DATA STRUCTURES

(CBCGS DEC 2017)

Q1. a) Explain ADT. List the Linear and Non-Linear data structures with example. (5)

Solution:

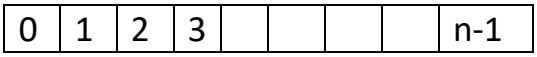
An ADT is a specified mathematical entity. It has a specific interface. A collection of signature of operations that can be involved on an instance.

It is a kind of abstraction which means the user does not know about the operations taking place at the backend.

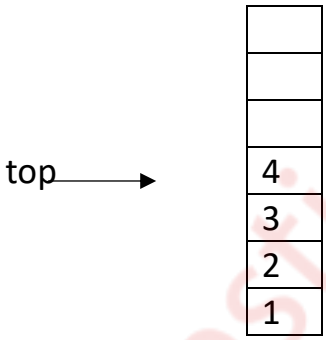
Linear data structures:

Elements are arranged in a linear fashion. Types of linear data structures are:

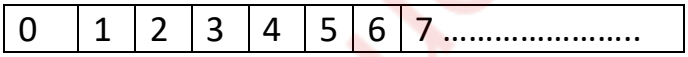
->Lists



->stack



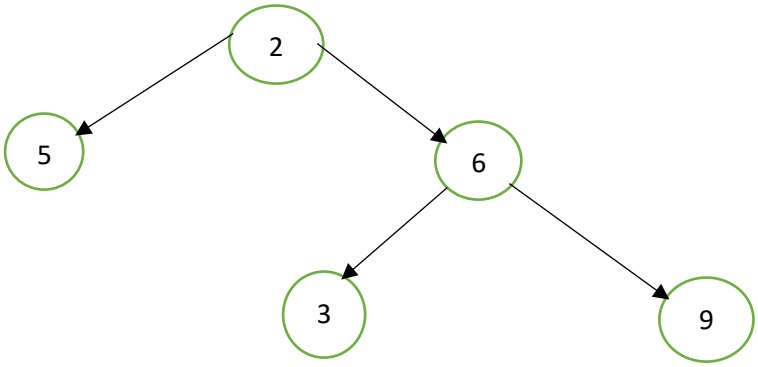
->queue



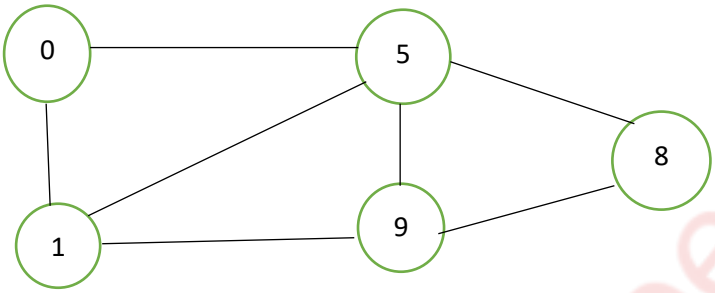
Non-linear data structures:

Every data element can have many successors and predecessors. Types of non-linear data structures:

->Tree



->Graph



->table

10	25	30	7	55
5	5	2	10	2
2	50	10	25	10

Q1. b) Explain B tree and B+ tree. **(5)**

Solution:

I.B tree

->A B-tree is a method of placing and locating files (called records or keys) in a database.

->The B-tree algorithm minimizes the number of times a medium must be accessed to locate a desired record, thereby speeding up the process.

->Properties:

1. All the leaf nodes must be at same level.

2 . All nodes except root must have at least [m/2]-1 keys and maximum of m-1 keys.

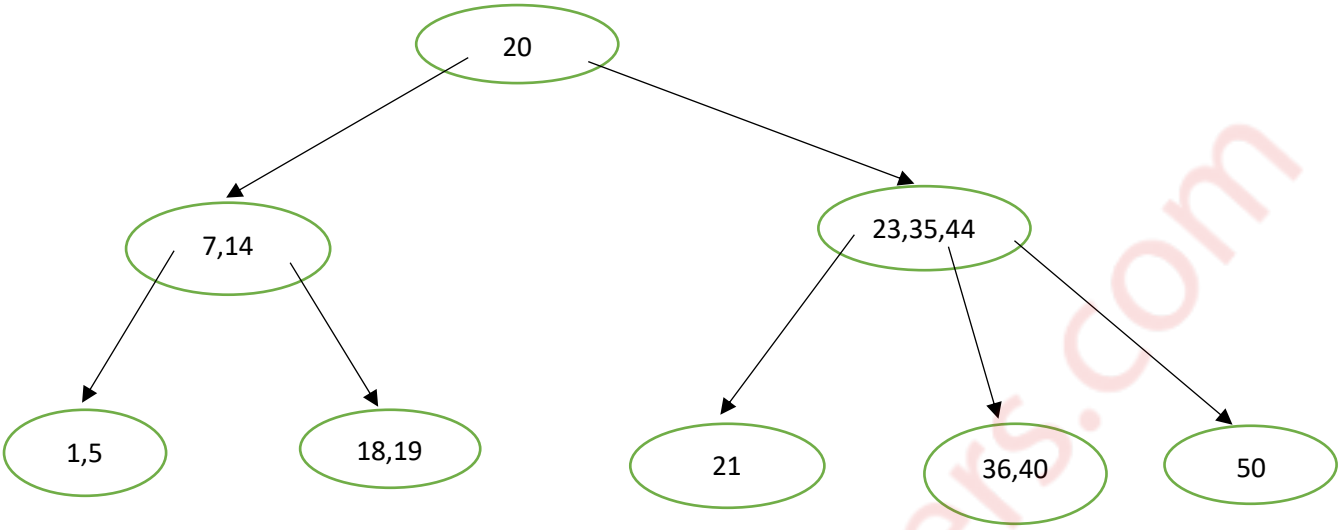
3. All non leaf nodes except root (i.e. all internal nodes) must have at least m/2 children.

4.If the root node is a non leaf node, then it must have at least 2 children.

5.A non leaf node with n-1 keys must have n number of children.

6.All the key values within a node must be in Ascending Order.

->example:



II.B+ Tree

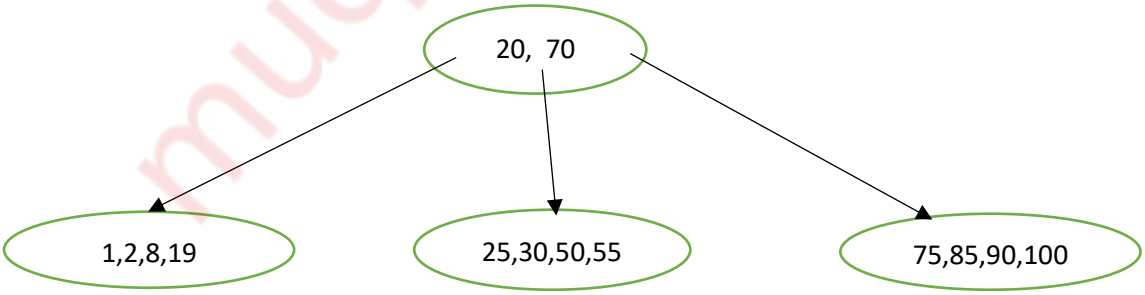
->A B+ tree is a data structure often used in the implementation of database indexes.

->Each node of the tree contains an ordered list of keys and pointers to lower level nodes in the tree. These pointers can be thought of as being between each of the keys.

->Properties:

1. The root node points to at least two nodes.
2. All non-root nodes are at least half full.
3. For a tree of order m, all internal nodes have m-1 keys and m pointers.
4. A B+-Tree grows upwards.
5. A B+-Tree is balanced.
6. Sibling pointers allow sequential searching.

->example:



Q1. c) Write a program to implement Binary Search on sorted set of integers. (10)

Solution:

Code:

```
#include<stdio.h>
int n;
int binary(int a[],int n,int x)
{
  int low=0,high=n-1,mid;
  while(low<=high)
  {
     mid=(low+high)/2;
    if(x==a[mid])
    return(mid);

 else if(x<a[mid])
    high=mid-1;
    else
    low=mid+1;
  }
  return(-1);
}
int main()
{
  int y,i,j;
  printf("enter the size of the array:");
  scanf("%d",&n);
  int a[n];
  printf("entered sorted list of array:");
  for(i=0;i<n;i++)
  {
    scanf("%d",&a[i]);
```

```
  }
  printf("enter the number to be searched:");
  scanf("%d",&y);
  j=binary(a,n,y);
  i=j;
  if(j==-1)

  printf("element not found");

else
  {
    printf("\n %d occurred at position=%d \n",y,i+1);
  }
  return 0;
}
```

Output:

enter the size of the array:5

entered sorted list of array:1 2 5 8 9

enter the number to be searched:5

5 occurred at position=3

Q2. a) Write a program to convert infix expression into postfix expression. (10)

Solution:

Code:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<stdlib.h>
#define size 50
```

```
int top=-1;
char s[size];
void push(char ch)
{
        s[++top]=ch;
}
char pop()
{
        return(s[top--]);
}
int prec(char ch)
{
    switch(ch)
        {
                case '#':return 0;
                break;
                case '(':return 1;
                break;
                case '-': case'+':return 2;
                break;
                case '/': case '*': return 3;
                break;
        }
}
int main()
{
        char ch,elem,ix[50],px[50];
        int i=0,k=0;
        push('#');
```

```
        printf("enter infix expression:");
        gets(ix);
        while((ch=ix[i++])!='\0')
        {
                if(isalnum(ch))
                px[k++]=ch;
                else if (ch=='(')
                push(ch);
                else if(ch==')')
                {
                        while(s[top]!='(')
                        px[k++]=pop();
                        elem=pop();
                }
                else
                {
                        if(s[top]=='#')
                        push(ch);
                        else if( s[top]=='(')
                        push(ch);
                        else if(prec(ch)>prec(s[top]))
                        push(ch);
                        else
                        px[k++]=pop();
                }
        }
while(s[top]!='#')
        px[k++]=pop();
        px[k]='\0';
```

```
        printf("postfix expression :");
        puts(px);
        return 0;
}
```

Output:

enter infix expression:a+b*(c+d)

postfix expression :abcd+*+

Q2. b) Explain Huffman encoding with an example. (10)

Solution:

-> Huffman encoding is a concept which is developed by David Huffman.

-> The original data can be perfectly reconstructed from compressed data.

-> It is mainly used in instant compression format, Jpack, png, mp3 and gzip.

-> Algorithm

Sort given symbol in increasing order of frequency then create a binary tree using following steps:-

1.From the frequency table select 2 symbols s1 and s2 with minimum frequency f1 and f2. Create a node for s1 and s2.

2.Combine the nodes s1 and s2 and create a new symbol s12. Create a node for s12.

3.Remove s1 and s2 from frequency table and insert s1 keeping the frequency table sorted.

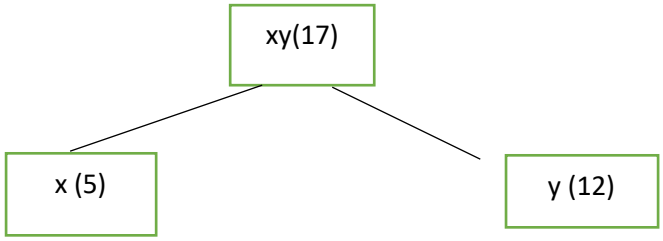
4.Repeat step 1 and 3 till only one symbol is left in frequency table.

-> Example:

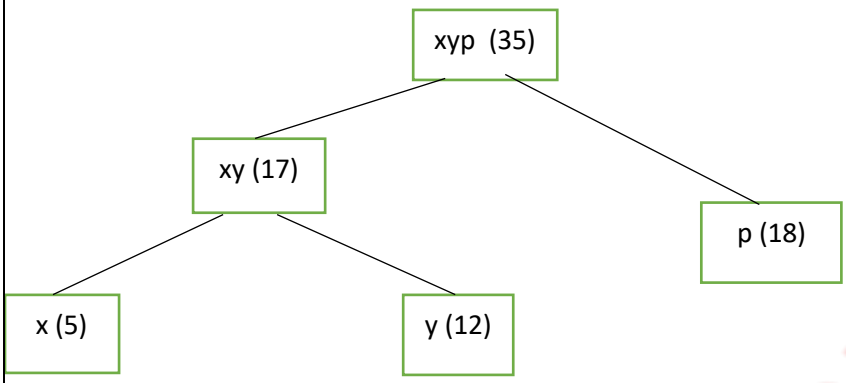
a	m	x	y	p
85	20	5	12	18

x	y	p	m	a
5	12	18	20	85

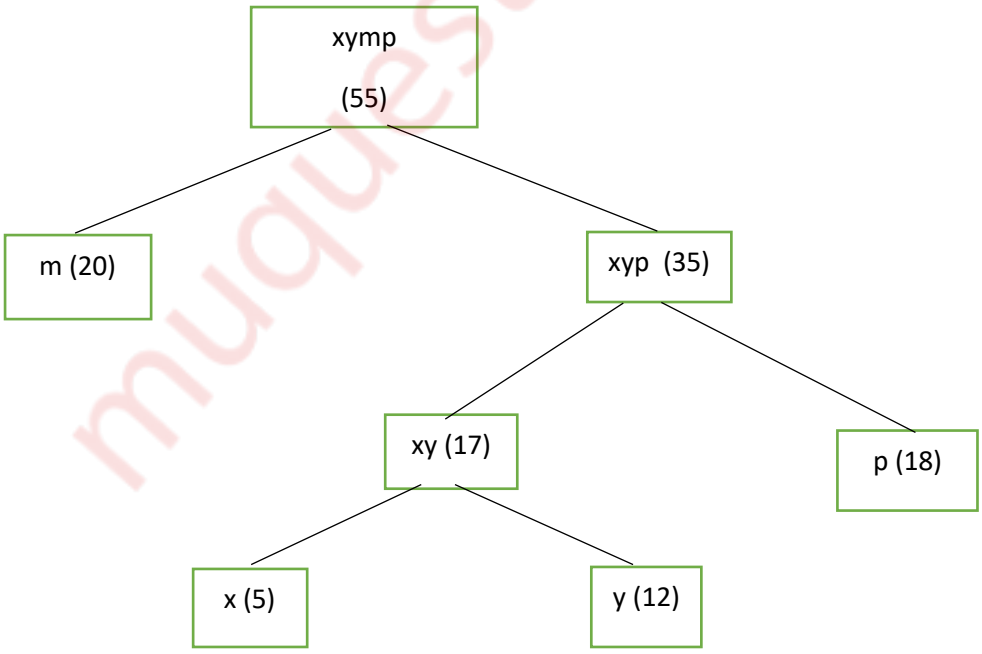
xy	p	m	a
17	18	20	85

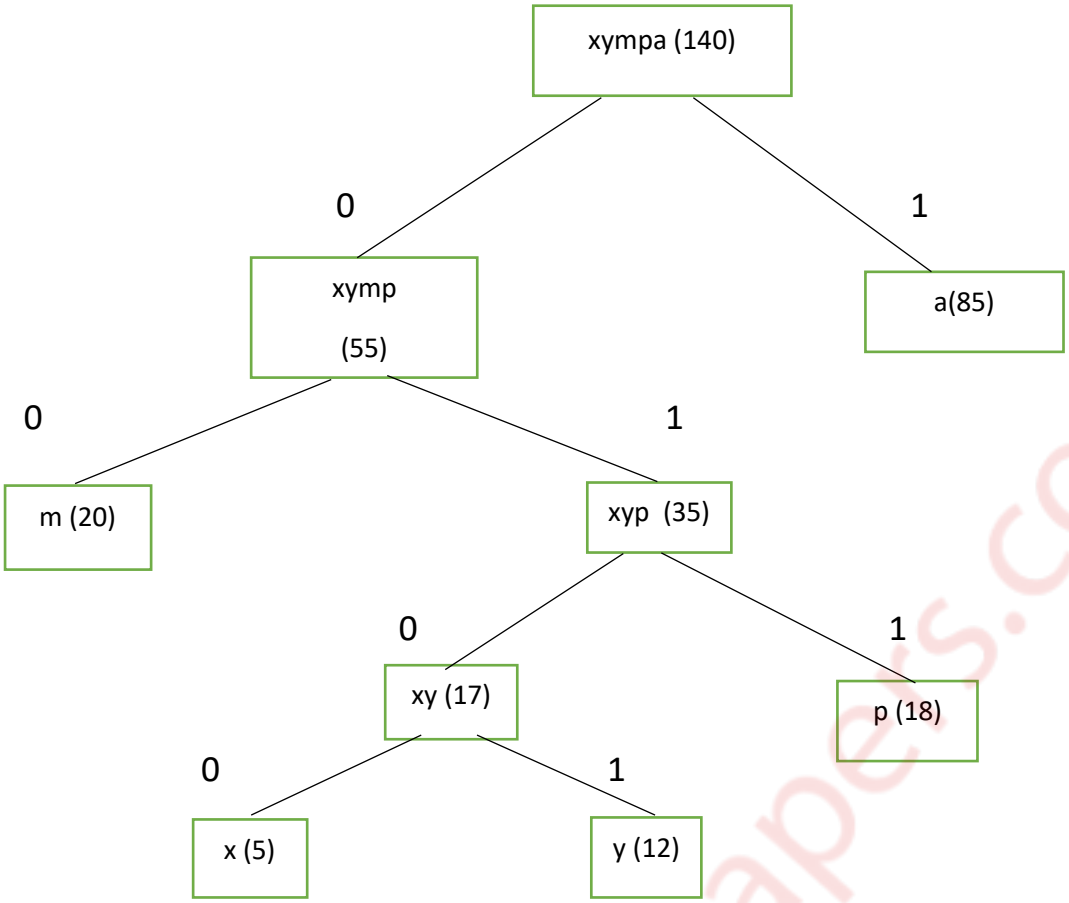


m	xyp	a
20	35	85



xypm	a
55	85





alphabets	frequency	bits	No. of bits
m	20	00	2
x	5	0100	4
y	12	0101	4
p	18	011	3
a	85	1	1

No. of bits required= 20 x2 + 5 x 4 + 12 x 4 +18 x 3 +85 x 1=247 bits

---

Q3. a) Write a program to implement Doubly Linked List. Perform the following operations:

(i) Insert a node in the beginning
(ii) Insert a node in the end
(iii) Delete a node from the end
(iv) Display a list **(10)**

Solution:

Code:

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

```
#include <stdlib.h>

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

struct node *head = NULL, *tail = NULL;

struct node * createNode(int data) {
    struct node *newnode;
    newnode = (struct node *)malloc(sizeof (struct node));
    newnode->data = data;
    newnode->next = NULL;
    newnode->prev = NULL;
    return (newnode);
}

void createDummies() {
    head = (struct node *)malloc(sizeof (struct node));
    tail = (struct node *)malloc(sizeof (struct node));
    head->data = tail->data = 0;
    head->next = tail;
    tail->prev = head;
    head->prev = tail->next = NULL;
}
void insertAtStart(int data)
{
    struct node *newnode = createNode(data);
```

```
        newnode->next = head->next;
        newnode->prev = head;
        head->next->prev = newnode;
        head->next = newnode;
        nodeCount++;
    }
    void insertAtEnd(int data)
   {
        struct node *newnode = createNode(data);
        newnode->next = tail;
        newnode->prev = tail->prev;
        tail->prev->next = newnode;
        tail->prev = newnode;
        nodeCount++;
    }


void deleteend()
{
        struct node *ptr=tail;
        tail->prev->next=tail->next;
        tail=tail->prev;

        free(ptr);
}

void display()
{
        struct node *ptr;
```

```
        ptr=head->next;
        while (ptr!=tail)
        {
                printf("%d \t ",ptr->data);
                ptr=ptr->next;
        }
}
  int main()
   {
        int data, ch, pos;
        createDummies();
        while (1)
                {
              printf("\n 1. Insert at start\n 2.Insert at end\n 3.delete at end\n 4. Exit\n");
              printf("Enter your choice:");
              scanf("%d", &ch);
              switch (ch) {
               case 1: printf("Enter your data to  insert at start:");
                        scanf("%d", &data);
                        insertAtStart(data);
                        display();
                        break;
                    case 2:
                        printf("Enter your data to insert:");
                        scanf("%d", &data);
                        insertAtEnd(data);
                        display();
                        break;
                   case 3: deleteend();
```

```
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("U have entered wrong option\n");
                break;
        }
    }
    return 0;
}
```

Output:

```
1. Insert at start
 2.Insert at end
 3.delete at end
 4. Exit
Enter your choice:1
Enter your data to  insert at start:2
2
 1. Insert at start
 2.Insert at end
 3.delete at end
 4. Exit
Enter your choice:1
Enter your data to insert at start:3
3       2
 1. Insert at start
 2.Insert at end
 3.delete at end
```

```
 4. Exit
Enter your choice:2
Enter your data to insert:5
3       2       5
 1. Insert at start
 2.Insert at end
 3.delete at end
 4. Exit
Enter your choice:3
3       2
 1. Insert at start
 2.Insert at end
 3.delete at end
 4. Exit
Enter your choice:4
```

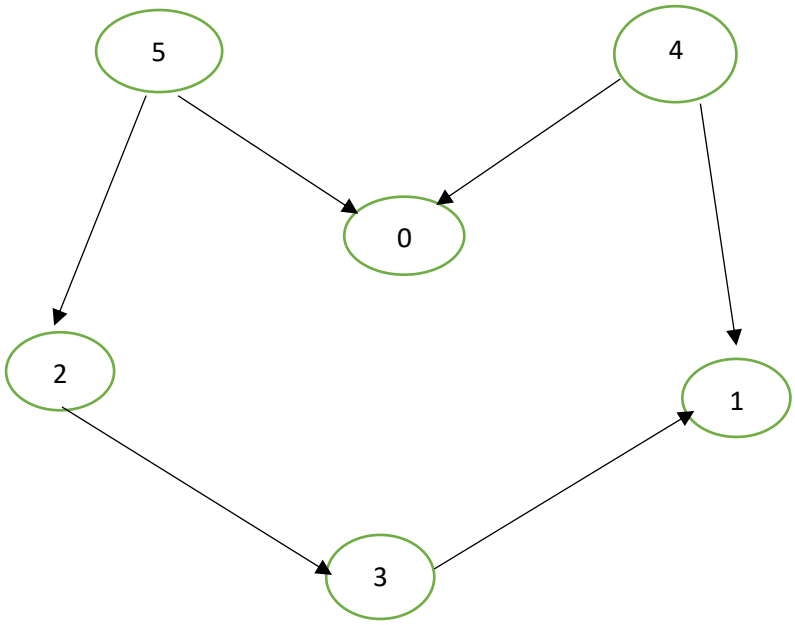
Q3. b) Explain Topological sorting with example. (10)

Solution:

->Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

->For example, a topological sorting of the following graph is “5 4 2 3 1 0”.

->There can be more than one topological sorting for a graph. For example, another topological sorting of the following graph is “4 5 2 3 1 0”. The first vertex in topological sorting is always a vertex with in-degree as 0



-> we use a temporary stack. We don't print the vertex immediately, we first recursively call topological sorting for all its adjacent vertices, then push it to a stack.

-> Finally, print contents of stack. A vertex is pushed to stack only when all of its adjacent vertices (and their adjacent vertices and so on) are already in stack.

Q4. a) Write a program to implement Quick sort. Show the steps to sort the given numbers: 25,13,7,34,56,23,13,96,14,2 (10)

Solution:

Code:

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

void quick(int a[40],int first,int last)
{
  int pivot,j,i,temp;
  if(first<last)
  {
    pivot=first;
    i=first;
    j=last;

    while(i<j)
    {
      while(a[i]<=a[pivot] && i<last)
      i++;
```

```
        while(a[j]>a[pivot])
        j--;

        if(i<j)
        {
          temp=a[i];
          a[i]=a[j];
          a[j]=temp;
        }
      }
      temp=a[pivot];
      a[pivot]=a[j];
      a[j]=temp;

      quick(a,first,j-1);
      quick(a,j+1,last);
    }
}
  void main()
{
   int a[20],n,i;
   printf("enter the no. of elements:");
   scanf("%d",&n);

   printf("enter the elements:");
   for(i=0;i<n;i++)
   {
     scanf("%d",&a[i]);
   }

   quick(a,0,n-1);
   printf("sorted elements are:");
   for(i=0;i<n;i++)
   {
    printf("%d \t",a[i]);
   }
}
```

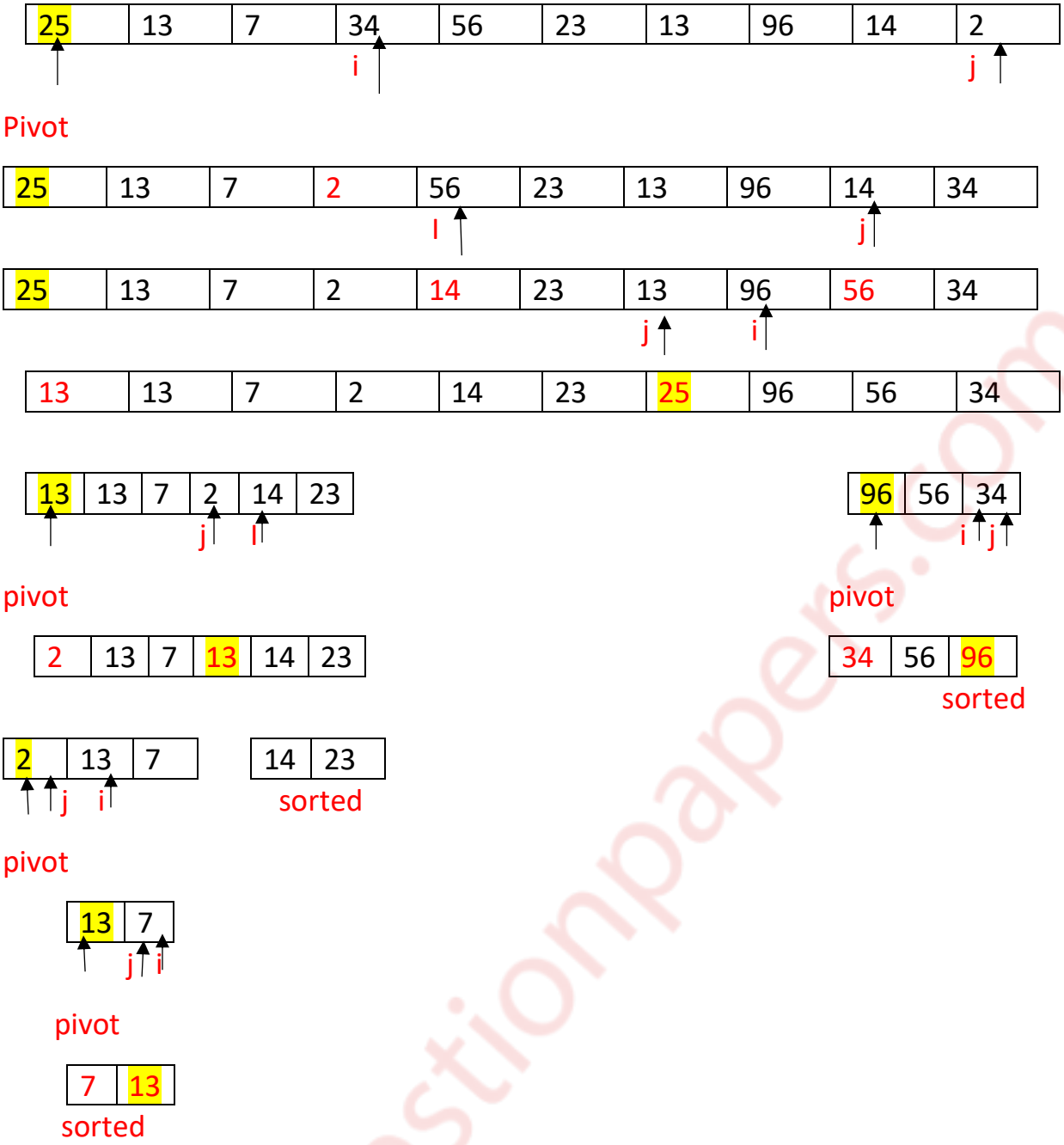
Output:

enter the no. of elements:10

enter the elements:25 13 7 34 56 23 13 96 14 2

sorted elements are:2 7 13 13 14 23 25 34 56 96

steps:



final sorted list is:

2	7	13	13	14	23	25	34	56	96

Q4. b) Write a program to implement linear queue using array. **(10)**

Solution:

Code:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 50
```

```
void insert();
void delet();
void display();
int queue[MAX],rear=-1,front=-1,item;
int main()
{
    int ch;
    do
    {
        printf("\n\n1.Insert\n2 Delete\n3.Display\n4.Exit\n\n");
        printf("Enter your choice\n");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:
                    insert();
                    break;

                case 2:
                    delet();

                    break;
                case 3:
                    display();
                   break;
                  case 4:
                   exit(0);
                   default:
                      printf("Invalid choice, Please try again\n");
```

```
                    }
  }while(1);
          getch();
}
void insert()
{
   if(rear==MAX-1) // condition for Queue Full
      printf("Queue is Full\n");
   else
   {
   printf("\n\n Enter item\n");
   scanf("%d",&item);

                if(rear == -1 && front ==-1  ) // is Queue Empty
                {
                rear=0;
                front=0;
                }
                else            // otherwise
                rear++;
         queue[rear]=item; // adding element to Queue
       printf("\n\n Item Inseted :%d",item);
   }
  } // end insert
void delet()
{
    if(front==-1) // Queue is empty
        printf( "\nQueue is empty\n");
       else //  it has element
```

```
    {
       item = queue[front];
         if(front==rear) // the element is the last element
       {
       front = -1 ;
       rear=-1;
       }
       else  // not the last element
       front++;
    printf("\n\n Item deleted: %d ",item);
    }
}
void display()
{
     int i;
    if( front ==-1)
       printf("\nQueue is empty\n");
    else
    {
       printf("\n\n");

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

Output:

1.Insert

2 Delete

3.Display

```
4.Exit
Enter your choice
1
Enter item
12
 Item Inserted :12
1.Insert
2 Delete
3.Display
4.Exit
Enter your choice
1
 Enter item
55
 Item Inserted :55

1.Insert
2 Delete
3.Display
4.Exit
Enter your choice
1
Enter item
88
Item Inserted :88
1.Insert
2 Delete
3.Display
```

```
4.Exit
Enter your choice
1
 Enter item
66
 Item Inserted :66
1.Insert
2 Delete
3.Display
4.Exit
Enter your choice
3
12    55    88    66
1.Insert
2 Delete
3.Display
4.Exit
Enter your choice
2
Item deleted: 12
1.Insert
2 Delete
3.Display
4.Exit
Enter your choice
3
 55    88    66
1.Insert
```

```
2 Delete
3.Display
4.Exit
Enter your choice
4
```

Q5. a) Write a program to implement STACK using Linked list. What are the advantages linked list over array? (10)

Solution:

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int Data;
   struct Node *next;
}*top;
void popStack()
{
   struct Node  *var=top;

   if (top==NULL)
   {
   printf("\nStack Empty");
   }
   else
   {
      top = top->next;
```

```
        free(var);
         printf("element removed");
      }
   }
   void push(int value)
{
      struct Node *temp;

      temp=(struct Node *)malloc(sizeof(struct Node));

      temp->Data=value;

      if (top == NULL)
      {
          top=temp;
          top->next=NULL;
      }
      else
      {
         temp->next=top;
         top=temp;
      }
   }
   void display()
   {
       struct Node *var=top;
       if(var!=NULL)
       {
           printf("\nElements are as:\n");
```

```
        while(var!=NULL)
        {
            printf("\t%d\n",var->Data);
            var=var->next;
        }
    printf("\n");
    }
    else
    printf("\nStack is Empty");
}
int main()
{
    int c,val;
    top=NULL;
    printf(" \n1. Push to stack");
    printf(" \n2. Pop from Stack");
    printf(" \n3. Display data of Stack");
    printf(" \n4. Exit\n");
 do
    {
        printf(" \nChoose Option: ");
        scanf("%d",&c);
         switch(c)
        {
            case 1:
          {
             printf("\nEnter a value to be to pushed into Stack: ");
            scanf("%d",&val);
```

```
    push(val);
    printf("%d added onto stack",val);

    break;
    }
    case 2:
    {
    popStack();
    break;
    }
    case 3:
    {
    display();
    break;
    }
    case 4:
   {
         struct Node *temp;
        while(top!=NULL)
    {
        temp = top->next;
        free(top);
        top=temp;
    }
    exit(0);
}
default:
    {
    printf("\n  Wrong choice.");
```

```
            }
        }
    }while(c!=4);
}
```

Output:

1. Push to stack

2. Pop from Stack

3. Display data of Stack

4. Exit

Choose Option: 1

Enter a value to be to pushed into Stack: 10

10 added onto stack

Choose Option: 1

Enter a value to be to pushed into Stack: 20

20 added onto stack

Choose Option: 1

Enter a value to be to pushed into Stack: 30

30 added onto stack

Choose Option: 1

Enter a value to be to pushed into Stack: 11

11 added onto stack

Choose Option: 3

Elements are as:

11

30

20

10

Choose Option: 2

element removed

```
Choose Option: 3
Elements are as:
        30
        20
        10
Choose Option: 4
```

Advantages of linked list over array:

1. There is no need to specify the size for linked list as it can grow and shrink during execution.
2. Insertion and deletion of an element is easier, faster and efficient.
3. It involves dynamic allocation of memory.
4. Memory utilization is efficient.

Q5. b) Write a program to implement Binary Search Tree (BST), Show BST for the following inputs: 10,5,4,12,15,11,3 (10)

Solution:

Code:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
  int data;
  struct node *left;
  struct node *right;
}node;
node *insert(node *t,int key)
{
  if(t==NULL)
  {
```

```
    t=(node *)(malloc(sizeof(node)));
    t->data=key;
    t->left=NULL;
    t->right=NULL;
  }
  else
  {
    if(key>t->data)
    t->right=insert(t->right,key);
    else
    t->left=insert(t->left,key);
  }
  return(t);
}
node *search(node *t,int key)
{
  if(t==NULL)
  return(NULL);
  else if(t->data==key)
  return(t);

else if(key>t->data)
  return(search(t->right,key));
  else
  return(search(t->left,key));
}
node *max(node *t)
{
  if(t==NULL)
```

```
  return(NULL);
  while(t->right!=NULL)
  t=t->right;
  return(t);
}
node *min(node *t)
{
  if(t==NULL)
  return(NULL);
  while(t->left!=NULL)
  t=t->left;
  return(t);
}
void preorder(node *temp)
{
  if(temp!=NULL)
  {

  printf("\n \n data: %d",temp->data);
    preorder(temp->left);
    preorder(temp->right);
  }

}
void postorder (node *temp)
{
   if(temp!=NULL)
   {
     postorder(temp->left);
```

```
    postorder(temp->right);
    printf("\n \n data: %d",temp->data);
   }
}
void inorder(node *temp)
{
  if(temp!=NULL)
   {
    postorder(temp->left);
    printf("\n \n data: %d",temp->data);
    postorder(temp->right);
   }
}
int main()
{
  struct node *root=NULL;
  struct node *temp=NULL;
  int c,key;
 do
  {
   printf("\n 1.insert node \n 2.search \n 3.maximum number \n 4.minimum number \n
5.display in inorder format \n 6.display in postorfer format \n 7.display in preorder format
\n 8.exit \n");
   printf("enter your choice:");
   scanf("%d",&c);
   switch(c)
   {
    case 1:printf("enter data:");
           scanf("%d",&key);
           if(root==NULL)
```

```
            root=insert(root,key);
            else
            temp=insert(root,key);
            break;
      case 2:printf("enter key:");
            scanf("%d",&key);
            temp=search(root,key);
            if(temp!=NULL)
            printf("%d found",key);
            else
            printf("%d not found",key);
      break;
      case 3:temp=max(root);
            printf("max=%d",temp->data);
      break;
      case 4:temp=min(root);
            printf("min=%d",temp->data);
      break;
      case 5:inorder(root);
      break;
      case 6:postorder(root);
      break;
      case 7:preorder(root);
      break;
    }
  }while(c!=8);
  return 0;
}
```

Output:

1.insert node

2.search

3.maximum number

4.minimum number

5.display in inorder format

6.display in postorfer format

7.display in preorder format

8.exit

enter your choice:1

enter data:10

1.insert node

2.search

3.maximum number

4.minimum number

5.display in inorder format

6.display in postorfer format

7.display in preorder format

8.exit

enter your choice:1

enter data:55

1.insert node

2.search

3.maximum number

4.minimum number

5.display in inorder format

6.display in postorfer format

7.display in preorder format

```
8.exit
enter your choice:1
enter data:7
1.insert node
 2.search
 3.maximum number
 4.minimum number
  5.display in inorder format
 6.display in postorfer format
 7.display in preorder format
 8.exit
enter your choice:2
enter key:7
7 found
 1.insert node
 2.search
 3.maximum number
 4.minimum number
  5.display in inorder format
 6.display in postorfer format
 7.display in preorder format
 8.exit
enter your choice:3
max=55
1.insert node
 2.search
 3.maximum number
 4.minimum number
```

5.display in inorder format

6.display in postorfer format

7.display in preorder format

8.exit

enter your choice:4

min=7

1.insert node

2.search

3.maximum number

4.minimum number

5.display in inorder format

6.display in postorfer format

7.display in preorder format

8.exit

enter your choice:5

data: 7

data: 10

data: 55

1.insert node

2.search

3.maximum number

4.minimum number

5.display in inorder format

6.display in postorfer format

7.display in preorder format

8.exit

enter your choice:6

data: 7

data: 55

data: 10

1.insert node

2.search

3.maximum number

4.minimum number

5.display in inorder format

6.display in postorfer format

7.display in preorder format

8.exit

enter your choice:7

data: 10

data: 7

data: 55

1.insert node

2.search

3.maximum number

4.minimum number

5.display in inorder format

6.display in postorfer format

7.display in preorder format

8.exit

enter your choice:8

Q6. Write Short notes on(any two) :

(a)AVL Tree **(10)**

Solution:

->An AVL tree is a height balance tree. These trees are binary search trees in which the heights of two siblings are not permitted to differ by more than one.

->Searching time in a binary search tree is O(h) where h is the height of the tree. For efficient searching it is necessary that the height should be kept minimum.

-> A full binary search tree with n nodes will have a height of O(log n). In practice, it is very difficult to control the height of a BST. It lies between O(n) to O(log n). An AVL tree is a close approximation of full binary search tree.

->The balance factor of a node in a AVL tree could be -1, 0 or 1.

1. 0 indicates that the left and right subtrees are equal.

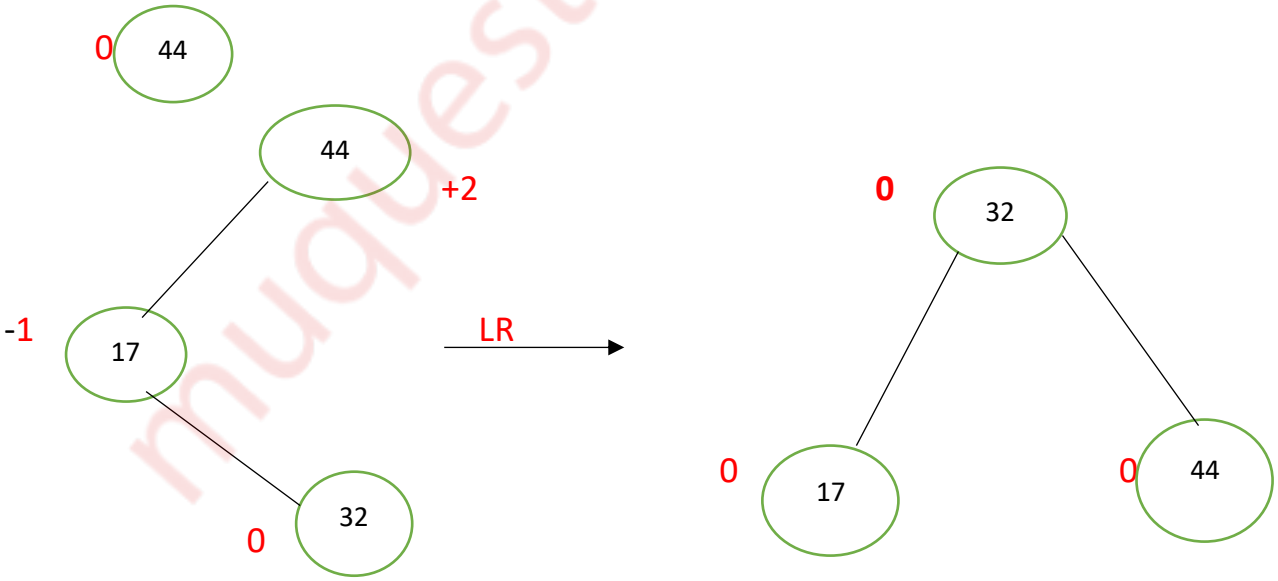
2. +1 means the height of the left subtree is one more than the height of the right subtree.

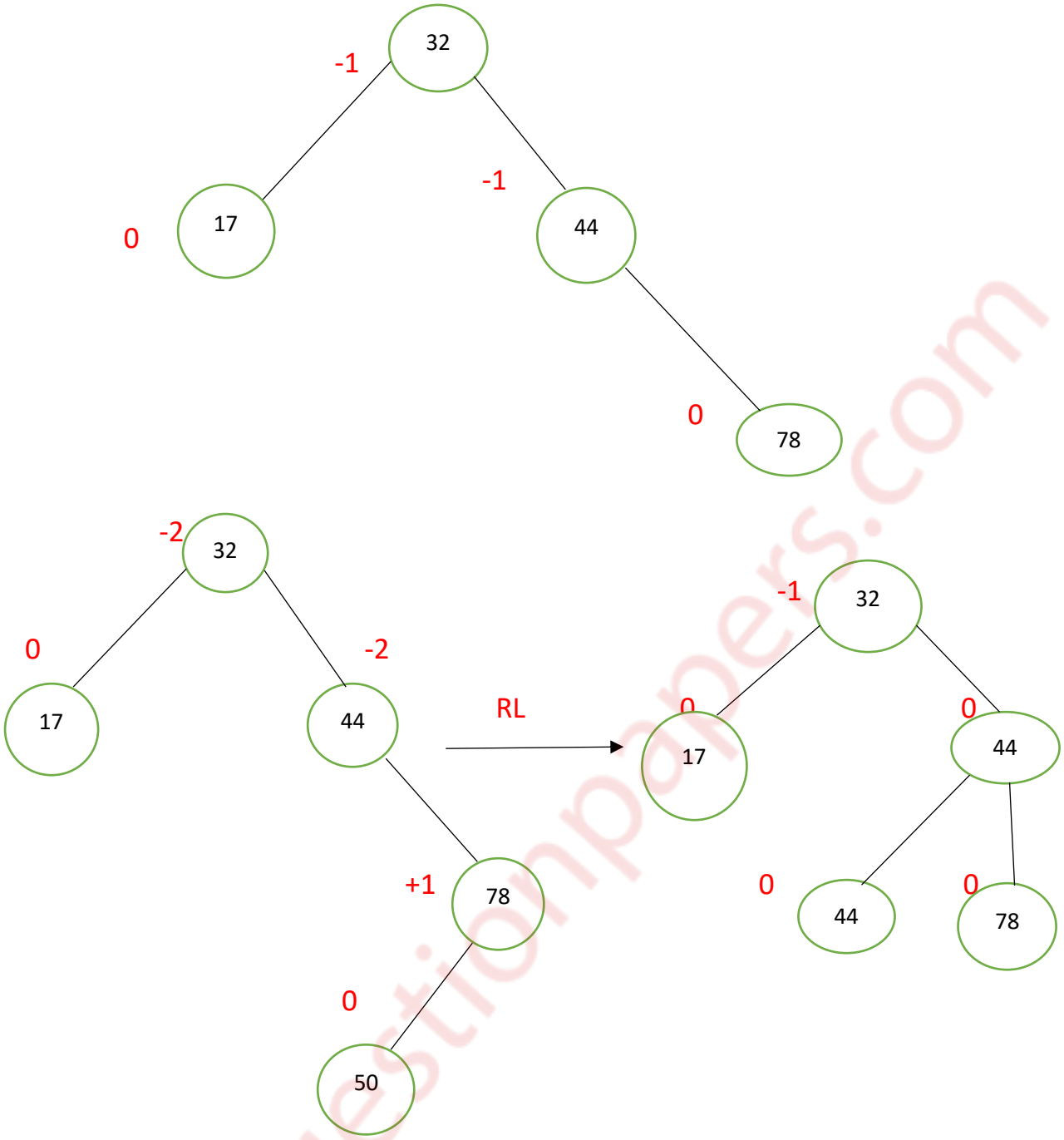
3. -1 indicates that the height of the right subtree is one more than the height of the left subtree.

->

Type of nodes	Rotation
Left left	Right
Right right	Left
Right left	i. right ii. left
Left right	i.left ii. right

->eg: 44,17,32,78,50,88,48,62,54





(a) Graph Traversal Techniques **(10)**

Solution:

Traversal of a graph means visiting each node and visiting exactly once.

Two commonly used techniques are:

i)Depth-First Traversal

Rule 1-If possible, visit an adjacent unvisited vertex, mark it, and push it on the stack.

RULE 2-If you can't follow Rule 1, then, if possible, pop a vertex off the stack.

RULE 3-If you can't follow Rule 1 or Rule 2, you're done

-> In this method, After visiting a vertex v, which is adjacent to w1, w2, w3, ...; Next we visit one of v's adjacent vertices, w1 say. Next, we visit all vertices adjacent to w1 before coming back to w2, etc.

->Must keep track of vertices already visited to avoid cycles.

->The method can be implemented using recursion or iteration.

->The iterative preorder depth-first algorithm is:

push the starting vertex onto the stack

while(stack is not empty){

pop a vertex off the stack, call it v

if v is not already visited, visit it

push vertices adjacent to v, not visited, onto the stack

}

ii)Breadth-First Traversal

->In this method, After visiting a vertex v, we must visit all its adjacent vertices w1, w2, w3, ..., before going down next level to visit vertices adjacent to w1 etc.

->The method can be implemented using a queue.

->A boolean array is used to ensure that a vertex is enqueued only once.

enqueue the starting vertex

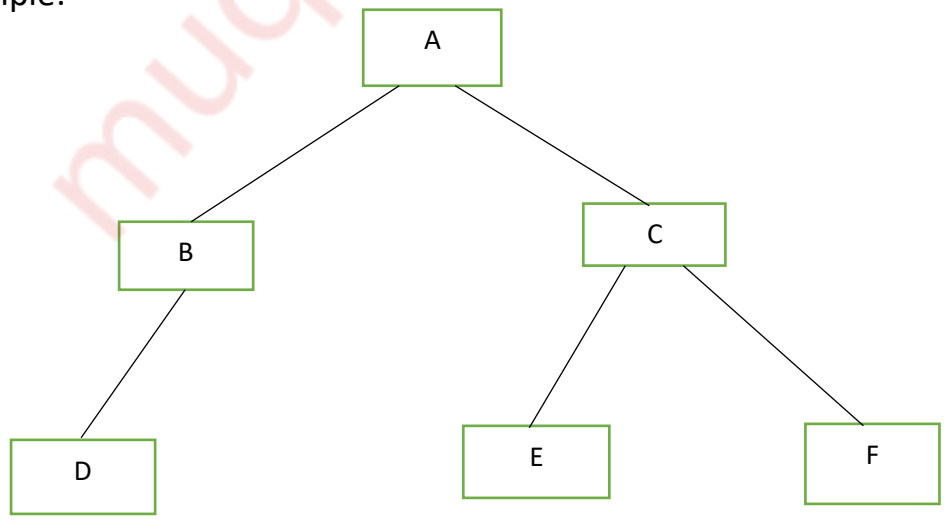
while(queue is not empty){

dequeue a vertex v from the queue;

visit v.

enqueue vertices adjacent to v that were never enqueued; }

Example:



DFS: A,B,D,C,E,F

BFS: A,B,C,D,E,F

(b) Expression Trees **(10)**

Solution:

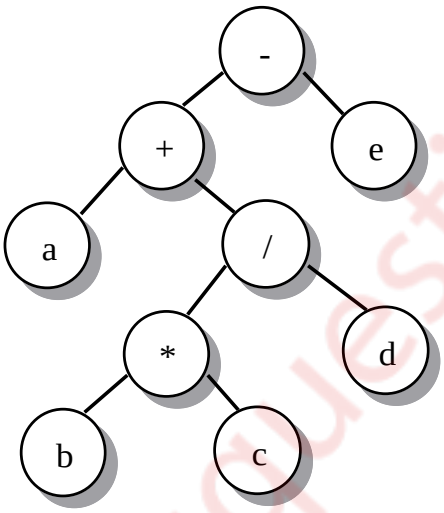
->Expression tree as name suggests is nothing but expressions arranged in a tree-like data structure. Each node in an expression tree is an expression.

->it is a tree with leaves as operands of the expression and nodes contain the operators. Similar to other data structures, data interaction is also possible in an expression tree.

->For example, an expression tree can be used to represent mathematical formula x < y where x, < and y will be represented as an expression and arranged in the tree like structure.

->Expression tree is an in-memory representation of a lambda expression. It holds the actual elements of the query, not the result of the query.

->Expression trees are mainly used for analyzing, evaluating and modifying expressions, especially complex expressions.



a + b*c/d - e

(c) Application of Linked list -Polynomial Addition **(10)**

Solution:

Code:

#include<stdio.h>

```
#include<stdlib.h>
#include<ctype.h>
struct node
{
    int coef;
    int exp;
    struct node*next;
};

struct node *insert(struct node *s, int c, int e)
{
    struct node *temp;
    struct node *ptr;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->coef=c;
    temp->exp=e;
    temp->next=NULL;
    if(s==NULL)
    {
            s=temp;
    }
    else

    {
            ptr=s;
            while(ptr->next!=NULL)
            {
                    ptr=ptr->next;
            }
```

```
            ptr->next=temp;
        }
        return s;
}

struct node *addpoly(struct node *t1,struct node *t2)
{
        struct node *t3=NULL;
        int sum;
        while(t1!=NULL && t2!=NULL)
        {
                if(t1->exp==t2->exp)
                {
                        sum=t1->coef+t2->coef;
                        t3=insert(t3,sum,t1->exp);
                        t1=t1->next;
                        t2=t2->next;
                }
                else if((t1->exp) > (t2->exp))

          {
                        t3=insert(t3,t1->coef,t1->exp);
                        t1=t1->next;
                }
                else if(t2->exp > t1->exp)
                {
                        t3=insert(t3,t2->coef,t2->exp);
                        t2=t2->next;
                }
```

```
        if(t1==NULL)
        {
                while(t2!=NULL)
                {
                        t3=insert(t3,t2->coef,t2->exp);
                        t2=t2->next;
                }
        }
        if(t2==NULL)
        {
                while(t1!=NULL)
                {
                        t3=insert(t3,t1->coef,t1->exp);
                        t1=t1->next;
                }
        }

}
    return t3;
}
void display(struct node*temp)
{

    while(temp!=NULL)
    {
        printf("%d %d\n",temp->coef,temp->exp);
        temp=temp->next;
    }
}
```

```
int main( )
{
    int n,m,i,c,e;
    struct node *p1;
    struct node *p2;
    struct node *p3;
    p1=NULL;
    p2=NULL;
    p3=NULL;
    printf("Enter no. of terms in first polynomial \n");
    scanf("%d",&n);
    for(i=0;i<=(n-1);i++)

{
            printf("enter coefficient and exponent\n");
            scanf("%d %d",&c,&e);
            p1=insert(p1,c,e);
    }
    printf("Enter no. of terms in second polynomial \n");
    scanf("%d",&m);
    for(i=0;i<=(m-1);i++)
    {
            printf("enter coefficient and exponent\n");
            scanf("%d%d",&c,&e);
            p2=insert(p2,c,e);
    }
    p3=addpoly(p1,p2);
    printf("first polynomail: \n");
```

```
    display(p1);
    printf("second  polynomail: \n");
    display(p2);
    printf("resultant polynomail: \n");
    display(p3);
    return 0;
}
```

Output:

```
enter coefficient and exponent
4 1
enter coefficient and exponent
6 0
Enter no. of terms in second polynomial
3
enter coefficient and exponent
7 2
enter coefficient and exponent
3 1
enter coefficient and exponent
1 0
first polynomail:
1 3
2 2
4 1
6 0
second  polynomail:
7 2
3 1
1 0
```

resultant polynomail:

1 3

9 2

7 1

7 0