DSU PT-2 Question Bank.

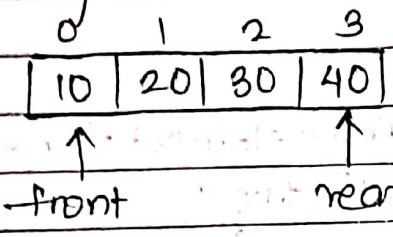
PAGE NO.
DATE / /

Q.1. Describe queue full & queue empty operation condition on linear queue with suitable diagram.

→ ① Queue full:-

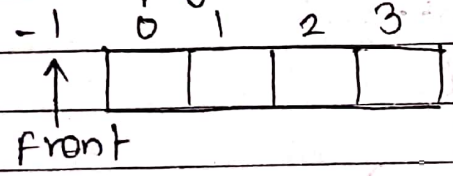
A queue is full when its rear pointer points to max-1 position. Max is maximum number of elements in queue. If rear pointer is not equal to max-1 then a new element can be added to a queue. If queue is full then new element cannot be added to a queue.

Example:- Consider max=4. First element is stored at 0 position & last element is stored at 3 positions in queue. In the diagram given below rear pointer is pointing to max-1(3) position so queue is full.



② Queue empty:- A queue is empty when its front pointer points to -1 position. When front pointer is -1 the one cannot delete any data from a queue.

Example:- In the diagram given below front pointer points to -1 value i.e., it points no location inside queue so queue is empty.



Q.2. C program to insert an element into the queue and delete an element from the queue.

→

```
#include<stdio.h>
#include<conio.h>
#define max 5
```

```
void main()
{
 int a[max],front,rear,no,ch,i;
 clrscr();
 front=rear=-1;
 do
 {
  printf("\n1. Insert");
  printf("\n2. Delete");
  printf("\n3. Exit");
  printf("\n Enter your choice:");
  scanf("%d",&ch);
  switch(ch)
  {
   case 1: print("\n Enter element to be inserted:");
           scanf("%d",&no);
           if(rear==max-1)
           {
            printf("\nqueue is full");
            break;
           }
           rear=rear+1;
           a[rear]=no;
           if(front==-1)
           front=0;
           break;
   case2: if(front==-1)
           {
            printf("\nQueue is empty");
            break;
           }
           no=a[front];
```

```
printf("\ndeleted element is:%d", no);
if(front==rear)
front=rear=-1;
else
front=front+1;
break;
case3:exit(0);
}
printf("\n Do you want to continue:(1 for yes/2 for n
scanf("%d",&ch);
}
while(ch==1)
getch();
}
```

Q.3. Describe terms. node, information, pointer, empty li

→ (1) Node:- The structure combination of data & its addre to the next structure is called as node.

(2) Information:- The data stored in data field of the linked list structure.

(3) Pointer:- A datatype used to point to the address of the next node.

(4) Empty List:- It signifies no data in the linked list.

Q.4. Describe w.r.t tree leaf node, level of node, root, sibling binary tree, binary search tree.

→ (1) Leaf node:- A node of 0 degree is also known as leaf node. A leaf node is a terminal node an it has no children.

(2) Level of node:- The level of a node is defined as follows:

(1) The root of the tree has level 0.

(2) The level of any other node in a tree is one more than the level of its father.

(3) Root :- It is a special node in a tree structure & the entire tree is referenced through it. This node does not have a parent.

(4) Sibling :- Nodes with the same parent are siblings.

(5) Binary tree :- A tree is binary if each node of the tree can have maximum of two children. Moreover, children of a node of binary tree are ordered. One child is called the "left" child & the other is called the "right" child.

(6) Binary Search tree :- A binary search tree is a binary tree, which is either empty or in which each node contains a key that satisfies the following conditions :-

(i) All keys are distinct.

(ii) for every node, X. in the tree, values of all the keys in its left subtree are smaller than the key value in X.

(iii) for every node, X. in the tree, values of all the keys in its right subtree are larger than the key value in X.

Q. 5. Define w.r.t. graph in-degree, out-degree.

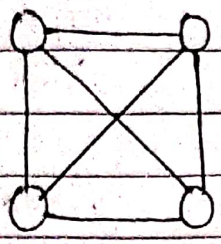
→ (1) In-degree :- It is number of edges coming towards a specified node. i.e number of edges that have that specified node as the head is known as indegree of a node.

(2) Out-degree :- It is number of edges going out from a specified node i.e. number of edges that have that specified node as the tail is known as outdegree of a node.

Q.6. Explain complete graph with diagram.

→ An undirected graph, in which every vertex has an edge to all other vertices is called as complete graph. A complete graph with N vertices has N(N-1)/2 edges.

eg:-



Q.7. Write operations on linked list & trees.

→ (I) Operations on linked list:-

- → Creating a linked list
- → Inserting an item or data
- → Deleting an item or data.
- → Traversing a linked list.
- → Sorting an element in a linked list.
- → Counting the nodes.
- → Merging of two sorted linked list.
- → Searching an item in a linked list.

(II) Operations on trees:-

- → Initialise
- → Find
- → makeempty
- → Insert
- → Delete
- → Create
- → findmin
- → findmax.

Q.8. Difference betwn array & linked list.

→

	Array	Linked list
①	Array is a collection of elements of similar data type.	Linked list is an ordered collection of elements of same type, which are connected to each other using pointers.
②	Array gets memory allocated in the stack section.	Linked list gets memory allocated in Heap section.
③	Size of the array must be specified at time of array declaration.	Size of the linked list is variable. It grows at runtime, as more nodes are added to it.
④	Array can be single dimensional, two dimensional or multidimensional	Linked list can be linear (singly), doubly or circular linked list.

Q.9. Write an algorithm to search a particular node in the given linked list.

→ Node contains two fields: info & next pointer.
start pointer: Header node that stores address of first node.

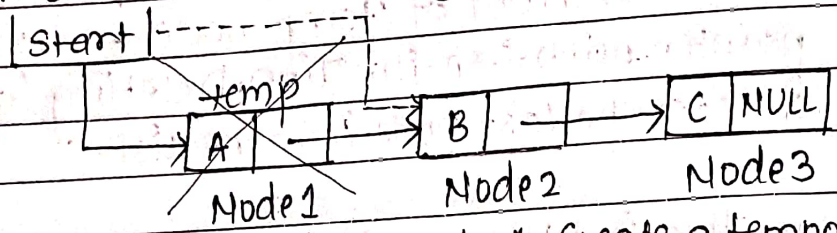
Step01:- Start
Step02:- Declare variable no, flag and pointer temp.
Step03:- Input search element.
Step04:- Initialize pointer temp with the address from start pointer (temp=start), flag with 0.
Step05:- Repeat step06 till temp != NULL.
Step06:- compare: temp->info = no then
set flag=1 and go to step07
otherwise
increment pointer temp & goto step05

DATE / /

Step07:- compare: flag=1 then
~~set flag=1~~ display "Node found"
otherwise
display "Node not found"
Step08:- stop

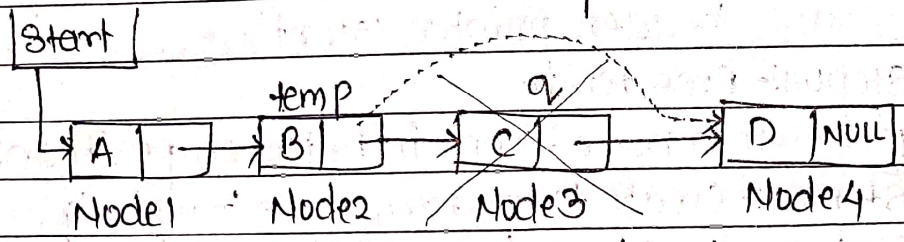
Q10 Show with suitable diagram for deleting a node in SLL at beginning, in between and end.

→ ① Delete a node from the beginning:-



Node to be deleted is node 1. Create a temporary node as 'temp'. Set 'temp' node with the address of first node. Store address of node 2 in header pointer 'start' and then delete 'temp' pointer with free function. Deleting temp pointer deletes the first node from the list.

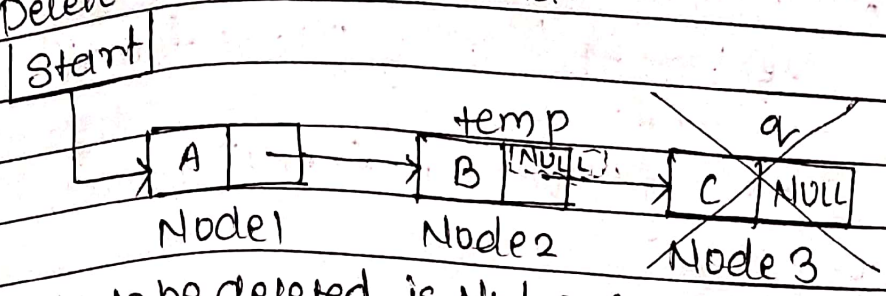
② Delete a node from inbetwn position:-



Node to be deleted is node 3. Create a temporary node as 'temp' & 'q'. Set 'temp' node with the address of first node. Traverse the list upto the previous node of node 3 & mark the next node(node 3) as 'q'. Store address from node 'q' into address field of 'temp' node. Then delete 'q' pointer with free function. Deleting 'q' pointer deletes the node 3 from the list

PAGE No.
DATE / /

③ Delete a node from the end:-



Node to be deleted is Node3. Create a temporary node as 'temp' and 'q'. Set 'temp' node with the address of first node. Traverse the list up to the second last node and mark the last node as 'q'. Store NULL value in address field of 'temp' node and then delete 'q' pointer with free function. Deleting 'q' pointer deletes the last node from the list.

Q.11: Algorithm to delete a node at beginning, in between and at the end.

→ (I) Delete a node at beginning:-

Step01:- Create temporary node 'temp'.
Step02:- Assign address of first node to 'temp' pointer.
Step03:- Store address of second node (temp->next) in header pointer 'start'.
Step04:- free temp.

(II) Delete a node from in between position:-

Step 01:- Create temporary node 'temp', 'q',
Step 02:- Assign address of first node to 'temp' pointer.
Step 03:- Traverse list up to previous node of node to be deleted.
Step 04:- Mark the node to be deleted 'q'
Step 05:- Store address from node 'q' in address field of 'temp' node (temp->next = q->next)

(III) Delete the node from the end:-

Step01:- Create temporary node 'temp' 'q'.
Step02:- Assign address of first node to 'temp' pointer.
Step03:- Traverse list upto second last node.
Step04:- Mark last node's address in node 'q'.
Step05:- Store NULL value in address field of second last node (temp->next)
Step06:- Free q.

Q.12. Algorithm to count nodes in linked list.

→ for example, the function should return 5 for linked list 1->3->1->2->1.

Algorithm: Using iterative solution.

1) Initialise count as 0.
2) Initialize a node pointer, current = head.
3) Do following while current is not NULL.
   a) current = current->next
   b) count++;
4) Return count.

Q.13. Algorithm to insert an element at the beginning and end of linked list.

→ (I) At the beginning:-

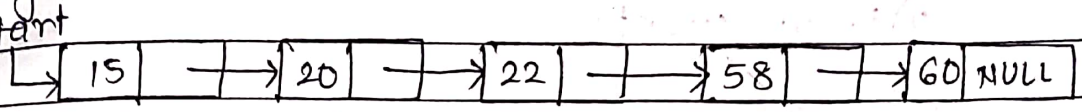
1) Start
2) Create the node pointer *temp Struct node * temp
3) Allocate address to temp using malloc temp = malloc (sizeof(struct node));
4) Check whether temp is null, if null then Display "Overflow"
   Else temp->info=data
   temp-> next= start.
5) start = temp
6) Stop.

(II) At the end:-

1) Start
2) Create two node pointers *temp, *q. struct node *temp, *q;
3) q = start
4) Allocate address to temp using malloc temp = malloc (sizeof(struct node));
5) Check whether temp is null, if null the display "Overflow"
else
temp->info = data
temp->next = null
6) while(q->next != null) q = q->next
7) q->next = temp
8) Stop

Q.14. Create a singly linked list using data fields 15, 20, 22, 58, 60. Search a node 22 from the SLL and show procedure step-by-step with the help of diagram from start to end.

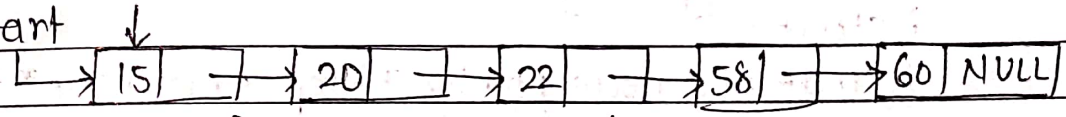
→ Original list:-



Searching a node:-

step01:- compare 22 with 15

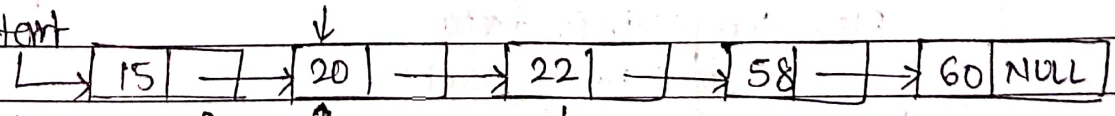
22 != 15



Node not found, move next.

Step02:- Compare 22 with 20

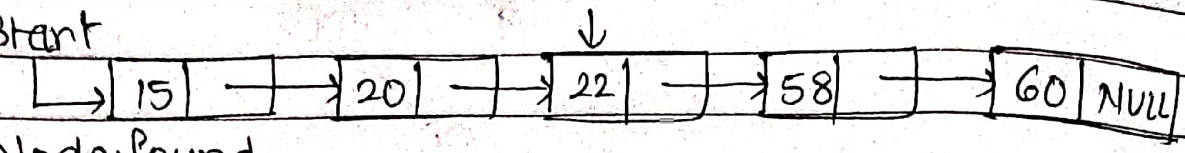
22 != 20



Node not found, move next.

Step03:- compare 22 with 22

22 = 22



Node found.

Q.15. Difference betwn trees and graphs.

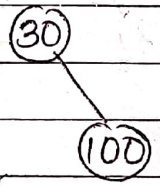
→

	Trees	Graphs.
(1)	Tree is special form of graph i.e. minimally connected graph and having only one path between any two vertices.	In graph there can be more than one path i.e. graph can have unidirectional or bidirectional paths (edges) between nodes.
(2)	Tree is a special case of graph having no loops, no circuits and no self-loops.	Graphs can have loops, circuits as well as can have self-loops.
(3)	Tree always have n-1 edges.	In graph number of edges depends on graph.
(4)	Tree is a hierarchical model.	Graph is a network model.

Q.16. Construct BST 30, 100, 90, 15, 2, 25, 36, 72, 78, 10.
Show step-by-step procedure of construction.

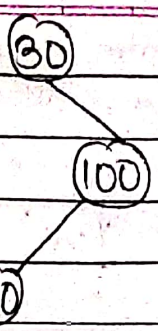
→ Step 01:-

Step 02:-

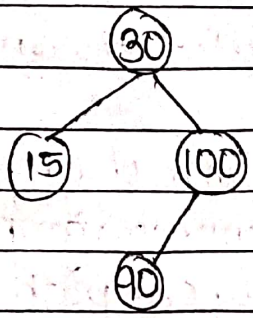


PAGE No
DATE / /

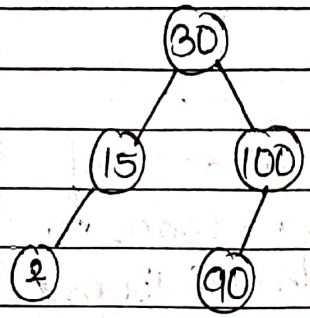
Step03:-



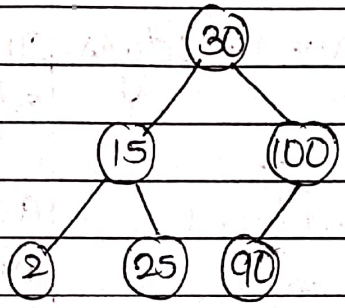
Step04:-



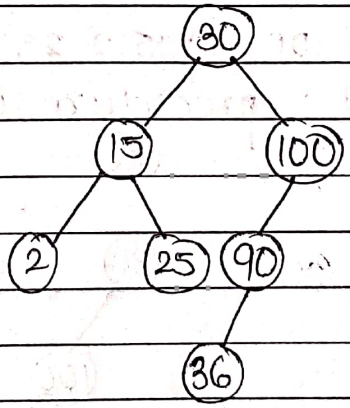
Step05:-



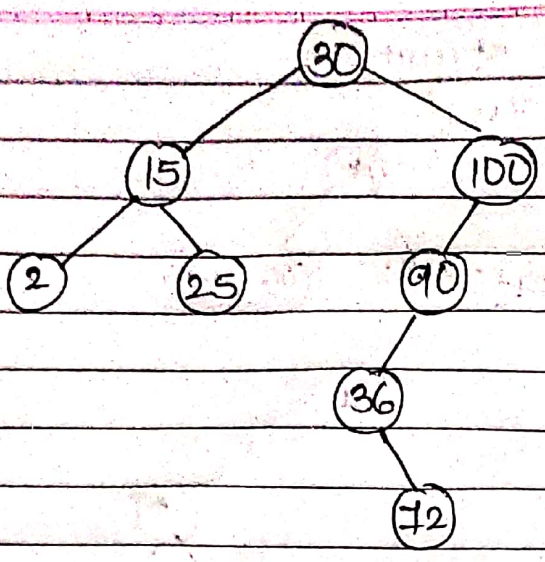
Step06:-



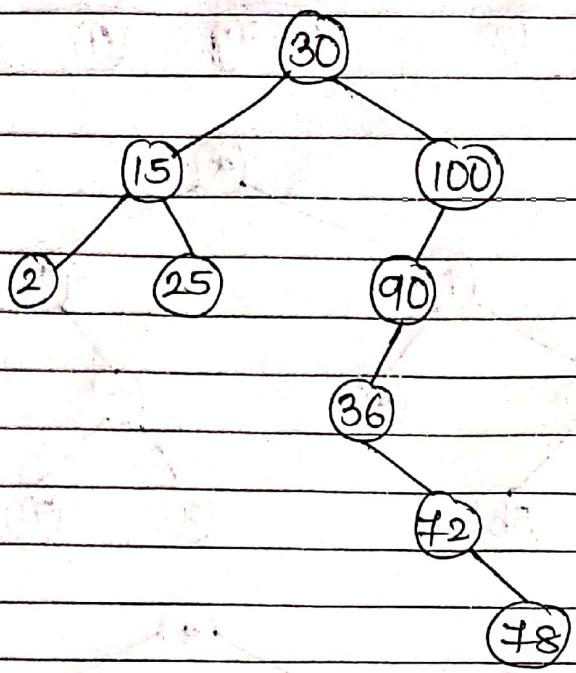
Step07:-



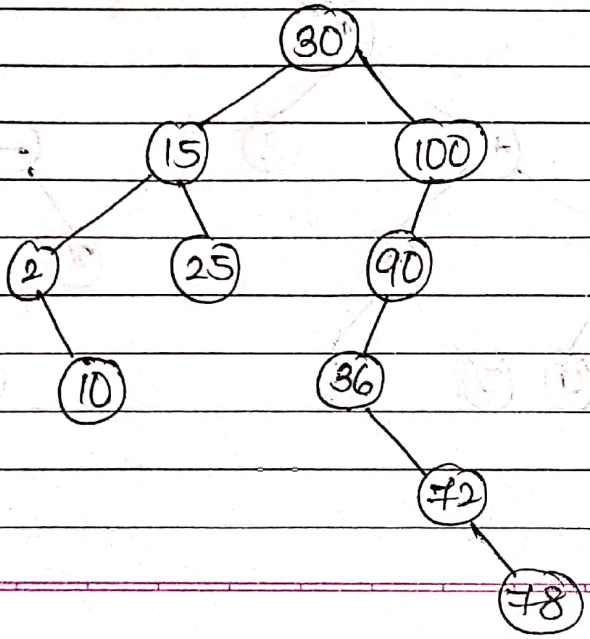
Step08:-



Step09:-



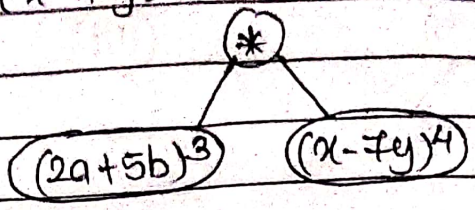
Step10:-



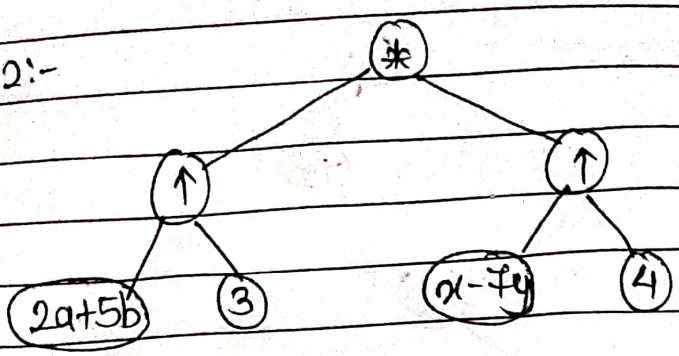
Q.17 Draw the tree structure.

(1) $(2a+5b)^3 * (x-7y)^4$

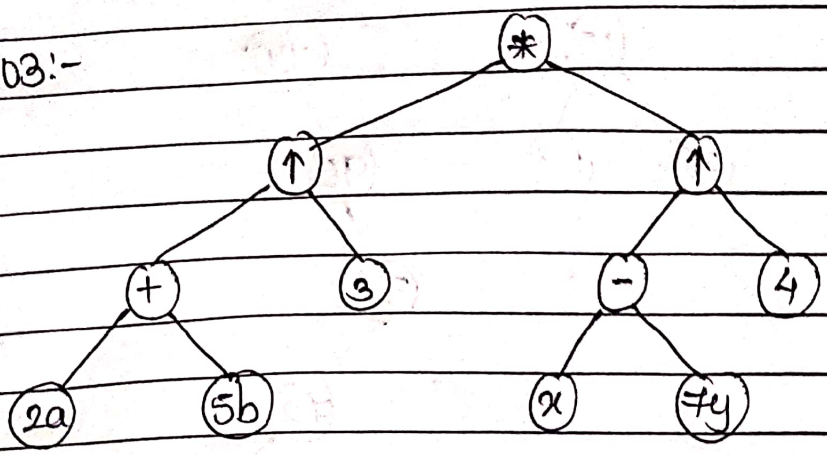
→ Step01:-



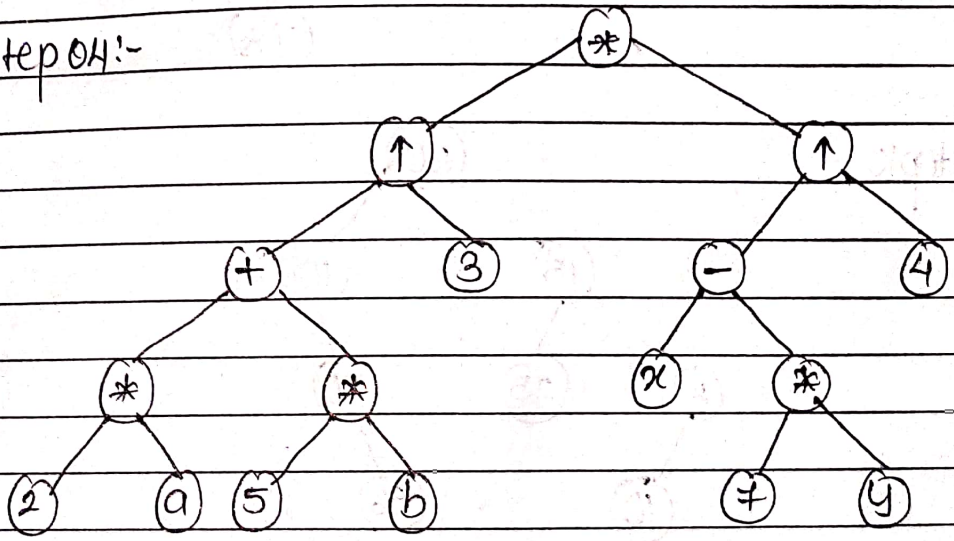
Step02:-



Step03:-

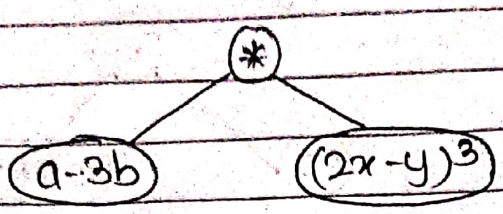


Step04:-

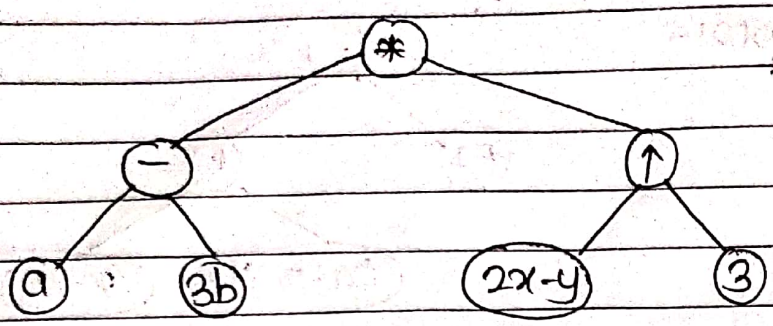


(2) $(a-3b)(2x-y)^3$

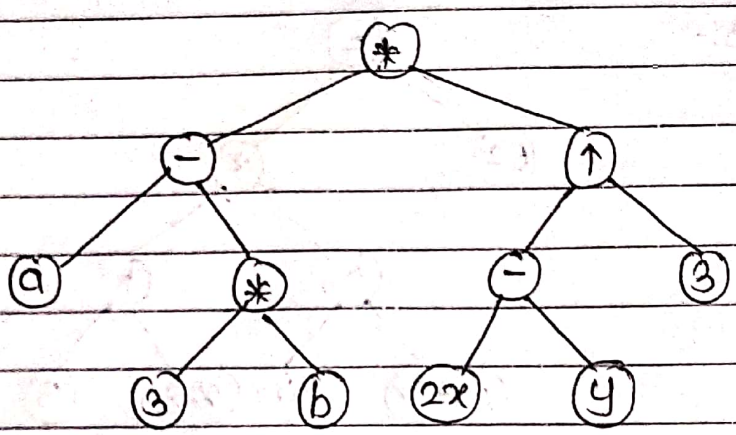
⟶ Step01:-



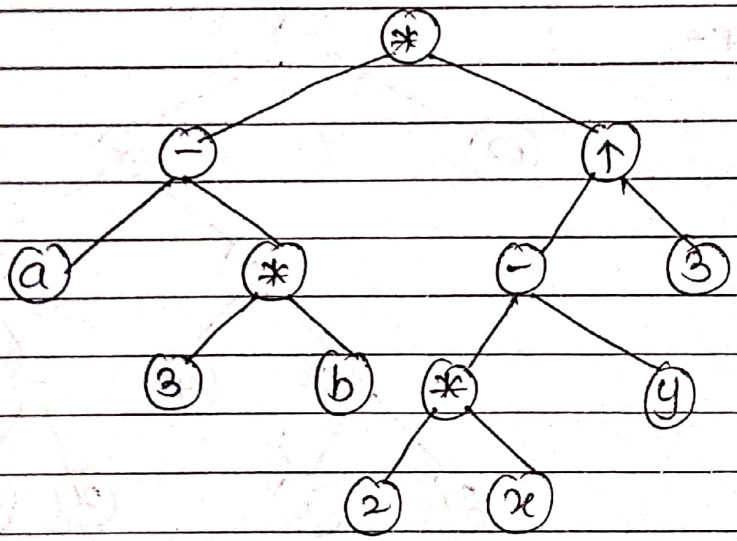
Step02:-



Step03:-

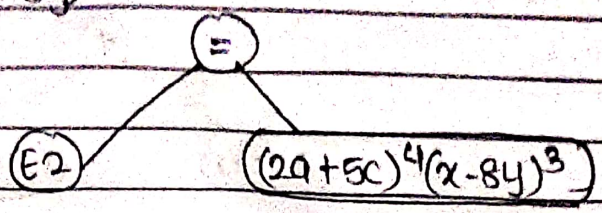


Step04:-

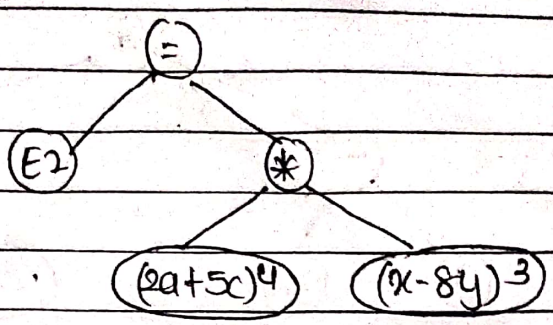


(3) $E2 = (2a+5c)^4 (x-8y)^3$

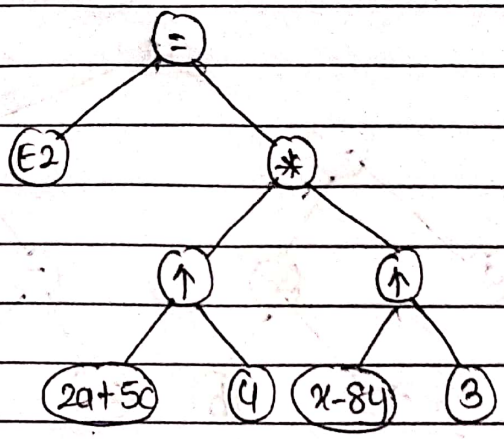
→ Step01:-



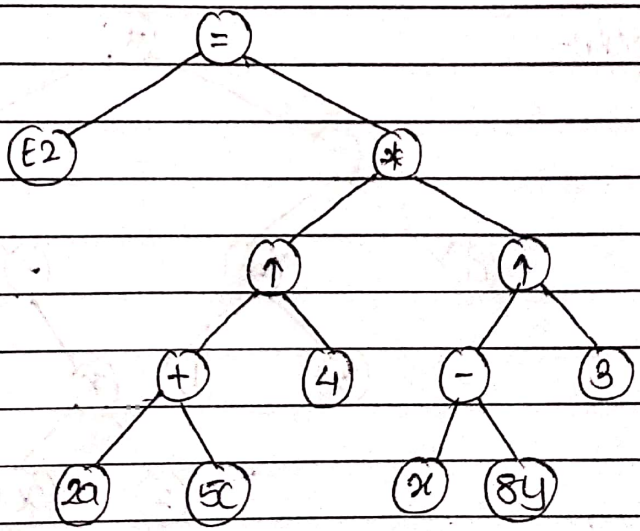
Step02:-



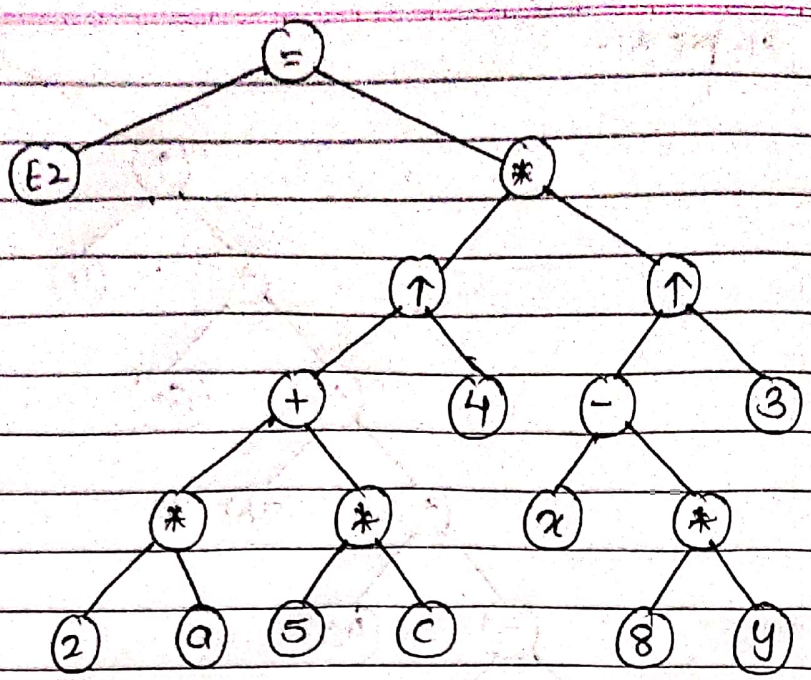
Step03:-



Step04:-

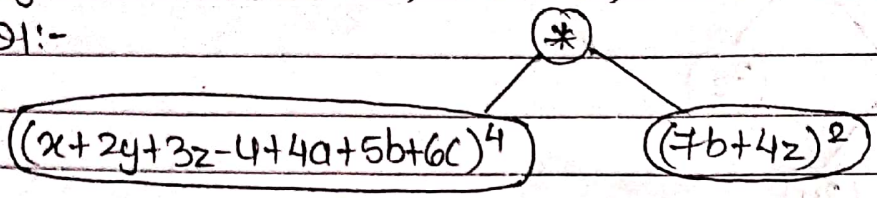


Step06:-

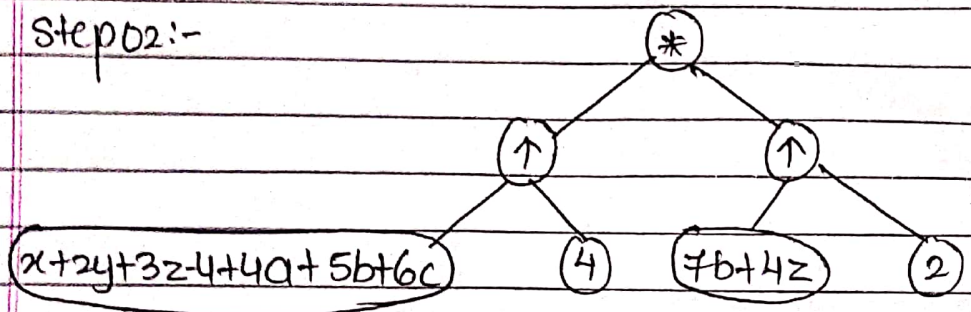


(4) $(x+2y+3z-4+4a+5b+6c)^4(7b+4z)^2$

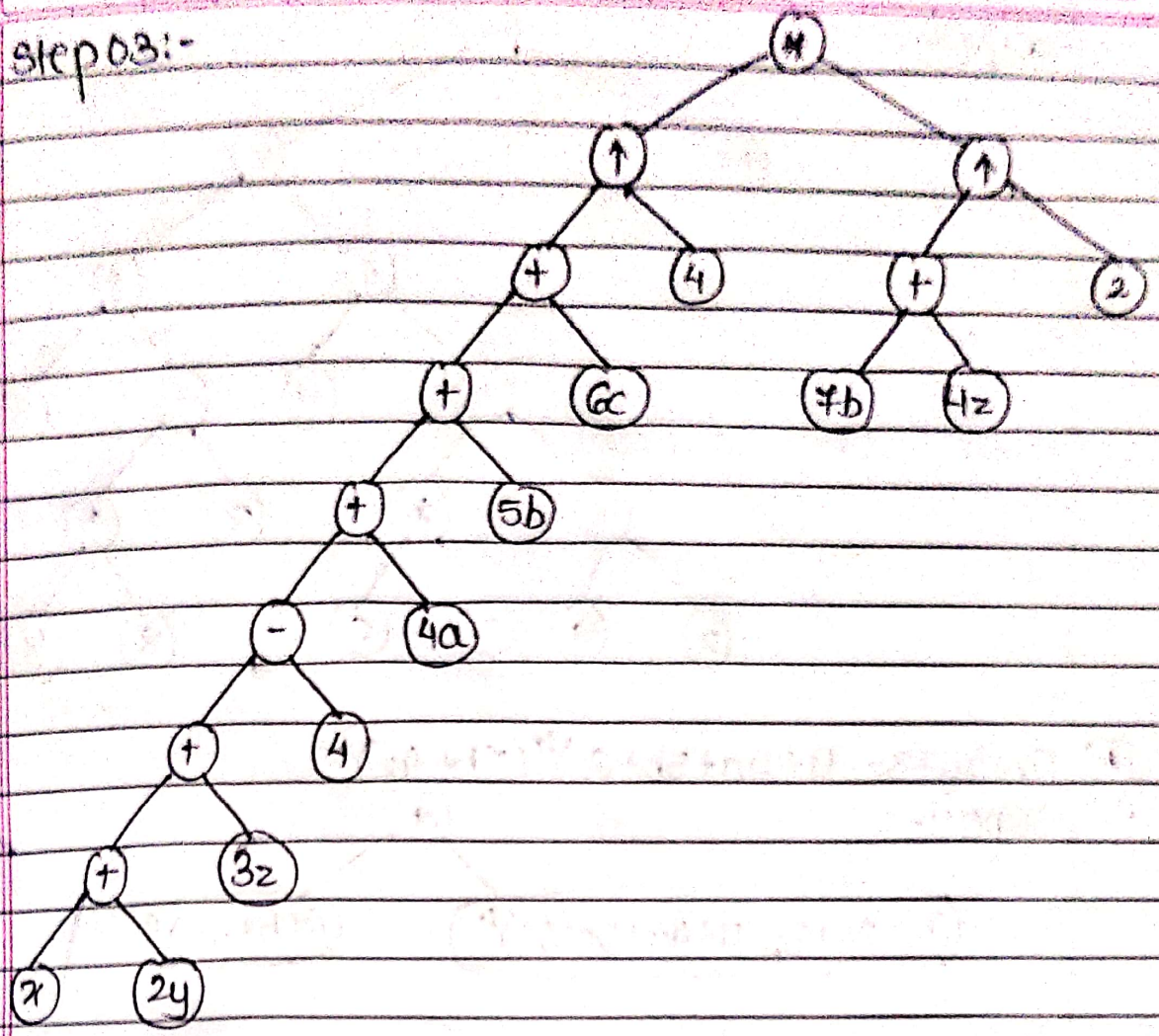
⟶ Step01:-



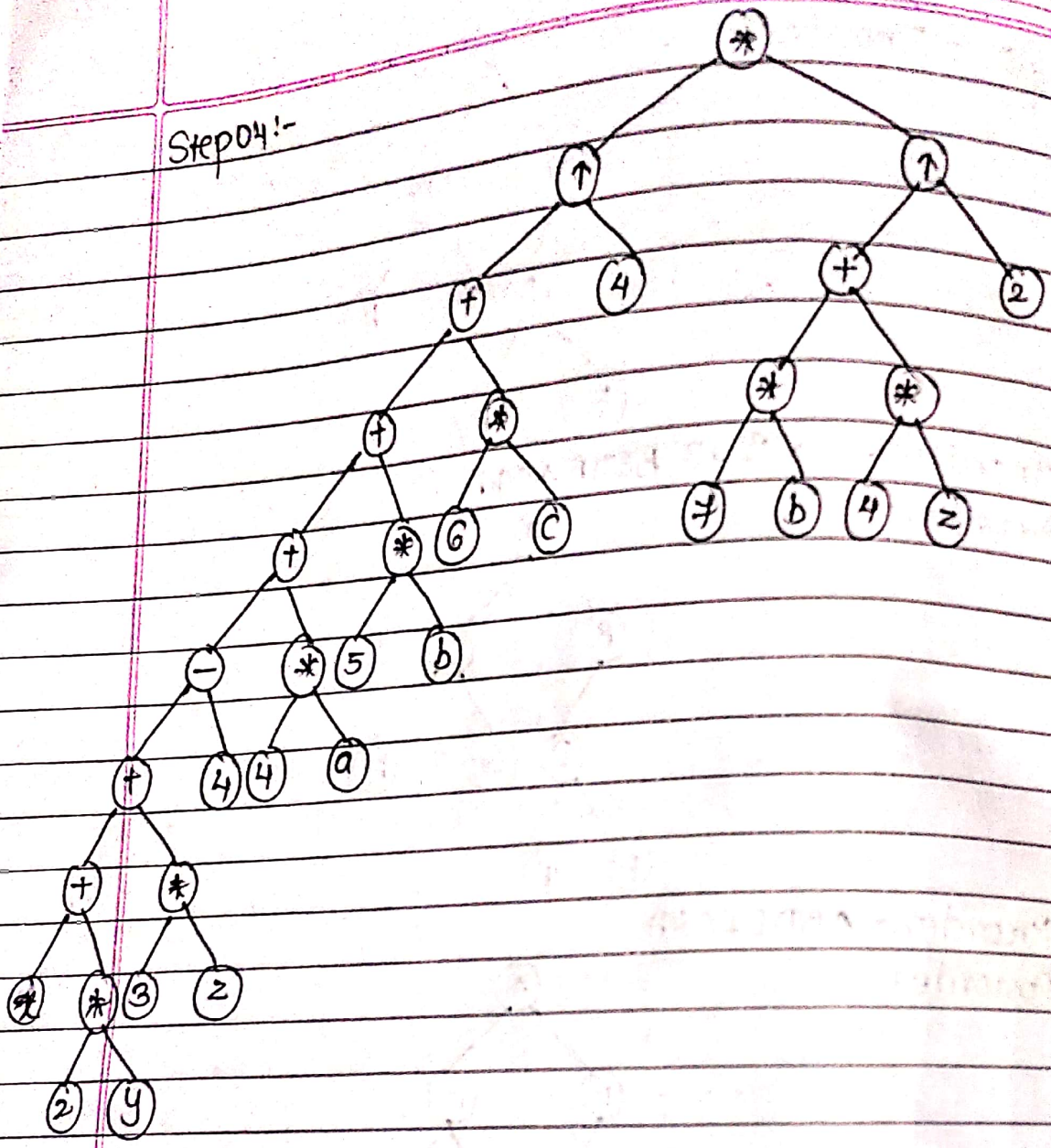
Step02:-



Step 03:-

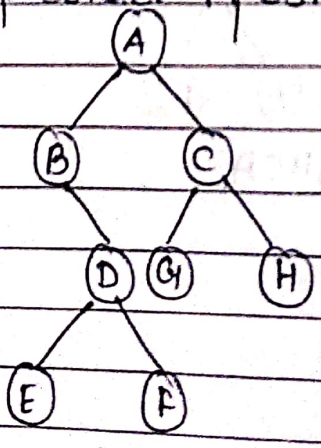


Step 04:-



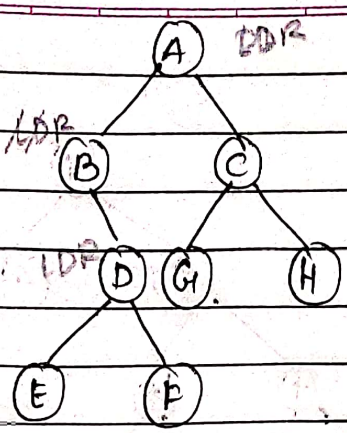
Q.18. write inorder, preorder & postorder.

(1)



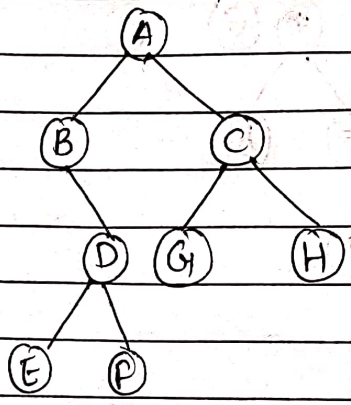
PAGE No.
DATE / /

→ ~~Step01~~ :- Inorder :-



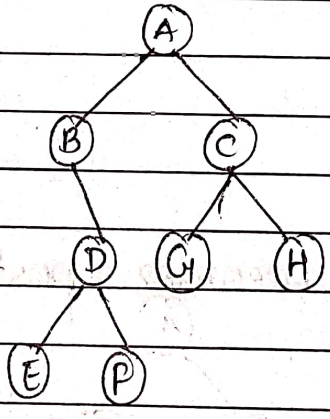
Inorder → ~~BDE~~ BEDFACGH

Preorder :-



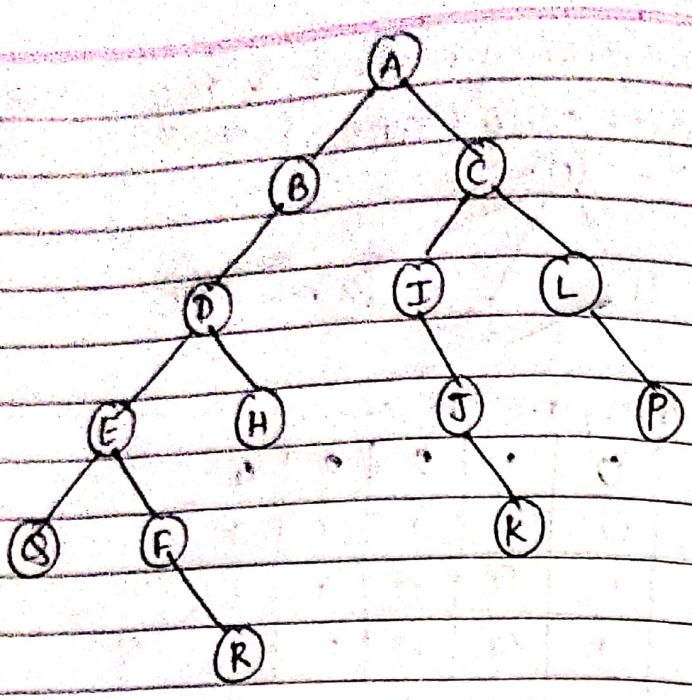
Preorder :- ABDEFCGH

Postorder :-



Postorder :- EFDBGHCA

(2)

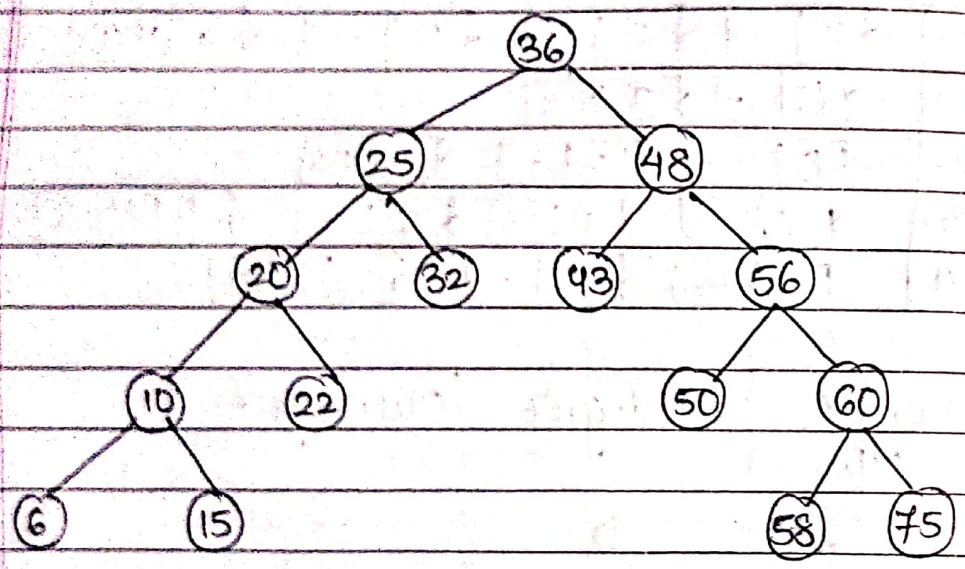


→ Inorder (LDR) :- ~~BGEFRDHDA~~ GEFRDHBAIJKCLP

Preorder (DLR):- ABDEGFRHCIJKLP

Postorder (LRD):- GREFHDBKJIPLCA

(3)



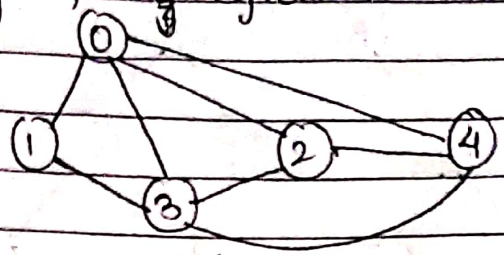
Inorder :-(LDR) :-6,10,15,20,22,25,32,36,58,60,75,50,56,43,48

(DLR) Preorder :- 36,25,20,10,6,15,22,32,48,43,56,50,60,58,75.

(LRD) Postorder:-6,15,10,22,20,32,25,58,75,50,60,43,56,48,36

Q.19. Write adjacency matrix & adjacency list representation & indegree, outdegree of each vertex.

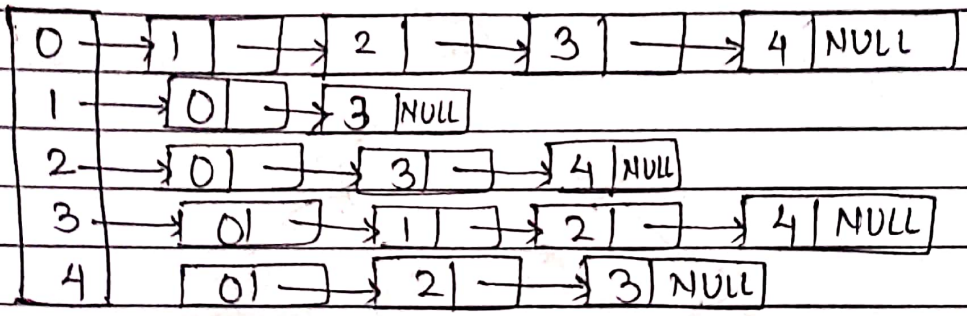
(1)



→ Adjacency matrix:-

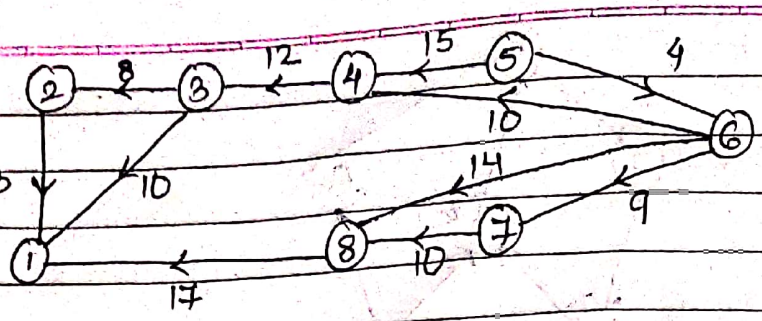
	0	1	2	3	4
0	0	1	1	1	1
1	1	0	0	1	0
2	1	0	0	1	1
3	1	1	1	0	1
4	1	0	1	1	0

Adjacency list representation:-



vertex NO	Indegree	Outdegree
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

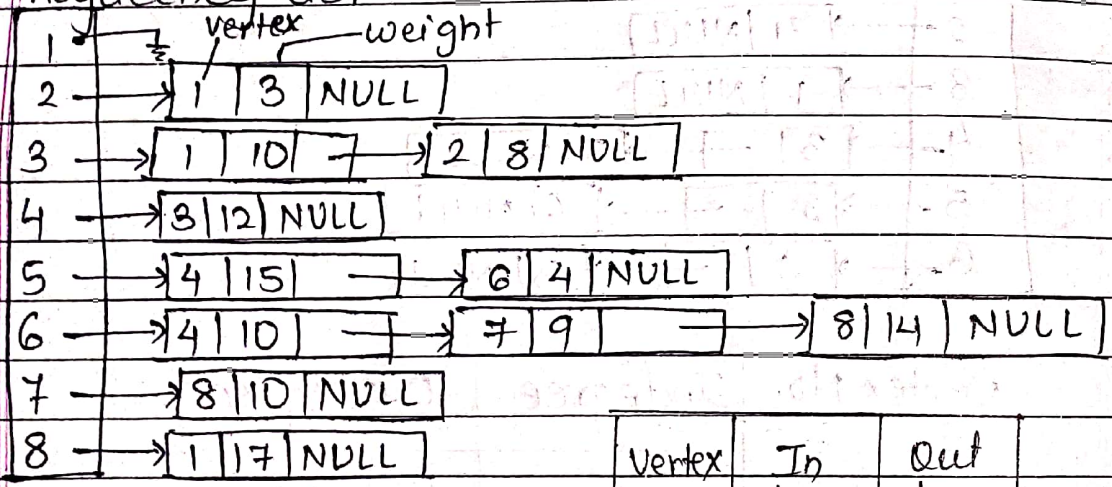
(2)



→ Adjacency matrix

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	3	0	0	0	0	0	0	0
3	10	8	0	0	0	0	0	0
4	0	0	12	0	0	0	0	0
5	0	0	0	15	0	4	0	0
6	0	0	0	10	0	0	9	14
7	0	0	0	0	0	0	0	10
8	17	0	0	0	0	0	0	0

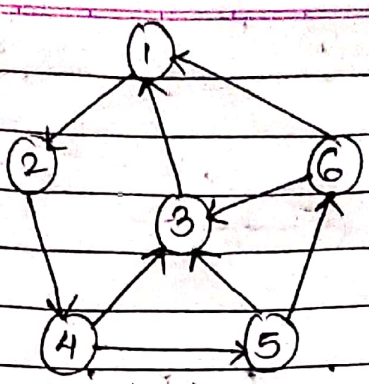
Adjacency list:-



Vertex No.	In degree	Out degree
1	3	0
2	1	1
3	1	2
4	2	1
5	0	2
6	1	3
7	1	1
8	2	1

PAGE No.
DATE / /

(3)



→ Adjacency matrix :-

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

Adjacency list :-

1 → [2 | NULL]
2 → [4 | NULL]
3 → [1 | NULL]
4 → [3 |] → [5 | NULL]
5 → [3 |] → [6 | NULL]
6 → [1 |] → [3 | NULL]

Vertex No.	Indegree	Outdegree.
1	2	1
2	1	1
3	3	1
4	1	2
5	1	2
6	1	2