and a[3]
  45 < 99 ∴ No Interchange.

- Compare a[3] and a[4]
  99 < 190 ∴ No Interchange.

0	-40
1	45
2	11
3	99
4	190

0	-40
1	11
2	45
3	99
4	190

(b) Merge Sort

-> The merge sort is a sorting algorithm that divide and conquer strategy. In this method division is dynamically carried out.

-> Steps:

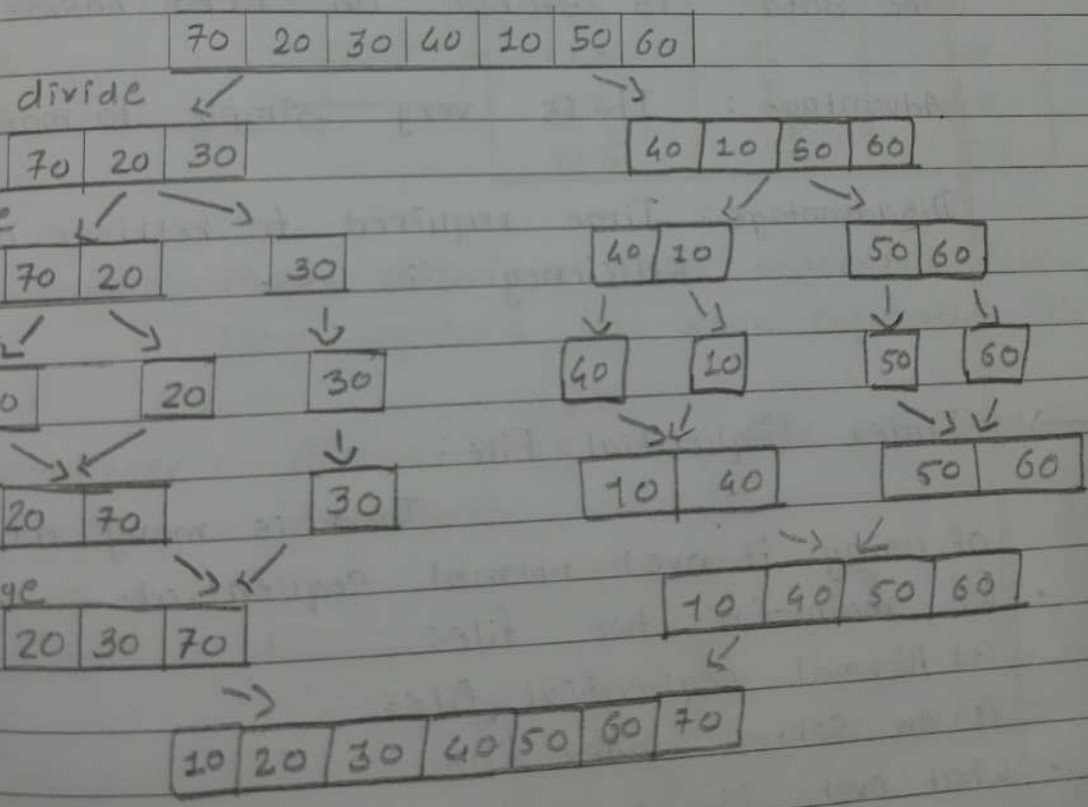
(1) Divide : Partition array into two sub list s1 and s2 with n/2 element each.

(2) Conqure : Then sort sub list s1 and s2

(3) Combine : Merge s1 and s2 unique sorted group.

Ex. 70, 20, 30, 40, 10, 50, 60

->



(c) Discuss sequential files and index sequential file.

→ Sequential File :

To understand this lets start with an example. Consider the example of tape rather a casatte where the song store sequentially. Whenever we want to play song from it we read it sequentially Storing data sequentially is the simplest form of file.
Reading data to a file and writing data frome it takes lot of time as data is not stored.
The data is stored on FCFS basis.

Advantage : It is very simple to manage.

Disadvantage : Time required to retrieve any record.
Efficiency is less.

→ Index Sequential File :

There is many advantage of using it over normal sequential.

- We maintain two files
  (1) Normal sequential files
  (2) An sorted index file
- What ever is our data we store it in the sequential file and in the index. file we have the id or primary key of the sequential file along with ofset of record in file.

Ex. We have to maintain grocery details, where in the sequential file.

item no. name price type

→ To refer any item we have to use normal sequential file, we would have to scan the entire file to find that particular item. But with index sequential we have another index file which will in this case contain.

item no. offset

item no.	offset
01	
03	

item no.	name	Price
01	A	10
02	B	20
03	C	30
04	D	40
05	E	50

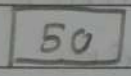
Advantage : More efficient than sequential file.
Time required less than sequential file.

Disadvantage : Maintaining two file increases the overhead.

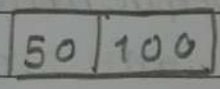
Q.5

(a) Create 2-3 tree for following sequence.
50, 100, 150, 200

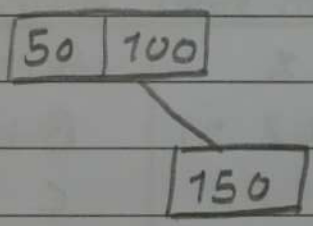
→ Step 1: insert 50



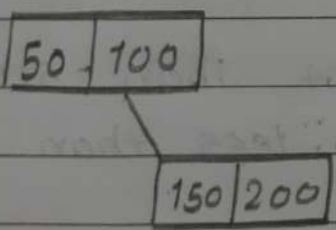
→ Step 2: insert 100



→ Step 3: insert 150

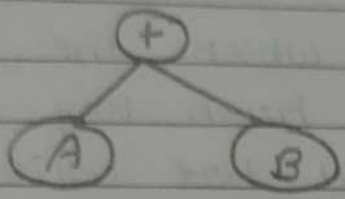


→ Step 4: insert 200

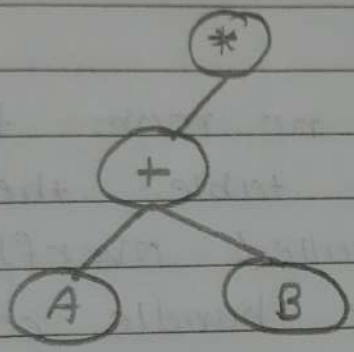


(b) Represent an expression tree: A + B * (C + D)

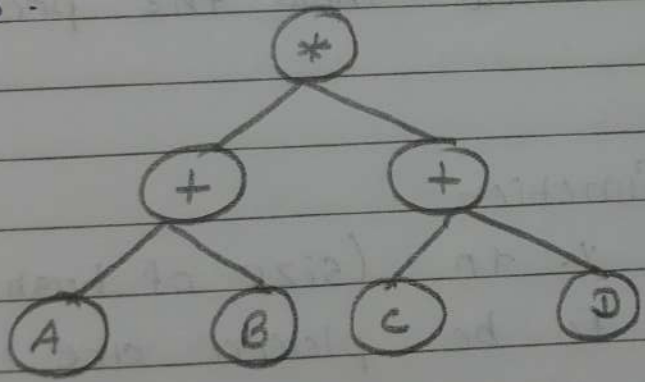
→ Step 1:



→ Step 2:



→ Step 3:



(c) State and explain collision resolution technique in hashing.

→ The situation in which the function return the same hash key for more than one record is called ~~hash~~ collision and two hash key return for different record is called synonyms.

→ Similarly there is no room for a new pair in the hash table then such a situation is called overflow. Sometimes when we handle collision it may lead to overflow condition. Collision and overflow show the poor hash function.

→ Consider hash function
H(key) = recordkey % 10 (size of hash table)
• The record key to be placed are
131, 44, 43, 78, 19, 36, 57 and 77

→ Now, if we try to place 77 in the hash table then we get the hash key 7 and at index 7 already the record key 57 is place. This situation is called collision. From index 7 if we look for next vacant position 8, 9 there is no room to place 77. This situation is called overflow.

0	
1	131
2	
3	43
4	44
5	
6	36
7	57
8	78
9	19

Summer - 2020

Q.1

(a) Difference between data type and structure.

Data Structure	Data type
→ Used to solving problems	It is represent basic type of data.
→ It can be reduced.	It can not be reduced.
→ It is formed using data type.	It is primitive data type.
→ Can hold different kind and types of data within one single object.	Can hold values and not data, so it is data less.
→ There is a time complexity when working with data structure.	There is no time complexity.
→ Ex. Stak, ~~Queue~~, Queues, tree	Ex. int, float, double.

(b)(a) Give Example of linear and Non-linear data Structure.

→ Linear Data-Structure :

It is a data structure when data is arranged in list or a straight sequence.

Ex. array , List , Stack , Queue

→ Non-linear Data-Structure :

It is a data structure where data is arranged in heirarchical manner.

Ex. trees, graph.

(b) What do you mean by Abstract data type.

→ The abstract data type is set of triple Domain (D) , Function (F) and axioms (A). This tripple (D,F,A) denotes the data structure.

→ In ADT only what is done is mentioned but how is to be done is not mentioned.

→ In ADT all the implementation details are hidden.

ADT = type + Function Name + Behaviour of each function

Q.2

(a) Difference between Stack and Queue.

	Stack	Queue
->	In this insertion and deletion of element can be done by one end.	In this insertion and deletion of element can be done by two end.
->	Stack is based on LIFO principal	Queue is based on FIFO principal.
->	In this we maintain only one pointor to access the list called top.	In this we maintain two pointor to access the Queue the front pointor and rear pointor.
->	It is used in solving problem works on recursion.	It is used to solving problem having sequential processing.

(b) What is top of Stack? Why Stack is called LIFO?

-> Top of Stack :

The top element is the element that is inserted at the last or most recently inserted element.

-> LIFO stands for Last in First Out. The element to be inserted last is to be deleted first.

(c) What is circular Queue? Write an algorithm to count the nodes in circular Queue.

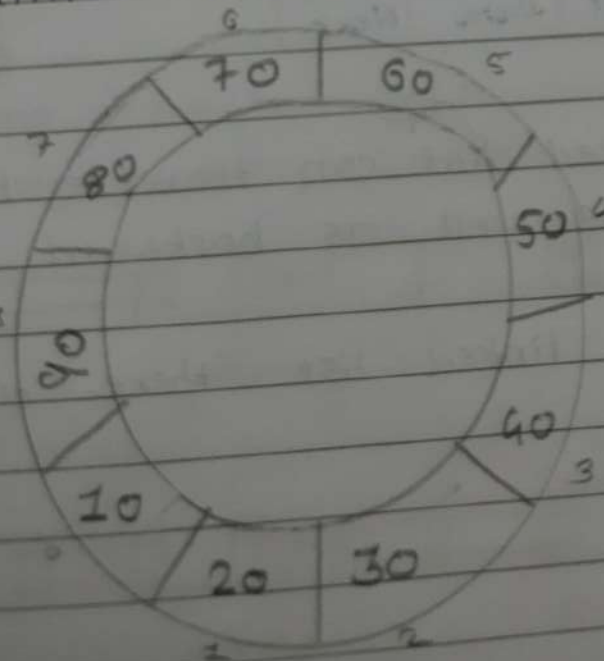
→ As we seen in case of linear queue element get deleted logically.

10	20	30	40	50	60	70
0	1	2	3	4	5	6

→ We can delete the element 10, 20, 30 means simply the front pointer is shifted ahead. We will consider a queue from front to rear always. And now if we try to insert any more element then it was not possible as it indicates "Queue full!" message.

→ Although there is a space of elements 10, 20, 30 we can not utilize them because queue is nothing but linear array.

→ Hence there is a concept of circular queue. The main advantage of circular queue is we can utilize the space of the queue fully.



-> There is a formula which can be applied for setting the front and rear pointers.

Insert :

rear = (Rear + 1) % size
= (4 + 1) % 5
= 0

-> Insert element at 0^{th} location.

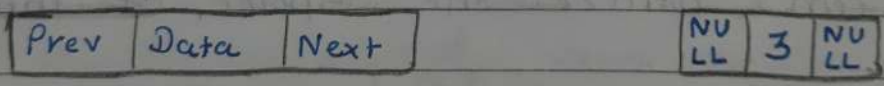
Delete :

front = (front + 1) % size
= (3 + 1) % 5
= 4

-> Delete element at 4^{th} location.

(c) Explain Creation, Insertion and Deletion in doubely link list.

-> Doubely link list has two link fields. One link field is previous pointer and the other link field is that next pointer.



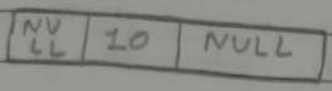
-> Double linked list can traverse both direction frorward as well as backward.

-> In doubly linked list there are following operation.

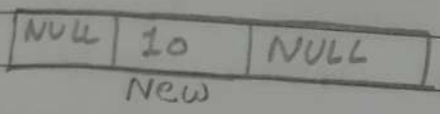
(i) Insert
(ii) Delete
(iii) Disply

* Insertion:

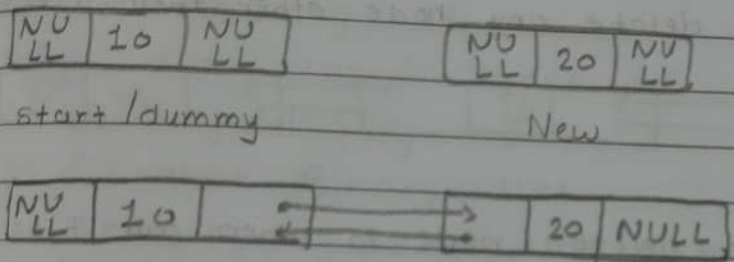
Step 1: Each node has doubly link list will look like this



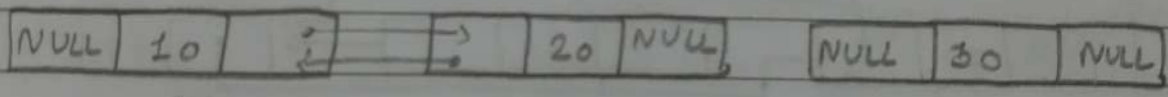
Step 2: Set a new node as



Step 3: For further addition of nodes the new node is created.

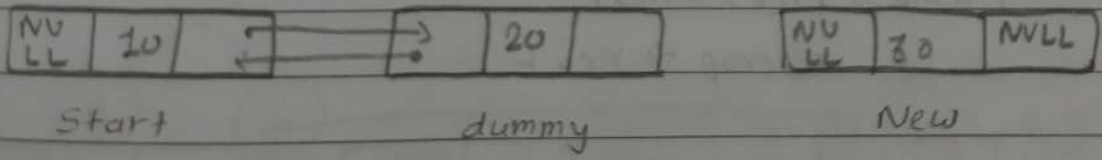


Step 4: For, further addition, New node is created

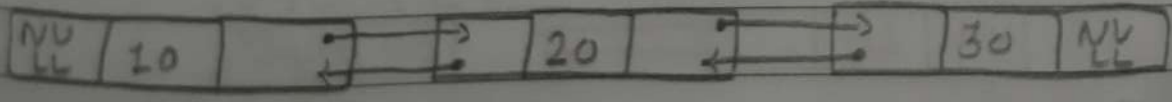


When, dummy -> next = NULL

dummy = dumy -> next

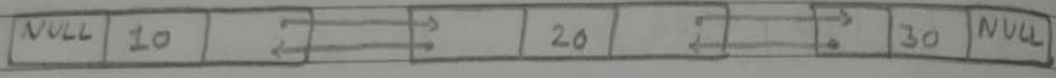


Attach new node



* Deletion

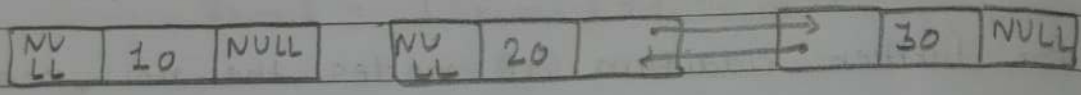
Step 1: We assume link list



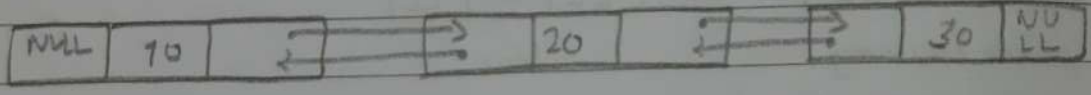
If we want to delete first node, then

Start -> Prev = NULL

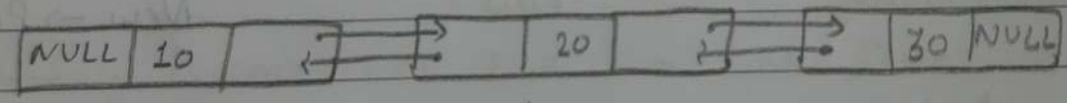
temp -> next = NULL

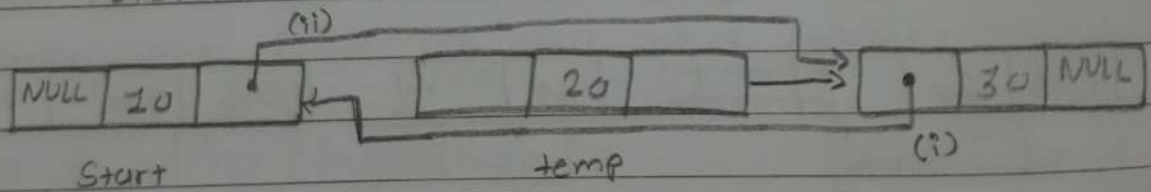


Step 2: If we want to delete any node other then first



If we want to delete node 20 then call it as temp.



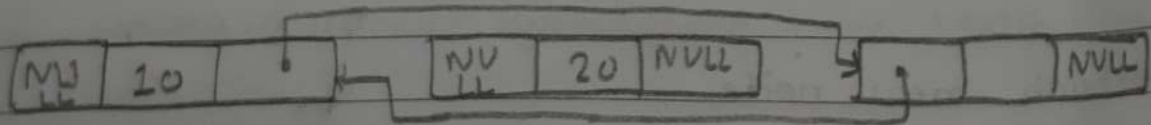


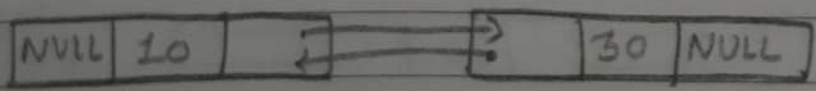
(i) (temp -> nex) -> Prev

= temp -> Prev

(temp -> Prev) -> next

= temp -> next





-> Thus node 20 get deleted.

Q.3

(a) What is binary tree? Mention different type of binary tree with example.

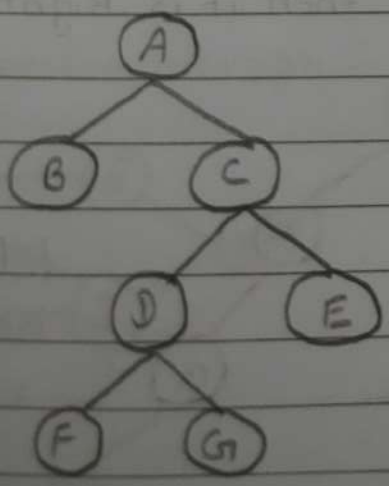
→ In the trees, we have number of child node to parent. If we put some restriction on child node. Ex. if we allow most two children nodes then it is called binary tree.

→ A binary tree is a finite set of nodes whose element have atmost 2 children is called binary tree.

→ There are three types of binary tree.
(i) Strictly Binary tree
(ii) Complete Binary tree
(iii) Left heavy and Right heavy tree

(i) Strictly Binary tree: If every non-leaf node in a binary tree has non-empty non Right and left sub tree it is called Strictly binary tree.

→ If there is N leaves, then 2N-1 nodes.



N = 4

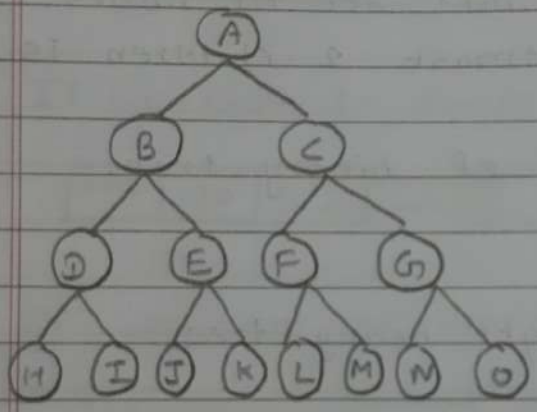
2N-1 = 2(4)-1 = 7 Nodes

(ii) Complete Binary Tree:

A complete binary tree is full binary tree in which all leaves are at same depth.

-> If binary tree contains at most one node at level. it can contain at most 2^l nodes at level l.

-> The binary tree of depth d that contains exactly 2^d nodes at level d.



Level-0

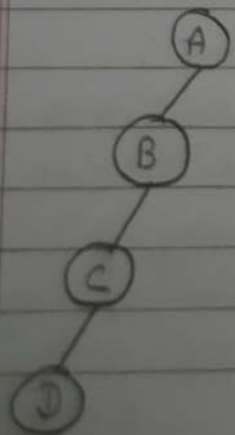
Level-1

Level-2

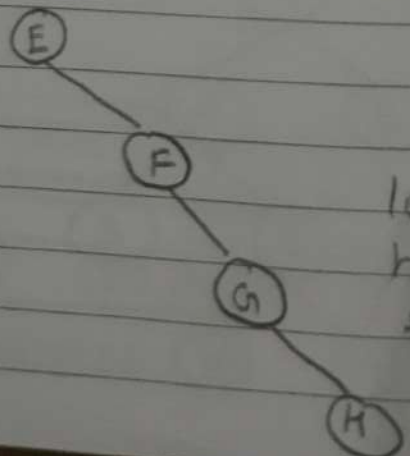
Level-3

[if, $l = 3$
$2^l = 2^3 = 8$]

(iii) Left Heavy and Right heavy tree: The tree in which each nod is attached as a left child of parent node then it is left heavy tree. The tree in which each node is attached as right child of parent node then it is Right heavy tree



Right heavy tree



left heavy tree

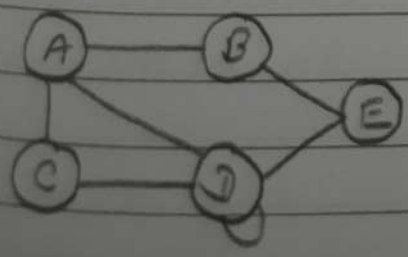
(b) What is graph ? Explain Representation of graph.

→ A graph is a collection of two sets V and E where V is finite non-empty set of vertices and E is finite non-emty set of edges.
→ Vetices are nothing but node in the graph.
→ Two adjacent verticies are joined by edges.
→ Any graph denoted as, $G = \{V, E\}$

→ Graph can be represented by two method.
(i) Adjacency Matrix
(ii) Adjacency list

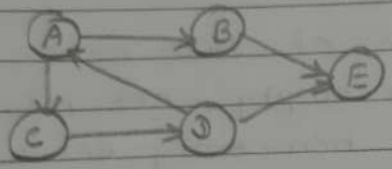
(i) Adjucency Matrix : In this representation matrix or 2 dimentional array is represent the graph.
→ A Graph $G = \{V, E\}$ where, $V = \{0, 1, \cdots, n-1\}$ can be represented using 2-dimentional array.
→ int adj [20] [20] can be used to store a graph with 20 vertices.
→ adj [i] [j] = 1, indicates presence of edge between two vertices i and j
→ adj [i] [j] = 0, indicates absence of edge between two vertices

Ex: Undirected Graph:



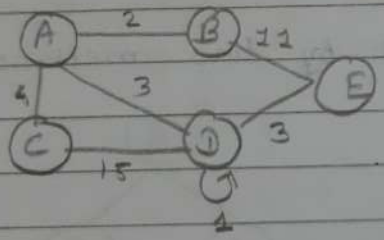
	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	0
D	1	1	1	1	1
E	0	1	0	1	0

Ex. Directed Graph:



	A	B	C	D	E
A	0	1	1	0	0
B	0	1	0	1	1
C	0	0	0	1	0
D	1	0	0	1	1
E	0	0	0	0	0

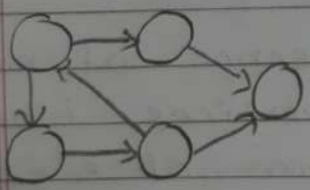
Ex. Weighted Graph:

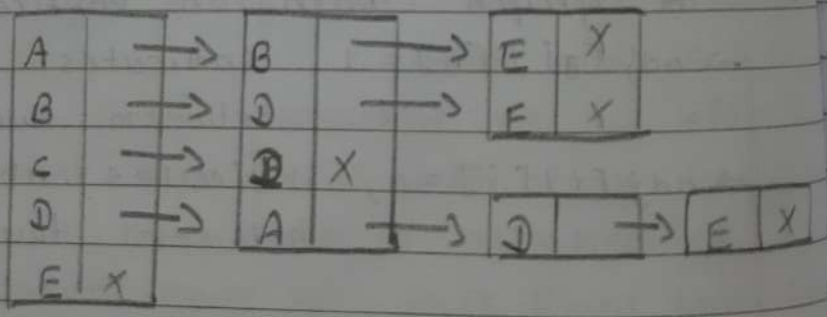


	A	B	C	D	E
A	0	2	4	3	0
B	2	0	0	6	11
C	4	0	0	15	0
D	3	6	15	1	3
E	0	11	0	3	0

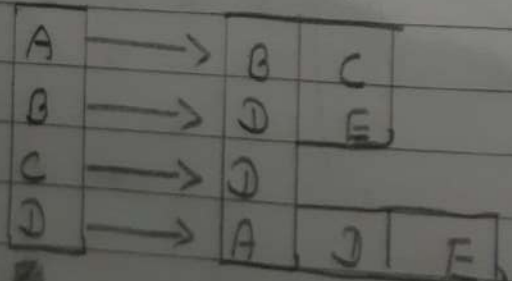
(ii) Adjancey list: In this representation every vertex of graph contains list of its adjacent vertex.

(a) Directed graph Representation using linked list





(b) Using Array.



(c) Write an algorithm to insert node in binary search tree.

-> While inserting any node in binary tree we have to look for its appropriate position in BST.

-> We have to compare this node with each node of BST.

-> If node which is inserted is greater than the current node we have to add those node at right side otherwise move on the left sub branch.

-> Algorithm : Tree to be searched from root node.

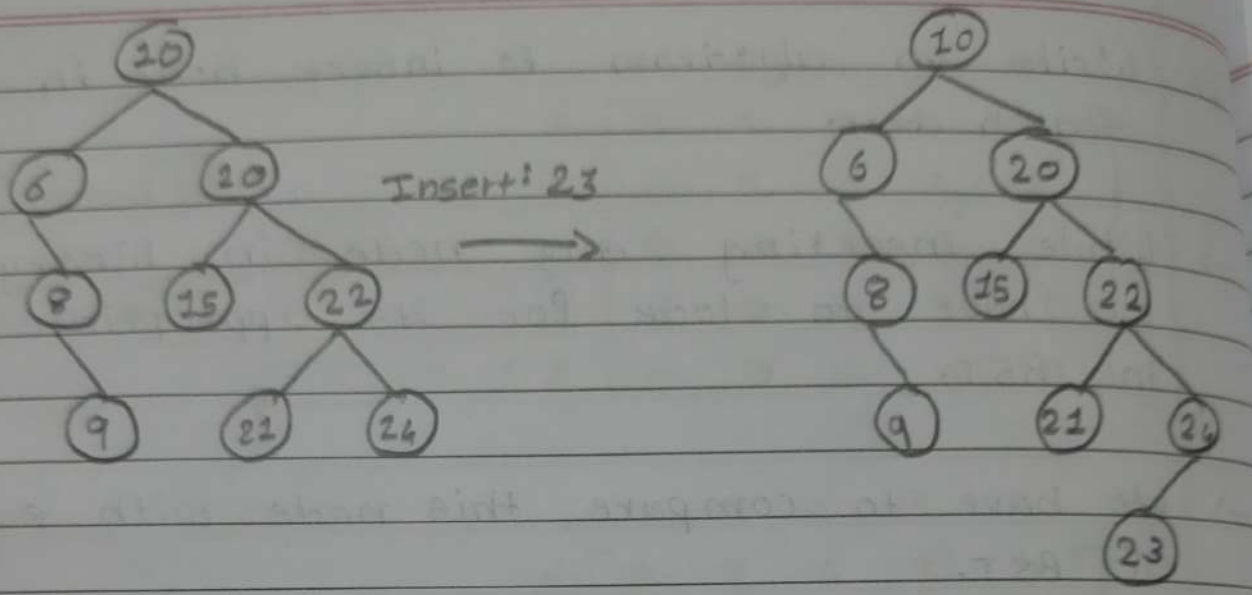
• If iteam is found do nothing. (or)

(i) Read the value for the node which is to be created and store it in a node called New

(ii) if (root != NULL) then root = New

(iii) Again read the next value of node created in New.

(iv) If (New->value < root->value) then attach new node as a left child of root. Otherwise attach New node as a right child of root.



→ If we want to insert 23. then we will start comparing 23 with value of root it is greater than 10. we will move towards right tree. Now we will compare 23 with 20 its greater than 20 then move right sub tree. Now compare 23 with 24 its less than 24 then attach 23 as left child of 24.

Q_3

(a) What is B-tree of order m? Draw B-tree of order 3.

→ B-tree specialized multiway tree used to store the record in a disk. There are number of sub tree to each node. So that the height of the tree is relatively small. So that only small number of nodes can be read to reteriew iteam. The goal of B-tree is to get fast acess of datu.

→ A multiway search tree of order m is an ordered tree where each node has atmost m children. If there is a n number of children in node then (n-1) is the number of key in the node.

* Properties

→ Every node in B-tree contains at most m children.

→ Every node in B-tree expect root node and leaf node contain m/2 children.

→ The root node must have 2 children.

→ All leaf not must be at the same level.

(b) Construct Binary tree:

Preorder : A B C D E F G H I

Inorder : B C A E D G H F I

→ PreOrder : [A] B C D E F G H I (D L R)

L.S R.S

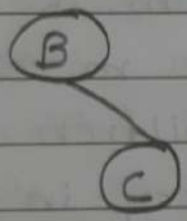
InOrder : B C [A] E D G H F I (L D R)

L.S R.S

- Left-Subtree

→ Pre : [B] C

In : [B] C



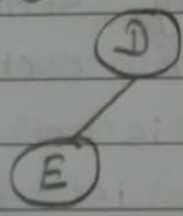
Right Subtree

(i) L.S R.S

Pre : [D] E F G H I

In : F [D] G H F I

L.S R.S



(ii) L.S R.S

Pre : [F] G H I

In : G H [F] I

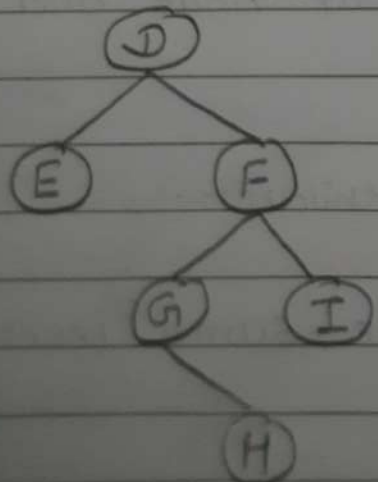
L.S R.S

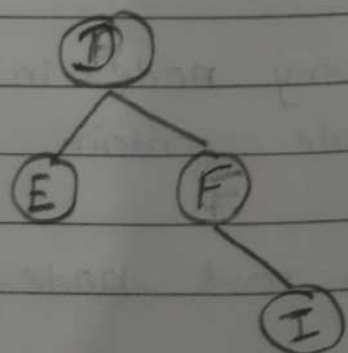
(iii) R.S

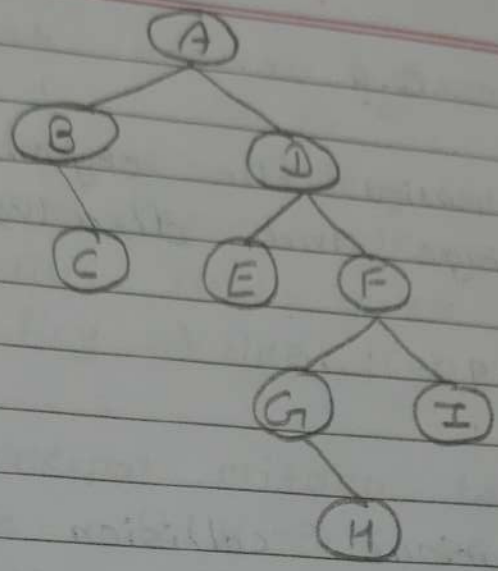
Pre : [G] H

In : [G] H

R.S







(c) Discuss algorithm for BFS traversal for graph.

→ Winter : 2019
Q.4 → (c)

Q.4

(a) Explain sequential file organization and list its advantage and disadvantage.

→ Winter : 2019
Q-5 → c

(c) Describe various collision resolution techniques in hashing.

→ Winter : 2019
→ Q.5 → c

Q.5

(a) Explain Indexed sequential file structure.

→ Winter : 2019
Q-5 → c

(b) Explain minimal spanning tree.

→ Winter : 2019
Q-3 → b

(a) What is hash function? Explain two method.

→ The reason behind hashing method is to find a one to one correspondence between hash key value and index in hash table. where the key value can be placed.

→ There are principal critaea to deciding a hash function. H: K → I as follows
(i) The function H should be easy to compute.
(ii) The function H should achieve on even distribution of key that actually occure across the range of indices.

→ There are five types of hashing method.
(i) Division method
(ii) Mid square method
(iii) Multiplicative hash function
(iv) Digit folding
(v) Digit anaylising

(i) Division Method

→ The hash function depends upon a reminder of division.
→ Typically the devisor is the table length.

→ If the record 54, 72, 89, 37 is to be placed in the hash table and table size is 10, then,

h(key) = record %. table size

4 = 54 %. 10

2 = 72 %. 10

9 = 89 %. 10

7 = 37 %. 10

0	
1	
2	72
3	
4	54
5	
6	
7	37
8	
9	89.

(ii) Mid Square Method.

→ In mid square method key is squared and the middle part of result is use as index.

→ If key is a string then it is pre proced to produce a number.

→ Consider if we want to place a record 3111

$(3111)^2 = 9678321$

H(3111) = 783 (Middle 3 digit)

(iii) Digit folding :

→ The key is divided into some part and using simple operation this part are combined to produce key.

Ex. 123654123

H(key) = 123 + 654 + 012
= 789

→ The record will be placed at location 789 in hash table.

Q.5

(a) Define topological sort.

→ Topological sorting for Directed Acyclic graph (DAG) is a linear ordering of vertices such that for every directed edge U → V, vertex U comes before V in the ordering.

→ Topological sorting for a graph is not possible if the graph is not a DAG.

(b) Compare Sequential and Binary Search.

→

→ Sequential search	Binary search
• For searching element by using method it is not required to arrange the element in specific order.	The elements need to be arranged either in ascending or descending order.
• Each and every element is compared with the key element from the beginning of the list.	The list is subdivided into two sublist. The key element is searched in the sublist.
• Less efficent method	Efficient method
• Simple to implement.	Additional computation is required for computing mid element.

Ans

(c) What is insertion sort ? Give Solve the insertion sort and sort the array:
77, 33, 44, 11, 88, 22, 66, 55

→ Insertion sort is a simple sorting algorithm that works similar to the way you sort playing card in your hands.
→ The array is virtually split into two part sorted and unsorted.
→ Values from the unsorted part are picked and placed at correct position in sorted part.

→

0	1	2	3	4	5	6	7
77	33	44	11	88	22	66	55

0	1	2	3	4	5	6	7
77	33	44	11	88	22	66	55

0	1	2	3	4	5	6	7
33	77	44	11	88	22	66	55

0	1	2	3	4	5	6	7
33	44	77	11	88	22	66	55

0	1	2	3	4	5	6	7
11	33	44	77	88	22	66	55

0	1	2	3	4	5	6	7
11	33	44	77	88	22	66	55

0	1	2	3	4	5	6	7
11	22	33	44	77	88	66	55

0	1	2	3	4	5	6	7
11	22	33	44	66	77	88	55

0	1	2	3	4	5	6	7
11	22	33	44	55	66	77	88

Ans

Q.5

(a) What do you mean by Internal sorting and external sorting.

→ Internal sorting :

If the data sorting process takes place entirely within the random-Access memory (RAM) of a computer it is called internal sorting.

→ External sorting :

The sorting of large database using hard-disk will require different sets of algorithm which are called external sorting.

(b) Write an algorithm for quick sort.

→ (1) Choose the highest index value has pivot.
(2) Take two variable to point left and right of list excluding pivot.
(3) left points to the low index.
(4) Right points to the high index.
(5) while value at left is less than pivot move right.
(6) While value at left is grater than pivot move left.
(7) If step : 5 and 6 does not match swap left and right.
(8) If left ≥ Right the point where they met is new pivot.

(c) What is binary search tree? Construct a binary tree for following element.
21, 16, 24, 18, 22, 25, 26, 27, 29, 33.

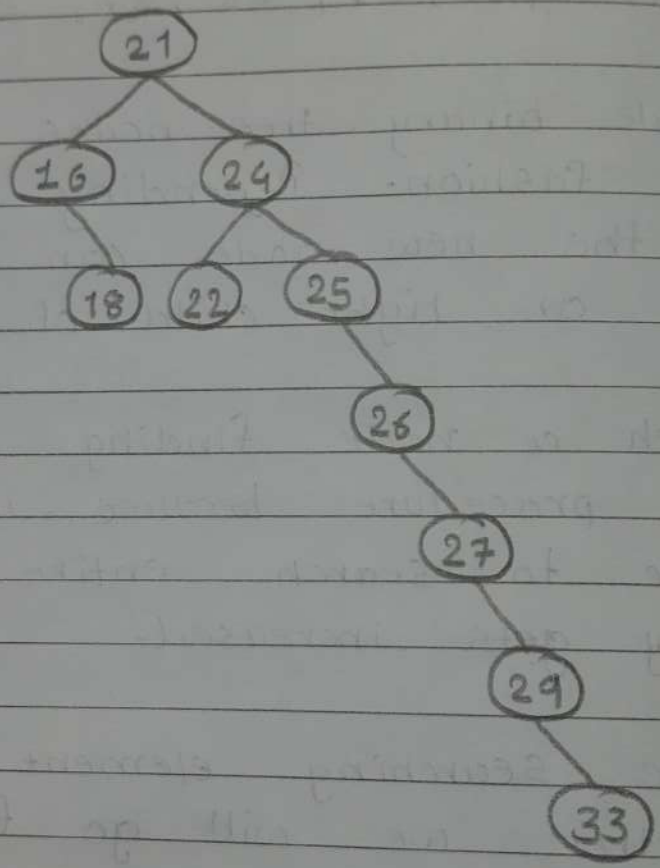
-> In simple binary tree nodes are arranged in any fashion. Depending on user's desire the new node can be attach as a left or right child of any desired node.

-> In such a node finding for any node is a long procedure because in this case we have to search entire tree. Thus the complexity gets increased.

-> To make searching element faster in a binary tree we will go for building binary search tree.

-> Rules: Left sub-tree < Root / Parent
Right sub-tree > Root / Parent.

-> 21, 16, 24, 18, 22, 25, 26, 27, 29, 33



Winter : 2020

Q.1

(a) Compare array and linked list.

→ Winter : 2019
Q.2 → (a)

(b) Compare primitive and non-primitive data type.

→

Primitive	Non-Primitive
• It is used to stores the data of only one type.	It is used to stores the data of more than one type.
• It can contain some value it can't be NULL.	It can consist NULL value.
• Size is depend on the type of the data structure.	In this size is not fixed.
• Start with lowercase character.	Start with an uppercase character.
• Ex. integer, character, float	Ex. Array, linked list, stack

(c) Write an algorithm to perform insert and delete operation in simple queue.

→ Insert Operation :

```
(1) If (REAR = Size) then,
(2) print "Queue is full."
(3) Exite
(4) Else
(5)     if (FRONT=0) and (REAR=0) then
(6)         FRONT = 1
(7)     End if,
(8)     REAR = REAR + 1
(9)     Queue[REAR] = ITEM
(10) End if
(11) Stop
```

→ Delete Operation :

```
(1) If (FRONT=0) then
(2)     print "Queue is empty"
(3)  Exit.
(4)  Else
(5)     ITEM = Que [FRONT]
(6)     if (FRONT = REAR)
(7)         REAR = 0
(8)         FRONT = 0
(9)  Else
(10)        FRONT = FRONT + 1
(11)  End if
(12) End if
(13)  Stop.
```

Q.2

(a) Search the numer 50 in given data using binary search technique. 10, 14, 20, 39, 41, 45, 49, 50, 60.

0	1	2	3	4	5	6	7	8
10	14	20	39	41	45	49	50	60

• We want to find 50 then,

$mid = \frac{start + end}{2} = \frac{0+8}{2} = 4$

$mid = A[4]$

$\therefore 50 > A[4]$

$\therefore start = mid + 1$

5	6	7	8
45	49	50	60

→

$mid = \frac{5+8}{2} = \frac{13}{2} \approx 6$

$mid = A[6]$

$\therefore 50 > A[6]$

$\therefore start = mid + 1$

7	8
50	60

→

$mid = \frac{7+8}{2} = \frac{15}{2} \approx 7.$

$A[7] = mid$

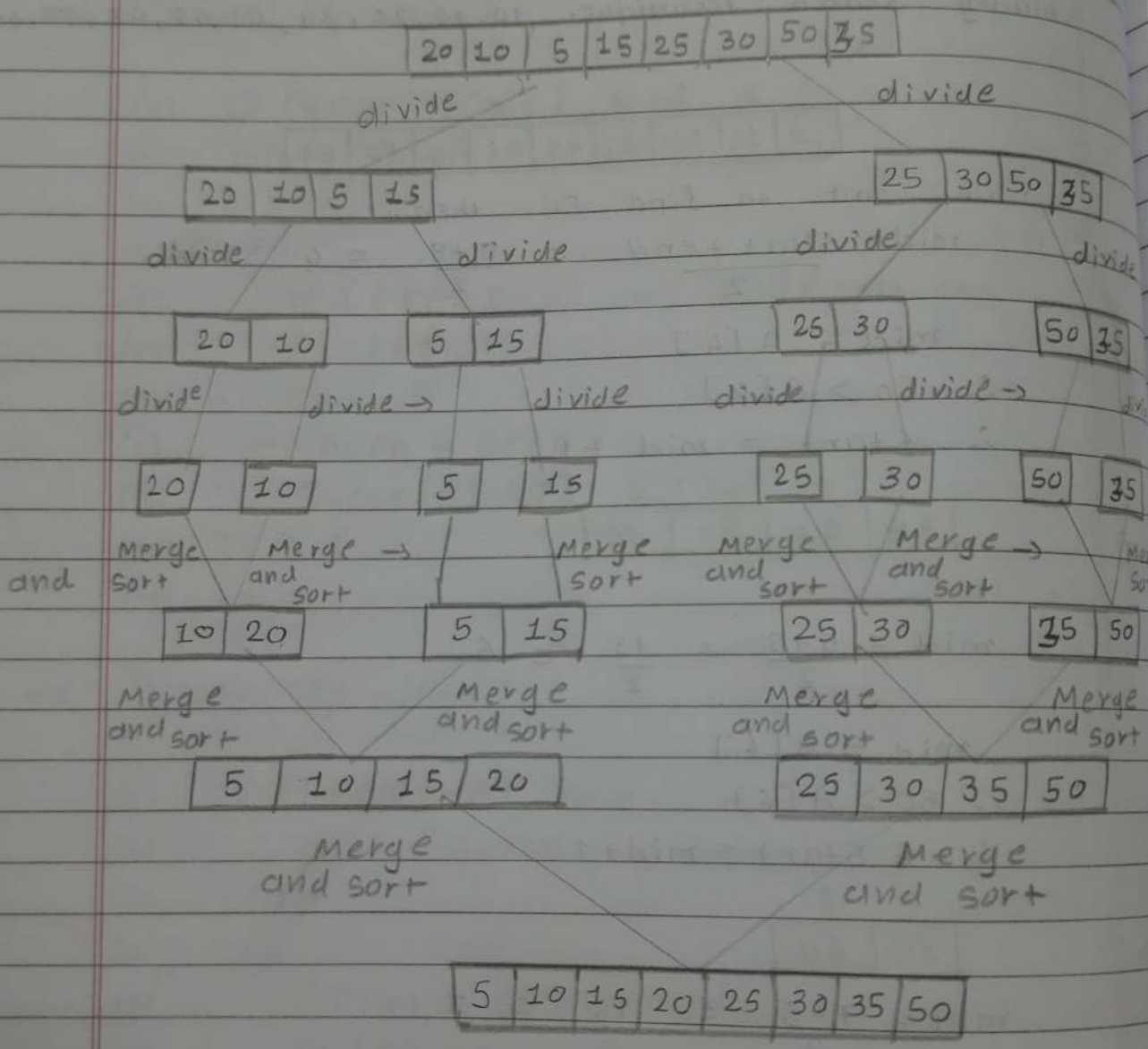
$A[7] = 50$

$\therefore 50 = A[7]$

$\therefore$ 50 is found At index 7.
So it is succesfull array.

(b) Apply merge sort algorithm.

20, 10, 5, 15, 25, 30, 50, 35



→ Sorted array :

5 10 15 20 25 30 35 50

(c) Write C program for bubble sort.

→ Winter : 2019
Q. 5→(a)

Q.3

(a) What is stack ? Why we use multiple stack?

→ A stack is an ordered list in which all insertion and delition are made at one end called top.

→ If we want to make stack of 10, 20, 30, 40, 50, 60 then 10 is bottomost element and 60 is topmost element.

Top →	60
	50
	40
	30
	20
	10

→ A single stack sometimes not suffcient to store large data to overcome this problem we can use multiple stack. For this we have to use single array having more than one stack. The array is divided for multiple stack.

(b) Convert infinix to prefix and postfix.

(i) A * B + C / D

(ii) (A * B) + (C / D) - (D + E)

→ Infinix to postfix :

(i) A * B + C / D
= AB *+C / D
= AB *+CD /
= AB * CD / +

(ii) (A * B) + (C / D) - (D + E)
= AB * + CD / - DE +
= AB * CD / + DE + -

Infinix to Prefix →

(ii) (A * B) + (C / D) - (D + E)
= * AB + / CD - + DE
= + * AB / CD - + DE
= - + * AB / CD + DE

(i) A * B + C / D
= * AB + C / D
= * AB + / CD
= + * AB / CD

Q.4

(a) Evalution of postfix expression
53 + 62 / * 35 * +

→ Symbol	Stack	Remark
5	5	5 → Operand push to stack
3	3 5	3 → Operand push to stack.
+	8	+ → pop 3 and 5 and perform the operation
6	6 8	6 → Operand push to stack
2	2 6 8	2 → Operand push to stack
/	3 8	/ → pop 6 and 2 and perform the operation
*	24	* → pop 8 and 3 and perform the operation
3	3 24	3 → operand push to stack.
5	5 3 24	5 → operand push to stack
*	15 24	* → pop 3 and 5 and perform the operation
+	39	+ → pop 24 and 15 and perform the operation.

Ans : 39

(b) Design an algorithm to perform insert operation in circular queue.

→ Winter : 2019
Q. 2 → (c)

(c) Design an algorithm to merge two linked list.

→
```
ListNode merge (ListNode A, ListNode B) {
  if (A == NULL) {
      return A ; }
  if ( B == NULL) {
      return B ; }

ListNode output = NULL

  if (A.val < B.val)
    {
      output = A ;
      A.next = merge (A.next, B) ;
    }
  else {
      output = B ;
      B.next = merge (A, B.next) ;
    }
  return output

}
```

Q.5

(a) Define : (i) Acyclic Graph (ii) Leaf node
(ii) Complete binary tree.

-> Acyclic Graph :

A graph which is not containing any cycle in it is called as acyclic graph.

-> Leaf Node :

The node which does not have ~~Contain~~ a child is called leaf node.

-> Complete Binary tree :

A complete binary tree is full binary tree in which all leaves are at same depth.

Q.6

(a) Define : (i) Connected graph (ii) Threaded tree (iii) Degree of node.

→ Connected graph :

A graph in which there is a path joining each pair of vertices, the graph being undirected. it is always possible to travel in connected graph between one vertex and any other - no vertex is isolated it is called connected graph.

→ Threaded tree :

Threaded binary tree is a simple binary tree but they have a speciality that null pointers of leaf node of the binary tree is set to be inorder predecessor or inorder successor.

→ Degree of node :

It is a total number of children of that node.

(b) Compare DFS and BFS

BFS	DFS
• BFS stands for Bredth first search	DFS stand for Depth first search
• It is use queue data structure for finding shortest part.	It is using stack data structure.
• BFS can be used to find single source shortest path in an unweighted graph.	In DFS we might traverse through more edges to reach a destination.
• BFS is used to searching element which are closer.	DFS is used to searching element which are away.
• Here sibling are visited before the children	Here children are visited before the sibling.

(c) Design an algorithm to insert in binary search tree.

⇒ Summer : 2020

Q.3 → (c)

Q.7

(a) Explain basic file operation.

-> A file is a collection of logically related data that is stored in second storage in form of sequence of operation.

-> The content of file is defined by creator who is created file.

-> The various operation such as read, write, open and close etc. are called file operation.

(i) Create Operation: It is used to creat a file in the file system. It is most widely used operation perform on the file system.

(ii) Write Operation: It is used to write information in a file. It is issued that specifies name, length of data hush to be written on the file.

(iii) Read operation: This operation reads the content of file. It is maintain by the OS, pointing to position which data has been read.

(iv) Delete Operation: It will not only delete all the data stored in file but it is also used so that disk space occupied by it is freed.

(b) Application of Hashing

-> In compilers to keep track of declared variables

-> For Online spelling checking hashing function is used.

-> Hashing help in Game playing programs.

-> For browser program, while catching the web pages hashing is used.

-> Cryptographic hash function is used in password verification

-> To examine similar data and to locate modified files, cloud storage service use hash task.

Summer: 2021

Q.1

(a) Explain Primitive and non-primitive data type.

→ Winter: 2020
Q.1 → (a)

(b) Explain Binary search with an example.

→ Search a sorted array by repetedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than iteam in the middle of the interval narrow the internal lower half. Otherwise narrow it to upper half. Repeadatly check untill the value find.

→ Ex. Wintere: 2020
Q.2 → (a)

(c) Explain asymptotic Notation.

→ The main idea of asymptotic notation is used to represent the complexities of algorithm for asymptotic analysis.
→ These notation are mathamatical tolls to represent complexity.
→ There are three notation that commonly used.

(i) Big Oh Notation :

Big-Oh (O) notation gives an upper bound for a function f(n) to within a constant factor.

(ii) Big Omega Notation : Big-Omega (Ω) notation gives a lower bound for a function f(n) to within constant factor.

(iii) Big Theta Notation : Big-theta (Θ) notation gives bound for a function f(n) to within constant factor.

Q.2

(a) Differenciate Static and dynamic memory allocation.

→

Static Memory	Dynamic Memory
• In this allocation variable get allocated permanantly till the program executed	In this variable get allocated only if your your program is active.
• Allocation is done before program execution.	Allocation is done during program execution
• It is use stack for managing the static allocation of memory	It is use heap for managing the dynamic allocation of memory.
• It is less efficient	It is more efficient.
• There is no memory reUsability	There is memory re-usability and memory gets free when not required
• In this memory size can not change.	Memory size can be changed.
• Ex. Array	Ex. linked list

(b) Explain linear and Non-linear data structure with an example.

→ linear data structure :

• Memory is not utilized in efficient way. • Used in software devlopment.	Data structure where data elements are arranged sequentially and linearly is called linear data structure.

Ex. Array, Stack, Queue, linked list

→ Non-linear data structure :

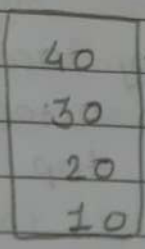
• Memory is utilized in efficient way. • Used in artificial intelligence and image processing.	Data structure where data elements are not arranged in sequentially and linearly is called Non-linear data structure.

Ex. Trees, graph

Q) What is stack? Explain operation in details.

→ A stack is an ordered list in which all insertion and deletion are made at one end called the top.

→ If we have to make stack of element 10, 20, 30, 40 then 10 will be bottom most element and 40 will be topmost element.



→ A stack can be implemented by means of array, structure, pointor and linked list.

→ Stack can be either be a fixed size one or it may have a sense of dynmaic resizing.

* Push Operation

→ The process of insert new data element onto stack is known as push operation.

(i) Checks if the stack is full.
(ii) if the stack is full produce an error and exite.
(iii) if the stack is not full, increament top to point next empty space.
(iv) Add data element to the stack location where top is pointing.
(v) Return sucess.

* Pop Operation :

-> The process of delete data element from the stack is called pop operation.
-> Pop removes the data element and deallocate memory space.

Step: (i) Check if the stack is empty
(ii) If the stack is empty, produce an error and exite.
(iii) if the stack is not empty accessing the data element at which top is pointing.
(iv) Decrease the value of top by 1.
(v) Return sucess.

Q.3

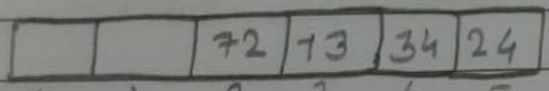
(a) Explain advantage of circular queue.

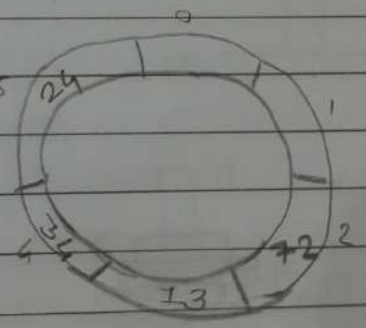
→ As the insertion in the queue is from the rear end and in the case of linear queue insertion is not possible where rear reaches the end of the queue.

→ But in the case of circular queue the rear end moves from the last position to the front position circularly.

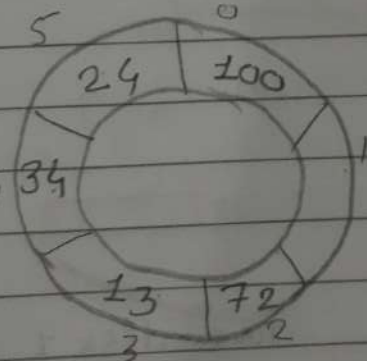
→ The circular queue has more advantage to linear queue.
(i) Easier for insertion-deletion
(ii) Efficient utilization of memory
(iii) Ease of performing operation.

Ex. Assume the queue.





→ If, we want to insert element 100. So we can not insert element in liniear queue but we can insert element in circular queue.



(b) Explain tower of Hanoi.

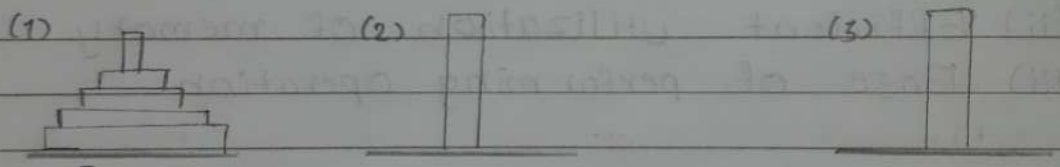
→ Suppose three tower labled 1, 2 and 3 are given.

→ N no. of disc with decreasing size placed on tower 1.

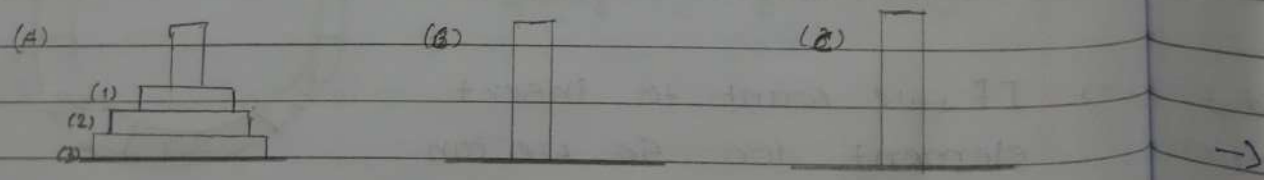
→ Aim of the game is to move all the disk from tower 1 to 3 using tower 2 as intermidate.

Rules: (i) At a time only one top most disk can be remove from one tower and add another tower.

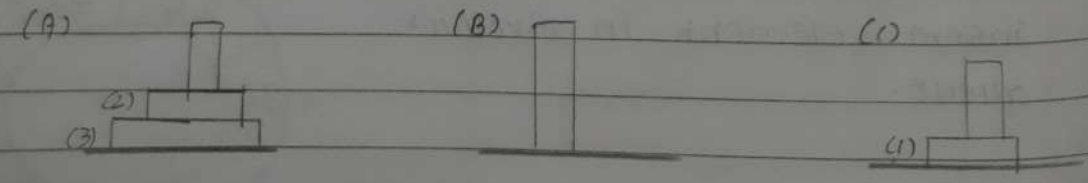
(ii) Larger disk can note be placed on smaller disk.



Ex.

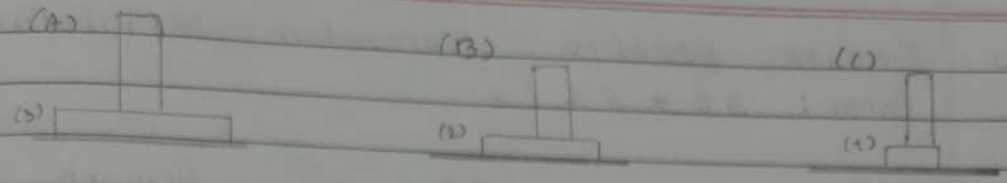


→



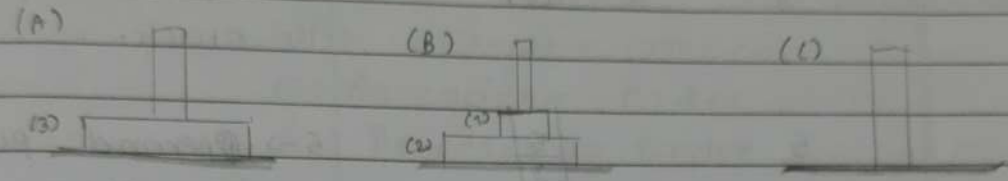
Move disk 1 from A to C

→



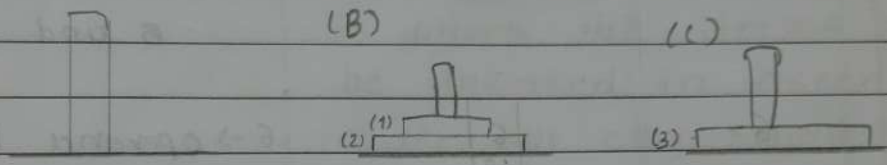
Move disk 2 from A to B

→



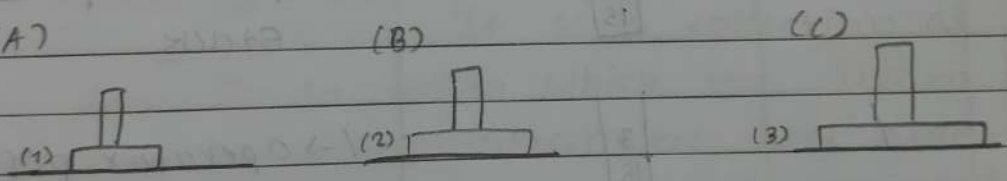
Move disk 1 from C to B

→



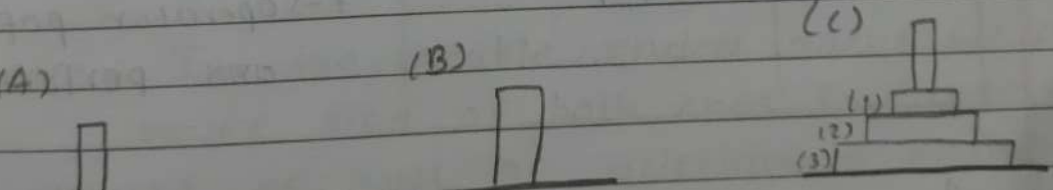
Move disk 3 from A to C

→



Move disk 1 from B to A

→



Move disk ~~to~~ 2 from B to C.
Move disk 1 from A to C.

Q.3

(a) Evalute postfix expression in tabular form : 3 5 * 6 2 / +

Symbol	Stack	Remark
3	3	3 -> operand push to the stack.
5	5 3	5 -> operand push to the stack.
*	15	* -> operator pop 3 and 5 and perform operation
6	6 15	6 -> operand push to the stack.
2	2 6 15	2 -> operand push to the stack.
/	3 15	/ -> operator pop 6 and 2 and perform operation
+	18	+ -> operator pop 15 and 3 and perform operation

Ans : 18

(b) Explain dequeue and priority queue.

→ Priority Queue:

The priority queue is a data structure having a collection of elements which are associated with specific order.

(i) Ascending order.

(ii) Descending order.

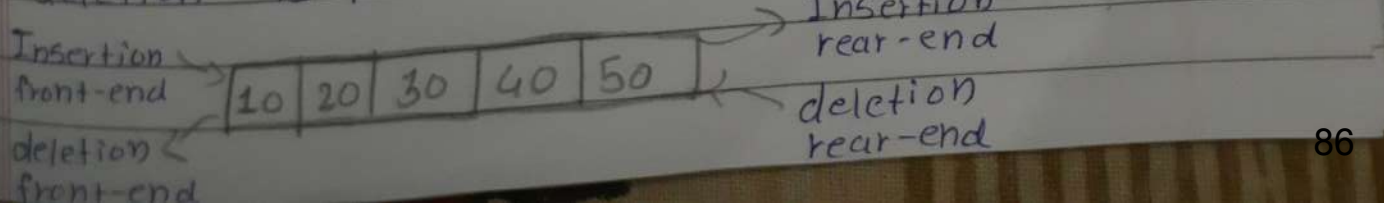
(i) Ascending order:

It is a collection of iteam in which the iteams can be inserted in ascending order but only smallest element can be removed.

(ii) Descending order:

It is a collection of iteam in which the iteam can arranged any order but only largest element can be removed.

→ Dequeue: In the double ended queue we can make use of both ends for insert the element as well as deletion of element. That means it is possible to insert element as well as front and by rear. And similarly deletion is possible to rear as well as front end.



Q.4 (a) Define (i) Sibling (ii) Forest (iii) Strictly Binary tree (b)

→

Sibling :

Nodes which becomes same parent are called siblings.

→ Forest :

A forest is an undirected graph in which any two verticies are connected by at most one path. The graph consists of a disjoint union of tree.

→ Strictly binary tree :

If every non-leaf node in binary tree has non empty left and right subtree it is called strictly binary tree.

or

All the nodes in ~~binary~~ strictly binary tree are of degree zero or two never degree one.

(c) Explain prim's algorithm with example.

→ Winter : 2019

Q.3 → (b)

Q.5

(a) Explain index file Organization.

→ An index file contains record ordered by a record key.

→ A record key uniquely identifies a record and determines the sequence in which it is accessed with respect to other record.

→ Each record contain field which contains record key.

→ A record key for a record might be an employee number, or an invoice number.

→ An index file can also use alternate indexes, that is record key that let you access the file using different logical arrangment. ex. you could access data through employee departmant rather than employee number.

(b) Explain rotation Rule for height balanced (AVL) tree.

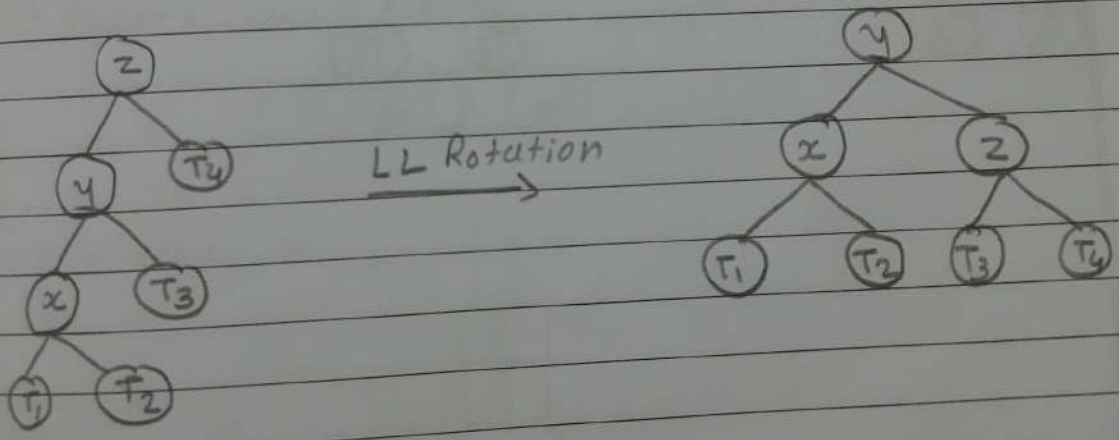
→ An empty tree is height balance if T is a non-empty binary tree will T_L and T_R as it's left and Right sub tree.

→ The tree is height balance tree if and if only
(i) T_L and T_R are height balanced
(ii) $(h_L - h_R) = -1 \leq 0 \leq 1$

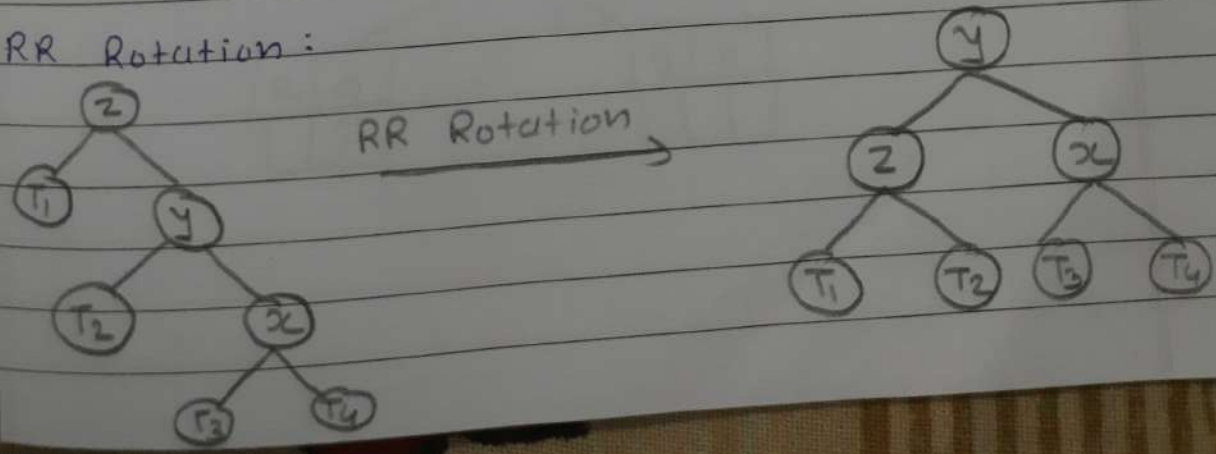
→ B(F) = -1, 0, 1

→ There are four type of rotation.
(i) LL Rotation (iii) LR Rotation
(ii) RR Rotation (iv) RL Rotation

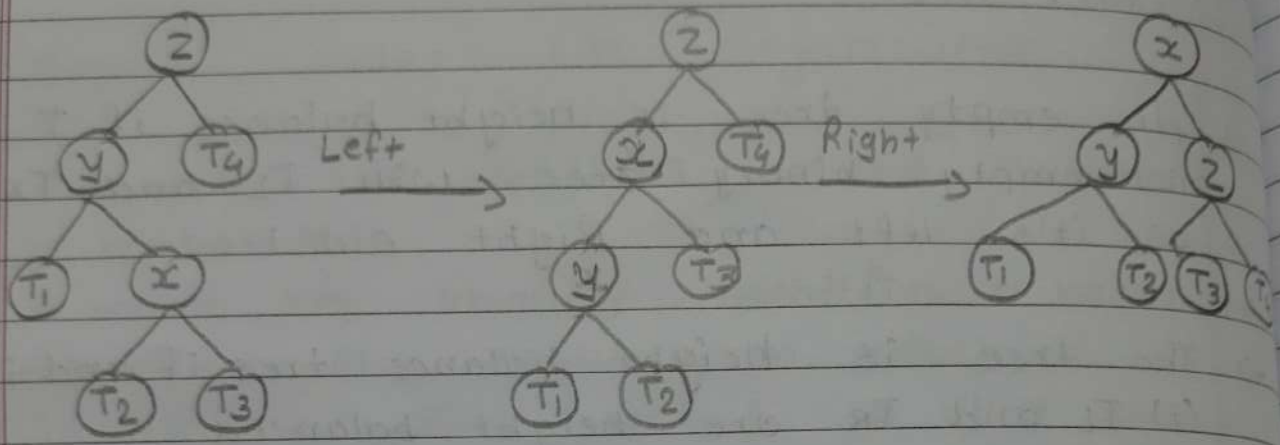
(i) LL Rotation.



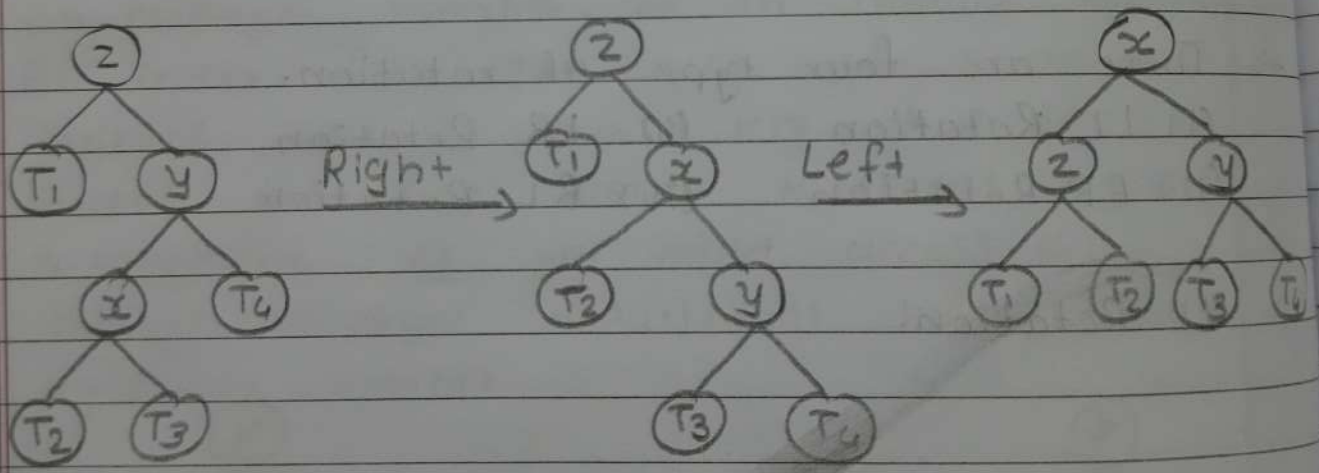
(ii) RR Rotation:



(iii) LR Rotation :



(iv) RL Rotation :



(c)

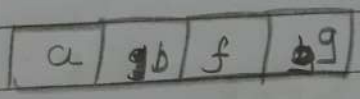
(c) Insertion and deletion in B-tree.

-> Insertion :

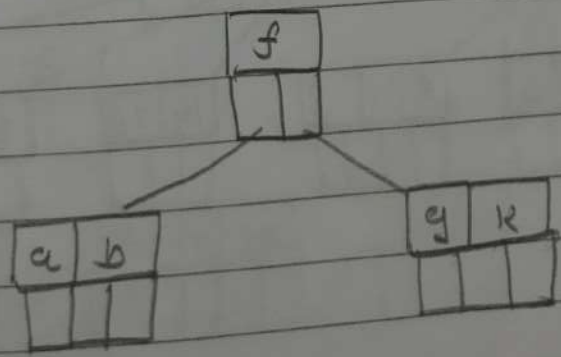
- Run the search algorithm to find appropriate location to insertion.
- Inser the new key at the proper location, but if node has a maximum number of key already
- The node along with a newly inserted key, will split from middle element.
- The middle element becomes parent for the other two children.
- The nodes must be rearrange keys in ascending order.

Ex. Construct B-tree [a, g, f, b, k, c, h, n, i]

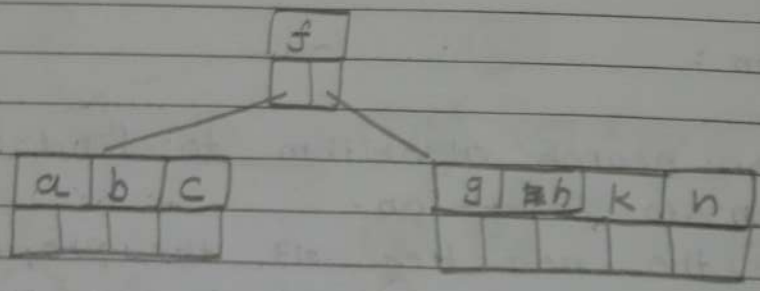
-> insert a, g, f, b



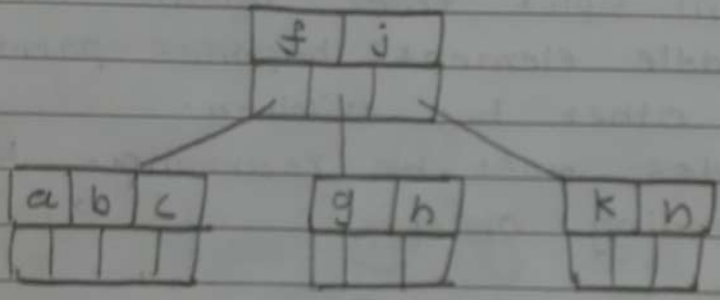
-> insert k



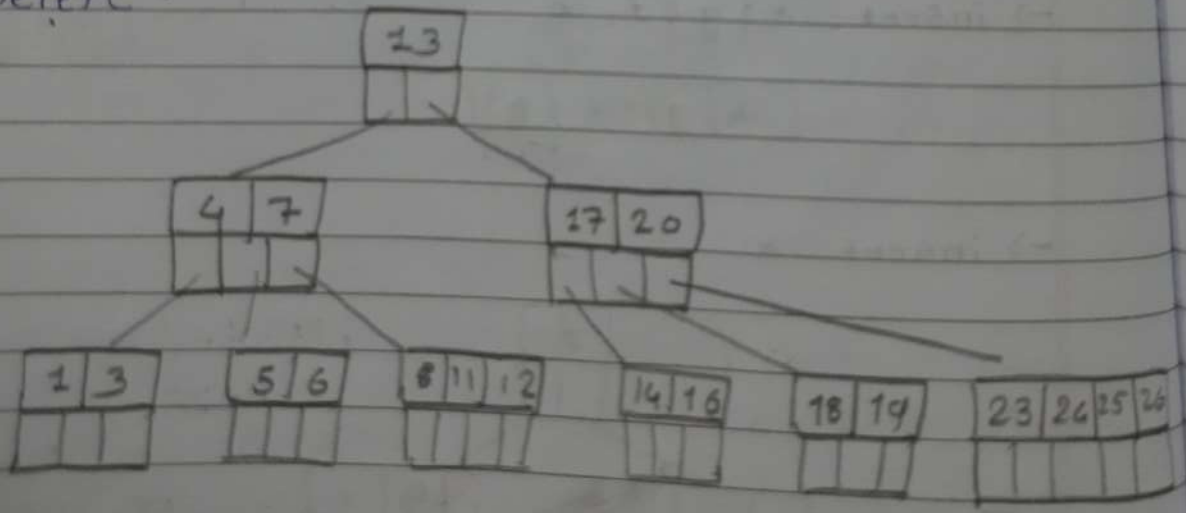
-> insert : c, h, n



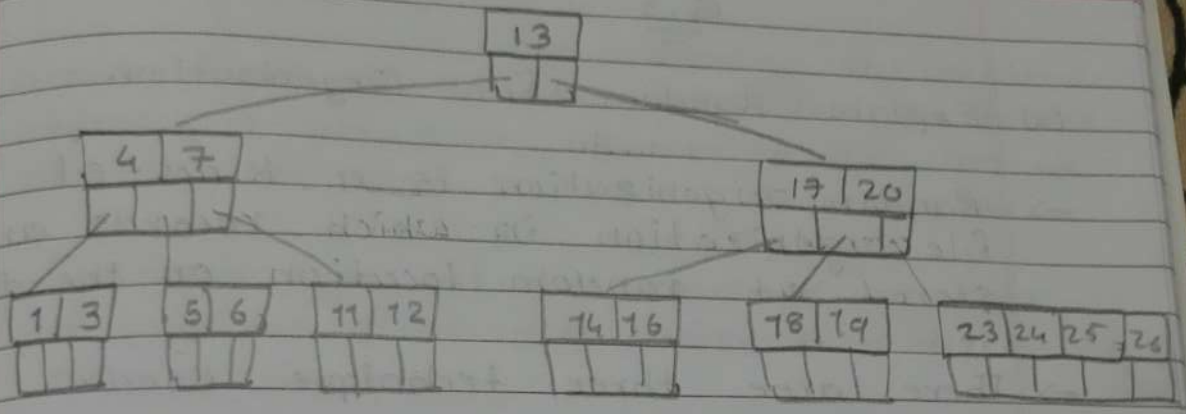
-> insert : j



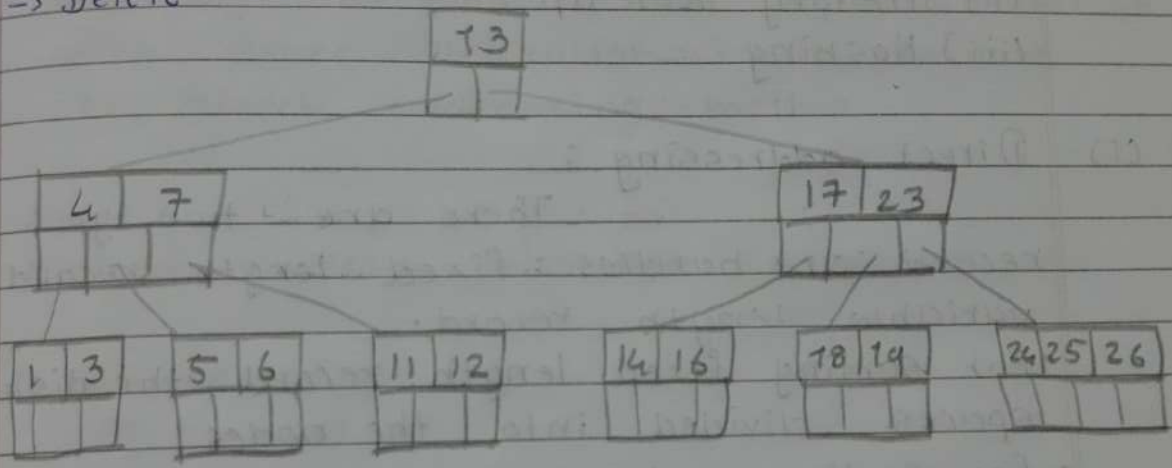
-> Delete :



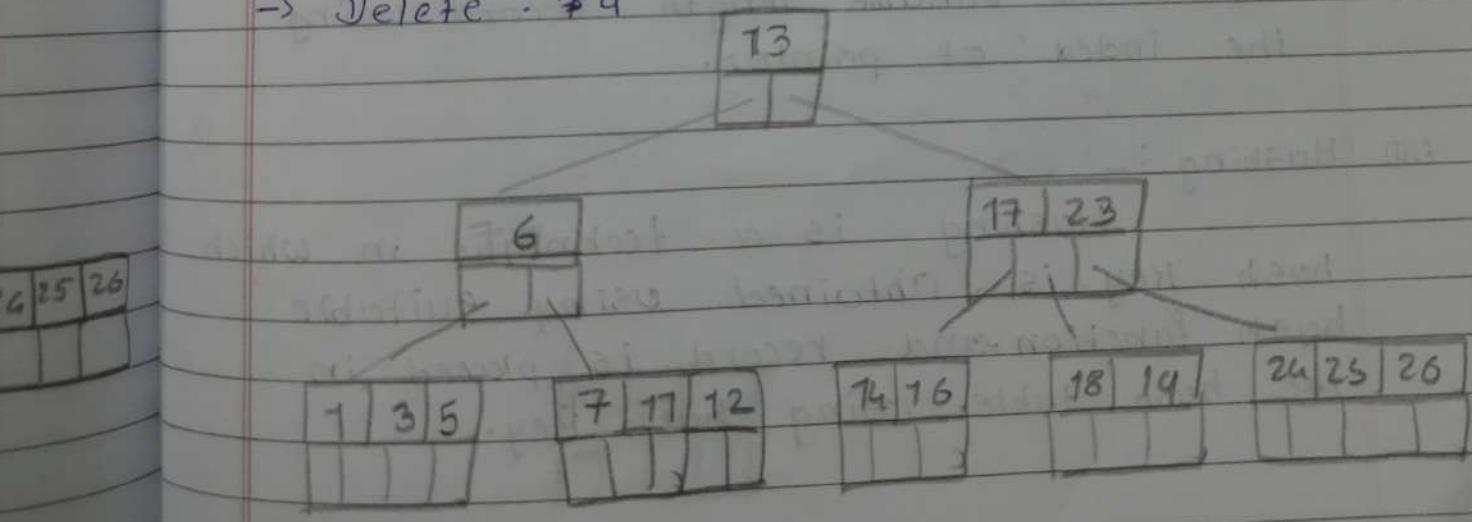
-> Delet : 8



-> Delete : 20



-> Delete : 4



Q.5

(a) Explain Random file Organization.

→ Random organization is a kind of file organization in which records are stored at random location on the disk.

→ There aree three technique used.
(i) direct addressing
(ii) Directory look up
(iii) Hashing

(i) Direct addressing :

There are two types of record are handles : fixed length record and variable length record.

- For sorting fixed length record the disk spaces divided into the nodes.
- For sorting variable length record we can locate the variable length record using the index of pointor.

(iii) Hashing :

Hashing is a technique in which hash key is obtained using suitable hash function and record is placed in the hash table using hash key.

(ii) Directorly look up :

For retrieving the desired record first of all the index for the record address is searched then using this record address the actual record is accessed.

- Disadvantage of these method is that it requires more disk access then direct address method.

- Advantage of this method is that effective disk space utilization in it compare to direct addressing method.

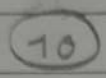
(b) Explain collision resolution techniques.

→ Winter : 2019

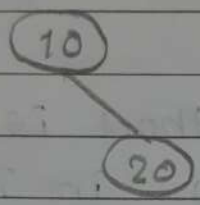
Q. 5 -> (c)

(c) Construct AVL tree.
10, 20, 30, 40, 50, 60, 70, 80

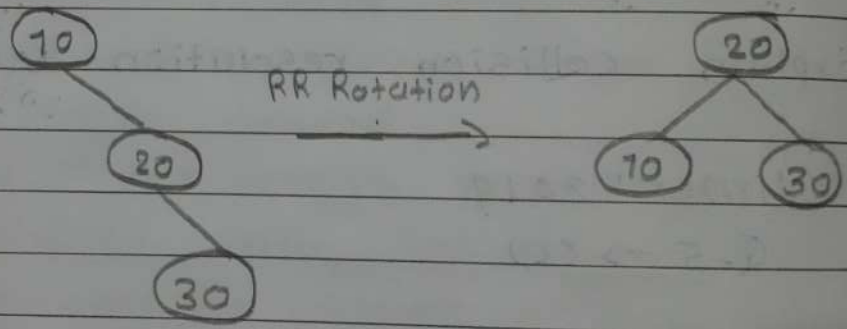
→ Step 1: insert 10



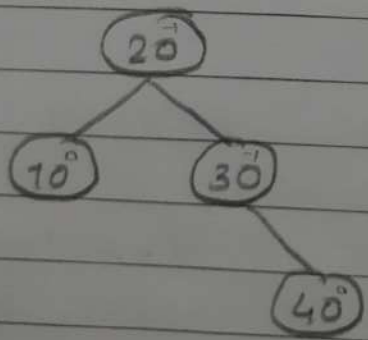
→ Step 2: insert 20



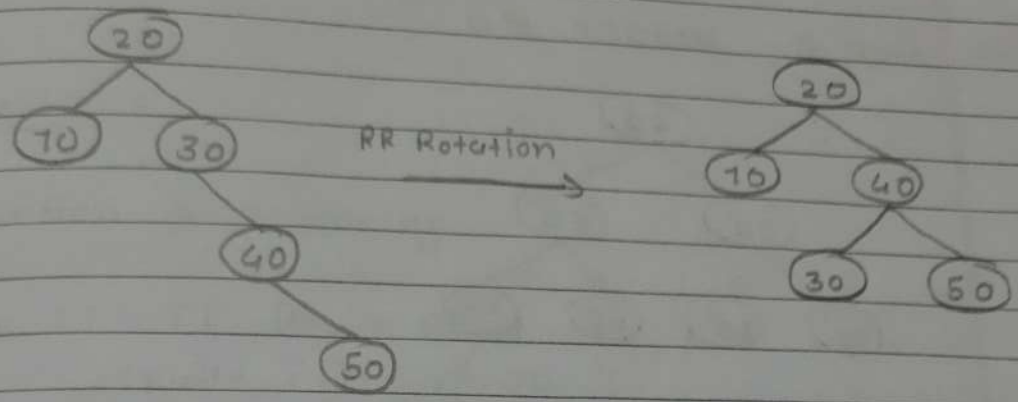
→ Step 3: insert 30



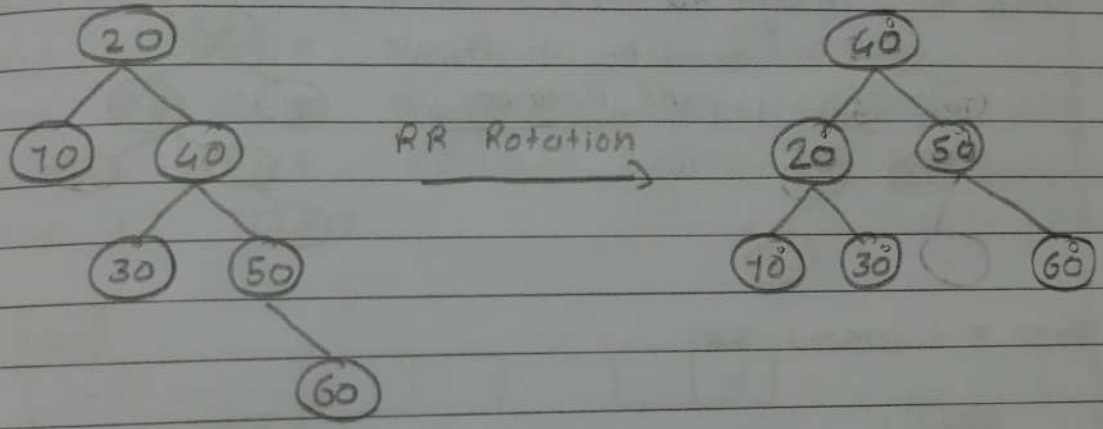
→ Step 4: insert 40



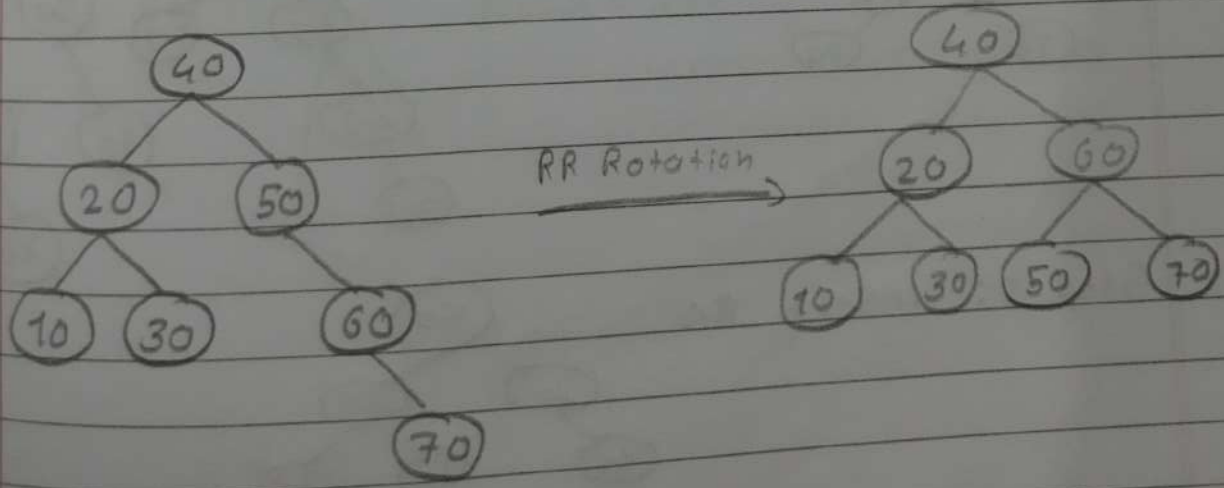
-> Step 5 : insert 50



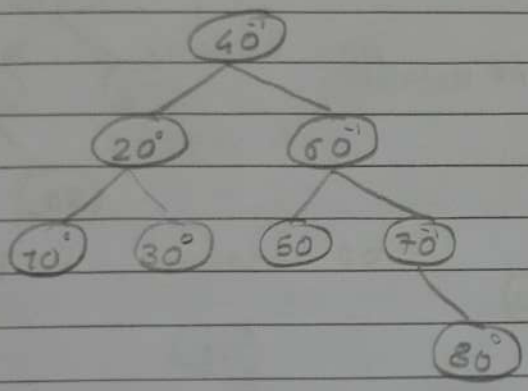
-> Step 6 : insert 60



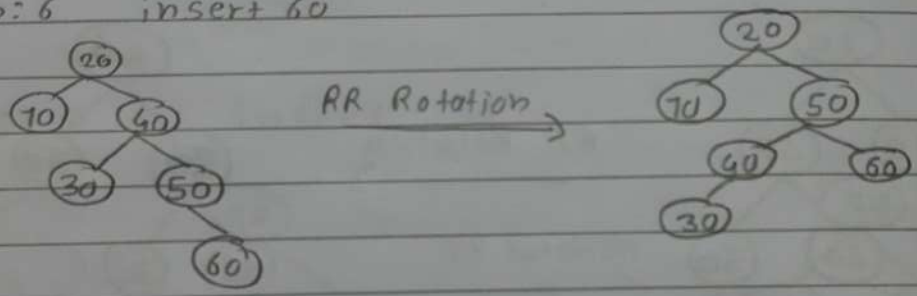
-> Step 7 : insert 70



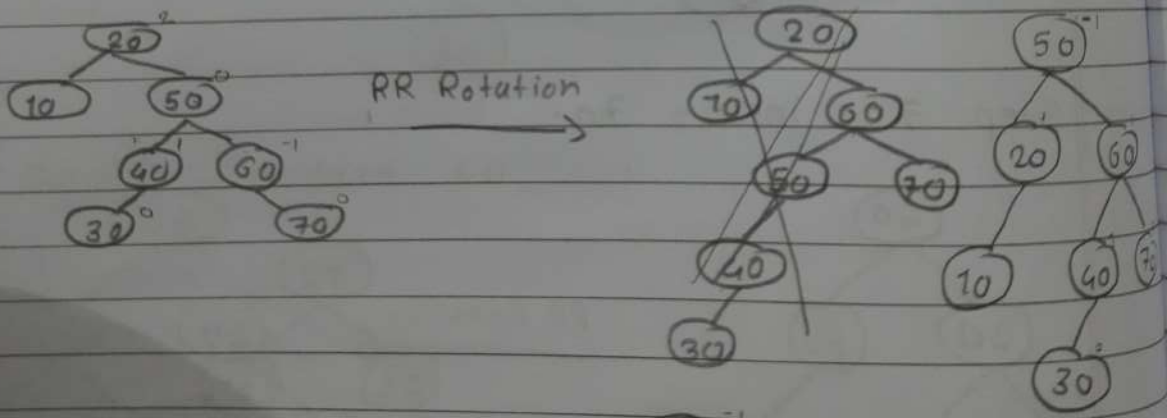
→ Step 5 : insert 80



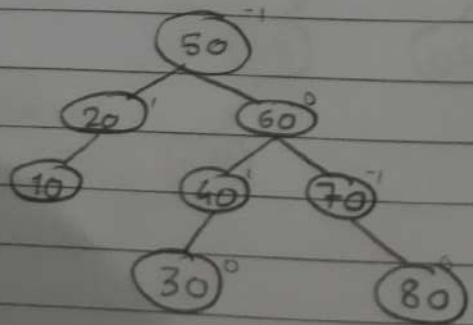
→ Step: 6 insert 60



→ Step 7 : insert 70



→ Step 8 : insert 80



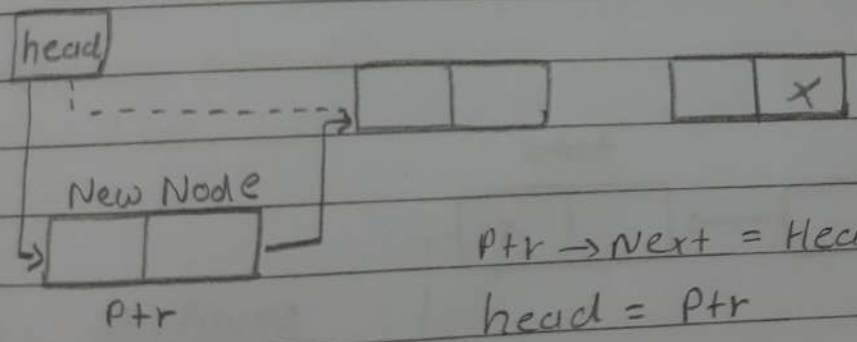
Singly linked list

* Insertion :

→ Insertion at beginning :

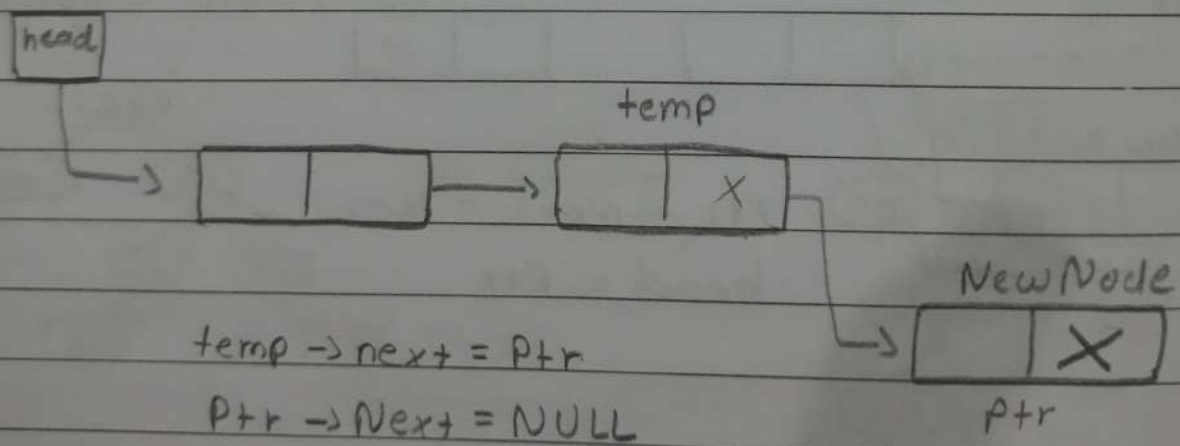
Step 1: If Ptr = NULL
write "OVERFLOW"

Step 2: SET NewNode = Ptr
Step 3: SET Ptr = Ptr → Next
Step 4: SET NewNode → Data = VAL
Step 5: SET NewNode → Next = HEAD
Step 6: SET HEAD = NewNode
Step 7: EXIT



→ Insertion at End :

Step 1: If Ptr = NULL
Write "OVERFLOW"
Exit
Step 2: SET NewNode = Ptr
Step 3: SET Ptr = Ptr -> Next
Step 4: SET NewNode -> Data = VAL
Step 5: SET NewNode -> Next = NULL
Step 6: SET Ptr Temp = head
Step 7: Repeat Step 8 while
Temp -> Next != NULL
Step 8: SET Temp = Temp -> Next
Step 9: SET Temp -> Next = NewNode
Step 10: Exit



→ Insertion at given location:

(1) If Ptr = NULL
Write "OVERFLOW"
Exit

(2) New-Node = Ptr.

(3) Ptr = Ptr -> Next.

(4) Ptr = Head.

(5) Repeat step 6 while
Ptr -> data != Num

(6) Ptr = Ptr -> next

(7) NewNode -> next = Ptr -> Next.

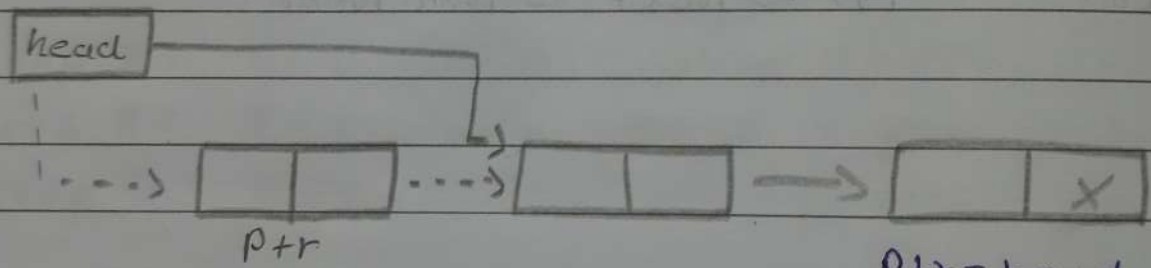
(8) Ptr. -> Next = New Node

* Deletion :

-> Deletion from beginning :

(1) If Head = NULL
Write "UNDERFLOW"
Exit

(2) PTR = Head
(3) Head = Head -> Next
(4) FREE PTR
(5) Exite.



ptr = head
head = ptr -> Next
Free (ptr)

-> Deletion from end :

(1) If Head = NULL
Write "UNDERFLOW"
Exit
(2)

-> Deletion from end:

(1) If Head = NULL
    Write "UNDERFLOW"
    Exit
(2) Ptr = Head
(3) Repeat step 4
    while , Ptr -> Next != NULL
(4) PrePtr = Ptr
(5) Ptr -> next = NULL
(6) FREE Ptr
(7) Exit.

(1) If Head = NULL
    write "UNDERFLOW"
    exit
(2) Ptr = Head
(3) Repeat step 4 and 5
    while , Ptr -> Next != NULL
(4) Preptr = Ptr
(5) Ptr = Ptr -> next
(6) PrePtr -> Next = NULL
(7) FREE Ptr
(8) Exit

→ Deletion at specific location:

(1) If Head = NULL
write "Underflow"
Exit
(2) Temp = Head
(3) Set i = 0
(4) Repeat step
until i = 0
(5) Temp 1 = Temp
(6) Temp = Temp → Next.
(7) If Temp = NULL
write "Desired iteam not found"
Exit
(8) i = i + 1
(9) ~~Temp1 = Temp~~ Temp1 → Next = Temp → next
(10) Free Temp
(11) Exit

* Doubly linked list

-> Insertion at beginning

(1) If Ptr = NULL
write "Overflow"
Exit
(2) NewNode = Ptr
(3) Ptr = Ptr -> next
(4) NewNode -> Data = VAL
(5) NewNode -> Prev = NULL
(6) NewNode -> Next = Start
(7) Head -> Prev = NewNode
(8) Head = NewNode
(9) Exit

-> Insertion at end

(1) If Ptr = NULL
write "Overflow"
Exit
(2) NewNode = Ptr
(3) Ptr = Ptr -> next
(4) NewNode -> Data = VAL
(5) NewNode -> Next = NULL
(6) Temp = Start
(7) Repeat Step 8 while Temp -> next != NULL
(8) Temp = Temp -> next
(9) Temp -> next = NewNode
(10) New Node -> Prev = Temp
(11) Exit

→ Insert at given location:

(1) If Ptr = NULL
write "Overflow"
Exit
(2) NewNode = Ptr
(3) Ptr = Ptr → Next
(4) New Node → Data = Val
(5) Temp = Head
(6) SET i = 0
(7) Repeat Step 8 and 9
(8) Temp = Temp → next
(9) Temp = NULL
write "Less than desired no."
(10) NeNode → next = Temp → next
(11) NewNode → Prev = Temp
(12) Temp → next = NewNode
(13) Temp → next → prev = NewNode
(14) Exit

* Circular linked list

-> Insertion at beginning

(1) If Ptr = NULL
write "Overflow"
Exit
(2) NewNode = Ptr
(3) Ptr = Ptr -> next
(4) NewNode -> Data = VAL
(5) Temp = Head
(6) Repeat Step 7 while Temp->next != Head
(7) Temp = Temp ->next
(8) New Node -> next = Head
(9) Temp-> next = New Node
(10) Head = New Node
(11) Exit.

-> Insertion at End

(1) If Ptr = NULL
write "Overflow" Exit
(2) NewNode = Ptr
(3) Ptr = Ptr -> Next
(4) New Node -> Data = VAL
(5) New Node -> next = Head
(6) Temp = Head
(7) Repeat Stp 8 while Temp ->next != Head
(8) Temp = Temp ->next
(9) Temp -> next = NewNode
(10) Exit

→ Deletion at beginning:

(1) If Head = NULL
write "Underflow"
Exit
(2) Ptr = Head
(3) Repeat step 4 while Ptr->next != NULL
(4) Ptr = Ptr->next
(5) Ptr->next = Head->next
(6) Free Head
(7) Head = Ptr->next
(8) Exit

→ Deletion at End:

(1) If Head = NULL
write "Underflow"
Exit
(2) Ptr = Head
(3) Repeat step 4 and 5
while Ptr->next != Head
(4) PrePtr = Ptr
(5) Ptr = Ptr->next
(6) PrePtr->Next = Head
(7) Free Ptr
(8) Exit.

Infinix to Postfix Conversion

* $A + B * C - D / E$

-> $A + (BC*) - D/E$ $[x_1 = (BC*)]$

$A + x_1 - (DE/)$ $[x_2 = (DE/)]$

$A + x_1 - x_2$

$(A + x_1) - x_2$ $[x_3 = (A + x_1)]$

$x_3 - x_2$

$x_3 x_2 -$

$A * x_1 + x_2 -$

$ABC*+DE/-$

(1) $, ↑, ^$

(2) $*, /$

(3) $+, -$

=) Using stack.

Symbol	Stack	Answere.
A	[ ]	A.
+	[+]	A.
B	[+]	AB
*	[*, +]	AB
C	[*, +]	ABC.
-	[-]	ABc*+
D	[-]	ABc*+D
/	[-, /]	ABc*+D
E	[-, /]	ABc*+DE ABC*+DE/-

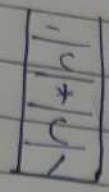
Infix to Prefix

→ (a+b-c)*d^e^f)/g

Reverse the expression :
g/(f^e^d*(c-b+a))

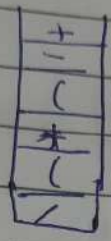
Symbol	Stack	Ans
g	/	g
/	/	g
(	(/	g
f	(/	gf
^	^ (/	gf
e	^ (/	gfe
^	^ (/	gfe^
d	^ (/	gfe^d
*	* (/	gfe^d^
(	(* (/	gfe^d^
c	(* (/	gfe^d^c
-	- (* (/	gfe^d^c

b



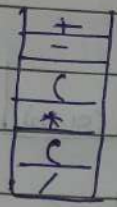
gfe^d^cb

+



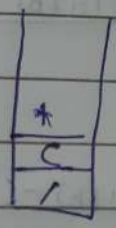
gfe^d^cb

a



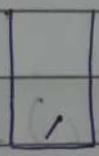
gfe^d^cba

)



gfe^d^cba+-

)



gfe^d^cba+-*

None

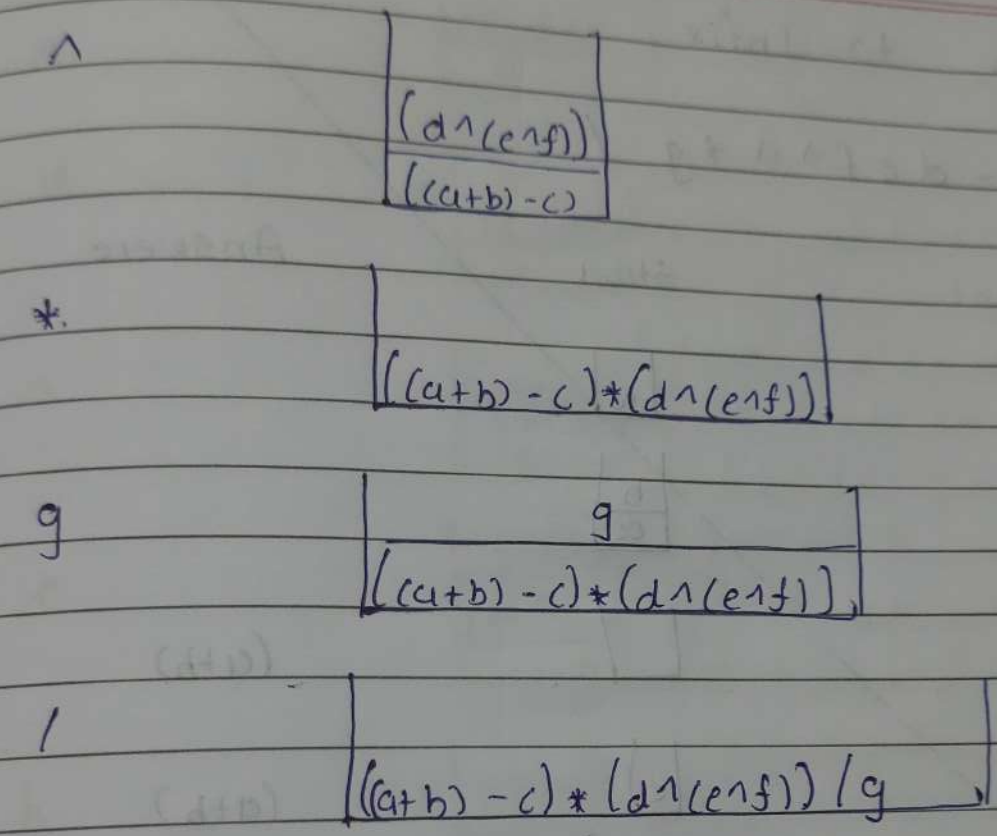
gfe^d^cba+-*/

Reverse the expression = /*-+abc^d^efg

☆ Postfix to Infix

-> a b + c - d e f ^ ^ * g /

Symbol	Stack
a	a
b	b / a
+	(a+b)
c	c / (a+b)
-	((a+b)-c)
d	d / ((a+b)-c)
e	e / d / ((a+b)-c)
f	f / e / d / ((a+b)-c)
^	e^f / d / ((a+b)-c)

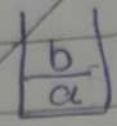

^
(d^(e-f))
((a+b)-c)
*
((a+b)-c)*(d^(e-f))
g
g
((a+b)-c)*(d^(e-f))
/
((a+b)-c)*(d^(e-f))/g

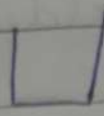
b

+

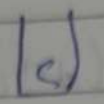
c

-





(a+b)



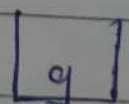
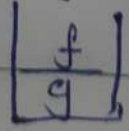
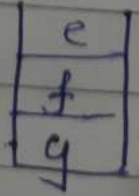
(a+b)

☆ Prefix to Infix

→ / * - + a b c ^ d ^ e f g

Reverse the expression

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

Symbol	Stack
g	
f	
e	

^

(e^f)
g

d

d
(e^f)
g

^

(d^(e^f))
g

c

c
(d^(e^f))
g

b

b
c
(d^(e^f))
g

a

a
b
c
(d^(e^f))
g

+

(a+b)
c
(d^(e^f))
g

−

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

*

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

/

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

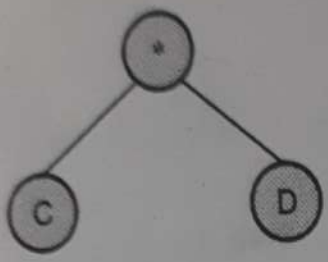
☆ Evalute Postfix Expression

-> +*AB-C+C*BA [where, A=4, B=8, C=12]

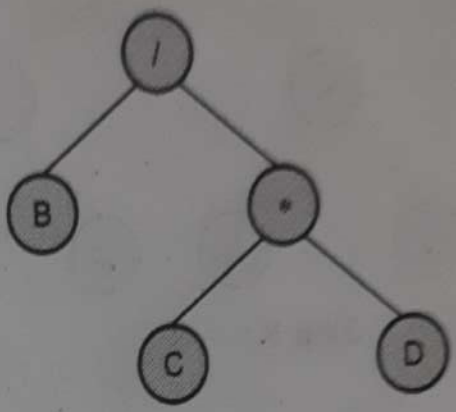
= +(A*B)-C+C(B*A)
= +(32)-C+C(32)
= +(32)-C(+C32)
= +(32)-C(C+32)
= +(32)-C(44)
= +(32)+(-C44)
= +(32)(C-44)
= +(32)(32)
= (32+32)
= 64

Ans. : i) A+B/C*D–E

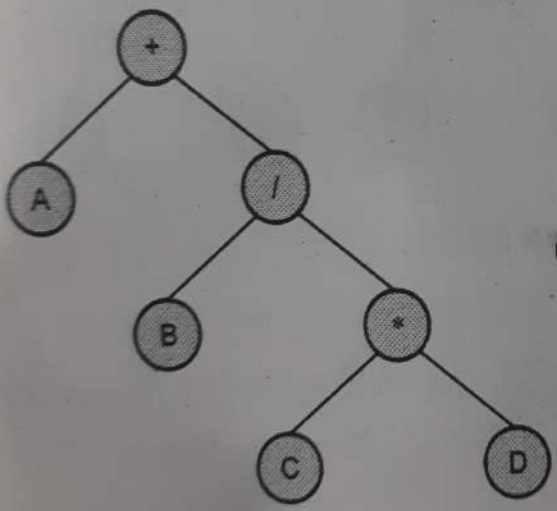
Step 1 :



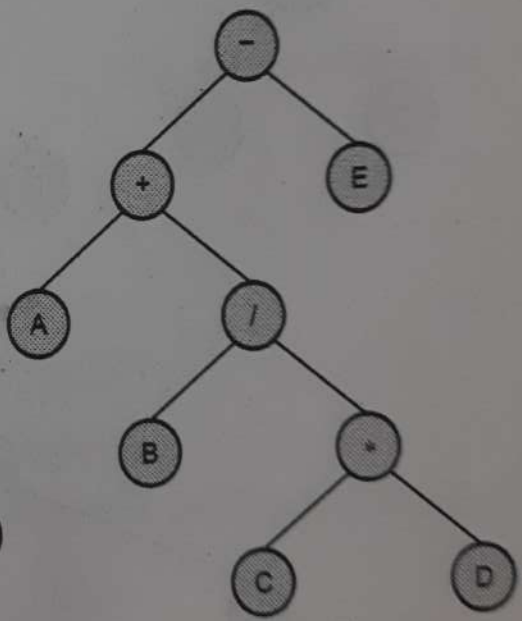
Step 2 :



Step 3 :

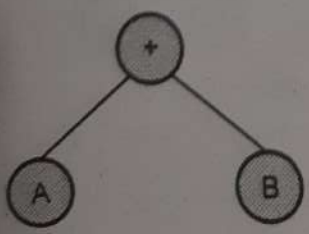


Step 4 :

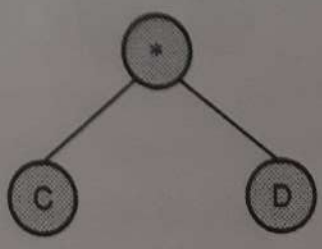


ii) ((A+B)–(C*D))%((E^F)/(G – H))

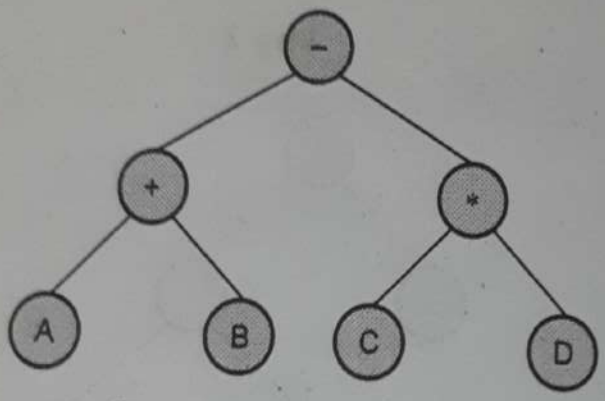
Step 1 :



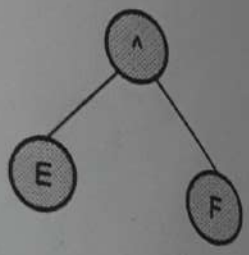
Step 2 :



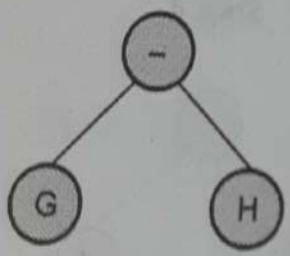
Step 3 :



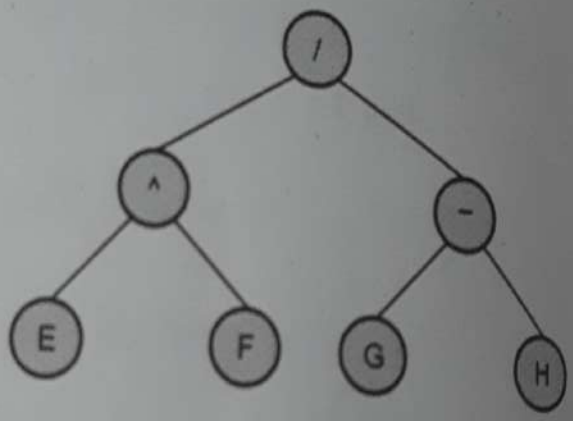
Step 4 :



Step 5 :



Step 6 :



Step 7 :