WINTER – 13 EXAMINATION

Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills.
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

Q.1.

A. Attempt any six questions from the following:

a) State different types of Data types.(1 mark for each type)

1) **Primitive** data type: These are basic strictures and directly operated upon by the machine instructions.

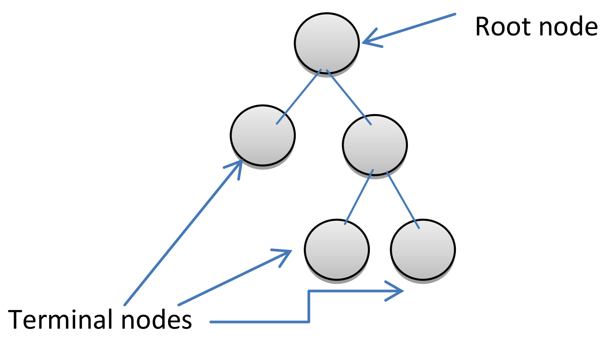
1. Integral type number: char ,int
2. Real type number: float, double
3. Void or nothing type : void

2) **Derived** data type: These are derived from the primitive data structures.

Array, Union, Structure, pointer

b) **Define the terms tree and forest.(1 mark each for definition of tree and forest)**

Tree: A tree is a nonempty collection of vertices and edges that satisfies certain requirements. A vertex is a simple object (node) that can have a name and carry other associated information. An edge is a connection between two vertices.



Forest: A forest is defined as set of trees. It can be represented by a binary tree.

c) Define Algorithm with example.(Definition 1 mark and example 1 mark)

The term algorithm refers to the sequence of instructions that must be followed to solve a problem. In other words, an algorithm is a logical representation of the instructions which should be executed to perform a meaningful task.

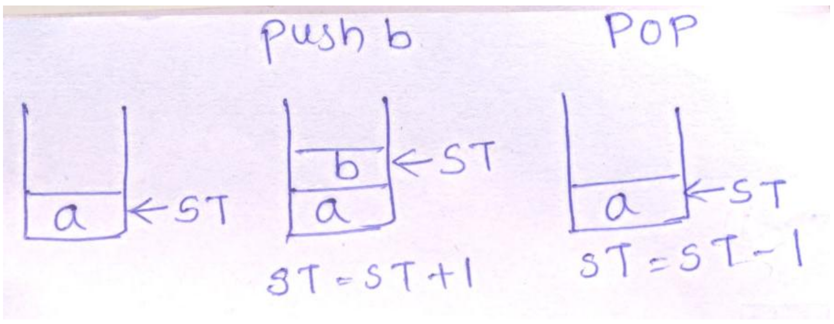
(Any other algorithm example can be considered)

```
Example: Algorithm to find largest of two numbers.
Step 1: Read the two numbers A and B.
Step 2: Compare values of A and B
         If A is greater than B then write message A is largest
        Otherwise
        Write B is largest
Step 4: stop.
```

d) State the operations of stack with example.(Listing 1 mark and example 1 mark)

Basic operations of stack are:
1) PUSH: Insert an element into a stack.
2) POP: Delete an element from stack.

Example:

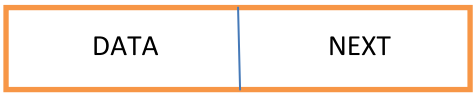


e) State any two application of Queue.(1 mark for each)
(Any other application can be considered)

1) the most useful application of queue s is the **simulation** of real world situation, so that it is possible to understand what happens in a real world in a particular situation without actually observing its occurrence.
2) Queues are useful in time sharing **computer system** where many users share the system simultaneously.
3) Queues are commercially used in **online business applications** for processing customer request in first in first serve manner.
4) An operating system always makes use of queue(ready queue, waiting queue) for scheduling processes or jobs.

f) Define node and pointer for a link list.(1 mark for each)

NODE: An element in the linked list is called as a node. The node contains two parts: Data and next. The data part contains actual information about the element and the next part contains a pointer to the next element or node in the list.



NODE

Pointer: Pointer is a container for storing address of next node. A linked list is a sequence of nodes which are connected to each other with pointer.

g) Define term related to binary Tree.(1/2 mark for each)

1) **Level:** level is a rank of tree hierarchy. The level of the root node is always at 0. The immediate children of root are at level 1 and their immediate children are at level 2 and so on.

2) **Depth:** Depth of the tree is the maximum level of any node in the tree. This is equal to the longest path from the root to any leaf node.

3) **Leaf Node:** leaf node is a terminal node of a tree. It does not have nodes connected to it.

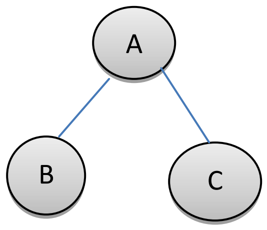
4) **Root Node:** it is the first node in the hierarchical arrangement of the tree and does not have parent node.

h) Describe siblings with the help of diagram.(2 mark)

Children of the same parent are called as siblings.

Example:

In this tree B, C are siblings



B. Attempt any two questions from the following.

a) What is data structure? Explain various operations of data structure.(Definition 2 mark, operations 2 mark)

Data structure: A data structure is the branch of computer science. Data structure is a particular way of storing and organizing a particular data in a computer so that it can be used efficiently or it is a mathematical or logical model of a particular organization of a data is called as data structures.

Data structure operations (Any four)

1) Traversing: Accessing each record exactly once so that certain items in the record may be processed.
2) Searching: finding the location of the record with the given key value or finding the locations of all records which satisfy one or more conditions.
3) Inserting: Adding a new record to the structure.
4) Deleting: Removing a record from the structure.
5) Sorting: Arranging the records in some logical order.
6) Merging: Combining the records in two different sorted files into a single sorted file.

b) Explain different approaches to design an algorithm.(2 mark for each approach)

There are two approaches for designing an algorithm.

Top down approach: A top down approach starts by identifying the major components of the system or program decomposing them into their lower level components and iterating until the desired level of module complexity is achieved.

Top down algorithm design is a technique for organizing and coding programs in which a hierarchy of modules is used, and breaking the specification down into simpler and simpler pieces. Each and every piece having a single entry and a single exit point, and in which control is passed downward through the structure without unconditional branches to higher levels of the structure. It tends to generate modules that are based on functionality, usually in the form of functions or procedures or methods.

Bottom up approach: A bottom up approach starts with designing the most basic or primitive components and proceeds to higher level components. Starting from the very bottom, the operations that provide a layer of abstraction are implemented. The operations of this layer are then used to implement more powerful operations and still higher layer of abstraction, until the stage is reached where the operations supported by the layer are those desired by the system.

c) Define stack and explain the various applications of stack.(Definition 2 mark and each application 1 mark)

Stack: A stack is a linear data structure in which insertion or deletion of element takes place at the same end. This end is called as **top** of stack.

Applications of stack: (any two)

1) Reversing a list: A simple example of stack application is reversal of a given list. It can be accomplished by pushing each character onto the stack as it is read. When the line is finished, characters are then popped off the stack- they come in reverse order.

2) Polish notations: the process of writing the operations of an expression either before their operands after them is called the polish notation. There are basically there types of notation for an expression 1) Infix 2) prefix and 3) postfix notation.

Conversion of infix expression into prefix and postfix form is done by using stack.

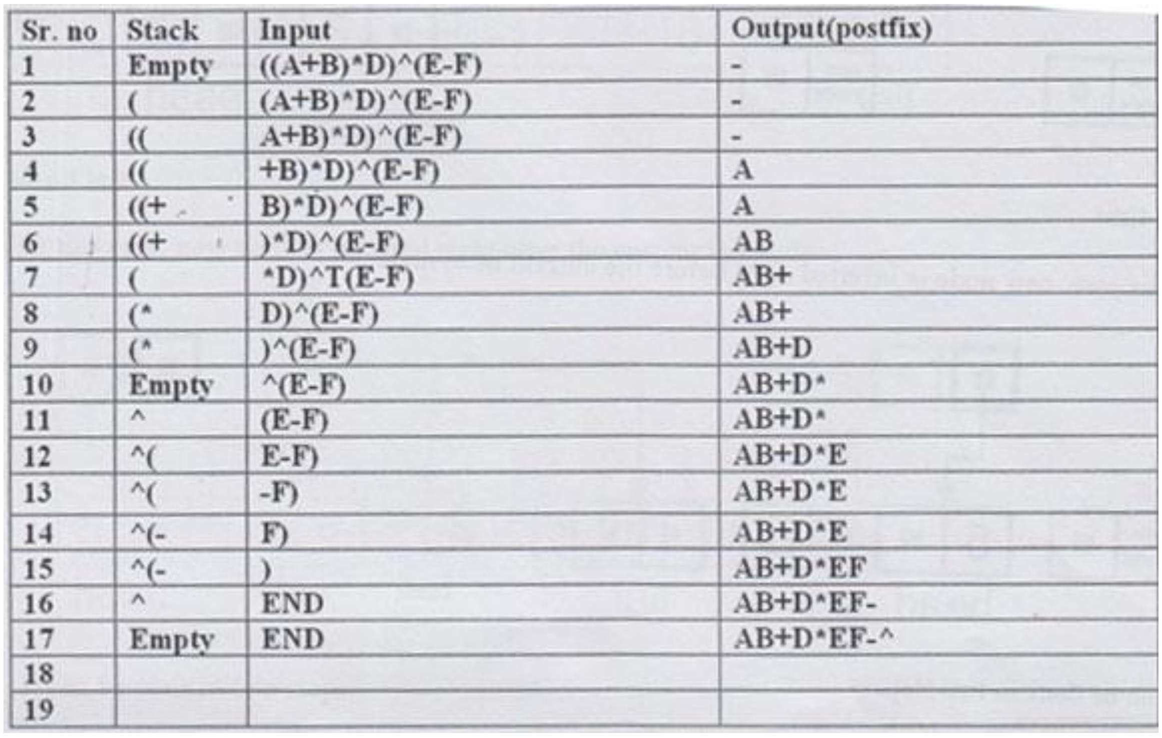
Evaluation of prefix and postfix expression is also done by using stack.

3) Recursion: Recursion is a process of expressing a function in terms of itself.

When recursive function isexecuted, the recursion calls are not implemented instantly. All the recursive calls are pushed on to the stack until the terminating condition is not detected. As soon as the terminating condition is detected, the recursion calls stored in the stack are popped and executed. The last call is executed first then second, then third and so on.

Q.2. Attempt any two question from the following:

a) Convert infix expression to postfix expression using stack with detail step expression:
$\left((A+B)*D\right)\uparrow(E-F)$**.(8 Marks)**

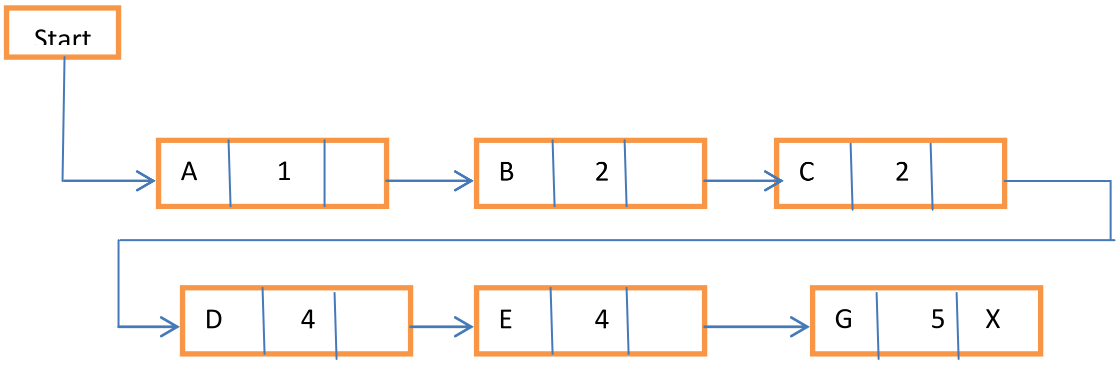


Sr. no	Stack	Input	Output(postfix)
1	Empty	((A+B)*D)^(E-F)	-
2	(	(A+B)*D)^(E-F)	-
3	((	A+B)*D)^(E-F)	-
4	((	+B)*D)^(E-F)	A
5	((+	B)*D)^(E-F)	A
6	((+	)*D)^(E-F)	AB
7	(	*D)^T(E-F)	AB+
8	(*	D)^(E-F)	AB+
9	(*	)^(E-F)	AB+D
10	Empty	^(E-F)	AB+D*
11	^	(E-F)	AB+D*
12	^(	E-F)	AB+D*E
13	^(	-F)	AB+D*E
14	^(-	F)	AB+D*E
15	^(-	)	AB+D*EF
16	^	END	AB+D*EF-
17	Empty	END	AB+D*EF-^
18			
19			

b) Explain array and linked list representation of priority queue.
(Each representation 2 marks and explanation 2 marks=8 marks)

A priority Queue is a collection of elements where each element is assigned a priority and the order in which elements are deleted and processed is determined from the following rules:
1) An element of higher priority is processed before any element of lower priority.
2) Two elements with the same priority are processed according to the order in which they are added to the queue.

1. **linked list representation:**
Each node in the list contains three items of information – an information field INFO, a priority number PRNO, and the link NEXT.
Node A will precede Node B in the list when A has higher priority than B or when both the nodes have same priority but Aadded to the list before B.

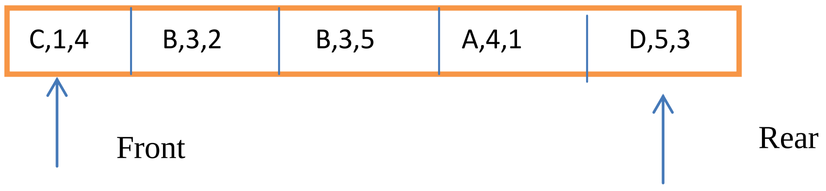


Above figure shows priority queue with 6 elements where element B& C have same priority numbers.

2. Array Representation:-

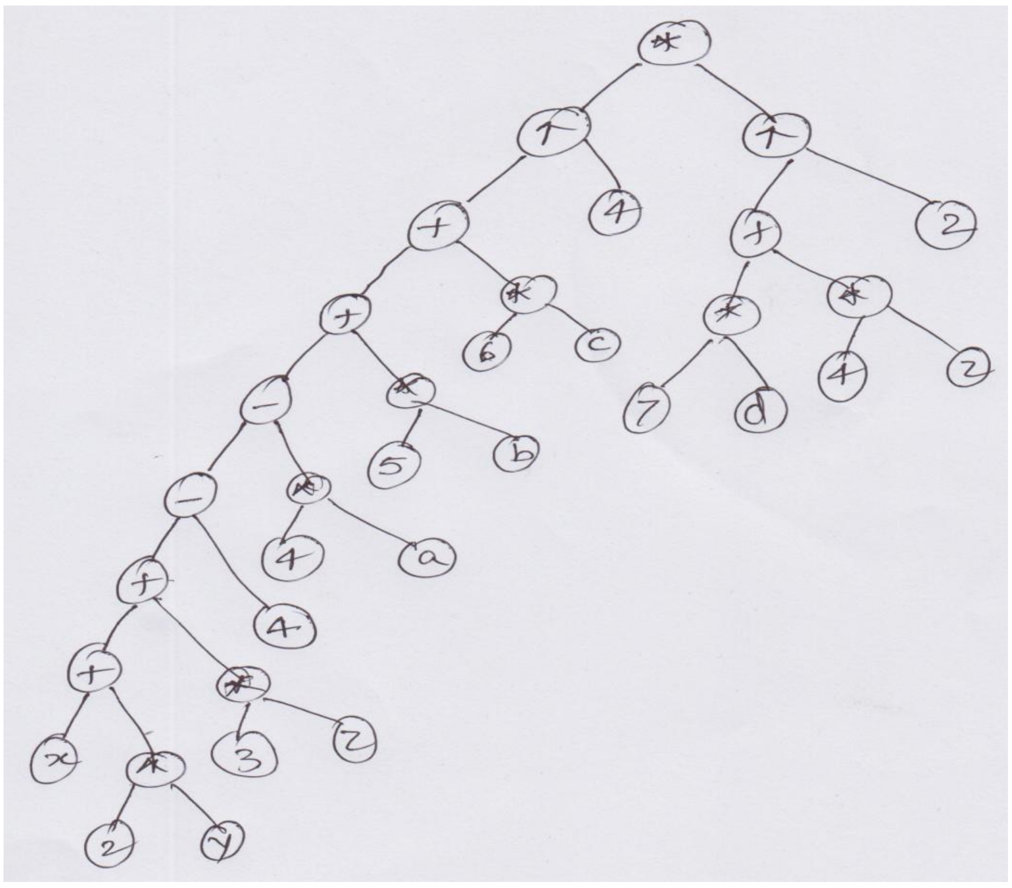
If an array is used to store the elements of priority queue then insertion is easy but deletion of elements would be difficult. This is because while inserting elements in the priority queue they are not inserted in an order. As a result, deleting an element with the highest priority would require examining the entire array to search for such an element. Moreover, an element in a queue can be deleted from the front end only. The array element of a priority queue can have the following structure.

```
Struct data
{
        Int item;
        Int priority;
        Int order;
};
```



c) Draw the tree structure for the following expression.(Correct tree structure 8 Marks)

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



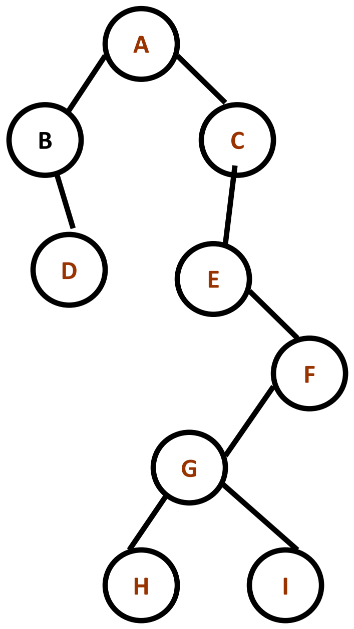
Q.3. Attempt any four questions from the following:

a) Write algorithm to traverse the tree in preorder with example.

(Algorithm 2 marks any example 2 marks)

Pre-order algorithm

Step 1: Visiting the root node.
Step 2: Traversing the left subtree in preorder (recursive call)
Step 3: Traversing the right subtree in preorder (recursive call)



A, B, D, C, E, F, G, H,I

b) Compare quick sort and radix sort with respect to working principal and time complexity.(2 marks for working principle and 2 marks for time complexity)

1. Working principle:

Quick sort:

The quick sort algorithm works as follows:

1. Select pivot element from the array elements
2. Re-arrange the elements in the array in such a way that all elements that are less than the pivot appear before the pivot and all elements greater than the pivot element comes after it.
3. After partitioning, the pivot is placed in its final position. This is called the **partition** operation.
4. Sort the two sub-arrays with above three steps till all the elements are placed in order.

The main task in the quick sort algorithm is to find the pivot element, which will partition the array into two halves.

Radix Sort:

1. Arrange 10 buckets for numbers or 26 buckets for alphabets.
2. Place each number in respective bucket depending on the position of digit in the number.
3. Rearrange numbers or alphabets from all buckets into an array.
4. Perform above steps 2 &3 till all elements are placed in sorted order.

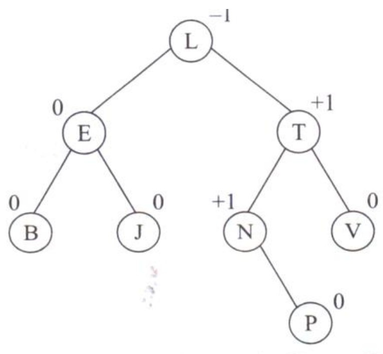
The number of passes will depend on the length of the number having maximum digits.

2. Complexity:
Quick sort:
average& best case:O($n\log n$)
worst case:O(n^2)

Radix sort:
O($n\log n$)

c) Explain height balance tree with diagram.
(Explanation 2 marks, any example with balancing factor 2 marks)

Height balanced tree is example of balanced binary tree. In height balanced tree, we attempt to keep each leaf node with the same distance from the root. To prevent unbalance of tree use balance indicator associated with each node. when the difference between height of left and right subtree is -1,0,1 then the tree is height balanced tree.
The balancing factor of a node in a binary tree can have 1, -1 or 0 depending on whether the height of its left subtree is greater than, less than or equal to the height of its right subtree. The balance factor of each node is indicated in figure.



Balance Factors in Binary Tree

d) Write an algorithm for Breath First search on graph.
(Algorithm 4 marks (Any example with algorithm steps can be considered))

Breadth-first search (BFS) is a graph search algorithm that begins at the specific node and explores all the neighboring nodes. Then for each of those nearest nodes, the algorithm explores their unexplored neighbor nodes, and so on, until it finds the goal.
That is, we start examining the node A and then all the neighbors of A are examined. In the next step we examine the neighbors of neighbors of A, so on and so forth

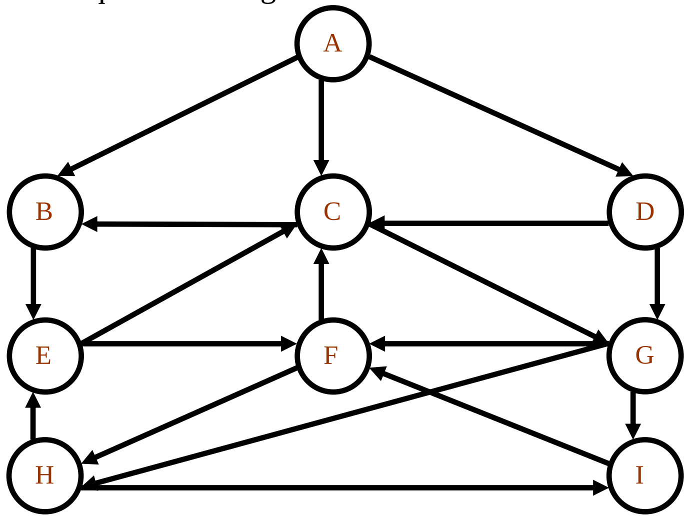
Algorithm for breadth-first search in a graph G beginning at a starting node A
Step 1: SET STATUS = 1 (ready state) for each node in G.
Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)
Step 3: Repeat Steps 4 and 5 until QUEUE is empty
Step 4:Dequeue a node N. Process it and set its STATUS = 3 (processed state).
Step 5:Enqueue all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

```
        [END OF LOOP]
Step 6: EXIT
```

OR

Example: Consider the graph G given below. The adjacency list of G is also given. Assume that G represents the daily flights between different cities and we want to fly from city A to H with minimum stops. That is, find the minimum path P from A to I given that every edge has length = 1.

Dequeue a node by setting FRONT = FRONT + 1 (remove the FRONT element of QUEUE) and enqueue the neighbors of A. Also add A as the ORIG of its neighbors, so



Adjacency Lists
A: B, C, D B: E C: B, G D: C, G E: C, F F: C, H G: F, H, I H: E, I I: F

Initially add A to QUEUE and add NULL to ORIG, so

FRONT = 1	QUEUE = A
REAR = 1	ORIG = \0

Dequeue a node by setting FRONT = FRONT + 1 (remove the FRONT element of QUEUE) and enqueue the neighbors of A. Also add A as the ORIG of its neighbors, so

FRONT = 2	**QUEUE = A B C D**
REAR = 4	**ORIG = \0 A A A**

Dequeue a node by setting FRONT = FRONT + 1 and enqueue the neighbors of B. Also add B as the ORIG of its neighbors, so

FRONT = 3	**QUEUE = A B C D E**
REAR = 5	**ORIG = \0 A A A B**

Dequeue a node by setting FRONT = FRONT + 1 and enqueue the neighbors of C. Also add C as the ORIG of its neighbors. Note that C has two neighbors B and G. Since B has already been added to the queue and it is not in the Ready state, we will not add B and add only G, so

FRONT = 4	**QUEUE = A B C D E G**
REAR = 6	**ORIG = \0 A A A B C**

Dequeue a node by setting FRONT = FRONT + 1 and enqueue the neighbors of D. Also add D as the ORIG of its neighbors. Note that D has two neighbors C and G. Since both of them have already been added to the queue and they are not in the Ready state, we will not add them again, so

FRONT = 5	**QUEUE = A B C D E G**
REAR = 6	**ORIG = \0 A AA B C**

Dequeue a node by setting FRONT = FRONT + 1 and enqueue the neighbors of E. Also add E as the ORIG of its neighbors. Note that E has two neighbors C and F. Since C has already been added to the queue and it is not in the Ready state, we will not add C and add only F, so

FRONT = 6	**QUEUE = A B C D E G F**
REAR = 7	**ORIG = \0 A AA B C G**

Dequeue a node by setting FRONT = FRONT + 1 and enqueue the neighbors of G. Also add G as the ORIG of its neighbors. Note that G has three neighbors F, H and I.

FRONT = 7	**QUEUE = A B C D E G F H I**
REAR = 10	**ORIG = \0 A AA B C G GG**

Since I is our final destination, we stop the execution of this algorithm as soon as it is encountered and added to the QUEUE. Now backtrack from I using ORIG to find the minimum path P. thus, we have P as
A -> C -> G -> I.

e) What is hashing? Explain any one hashing method.
(Definition 1 mark, any one method explanation 2 marks and example 1 mark)

Hash Function, h is simply a mathematical formula which when applied to the key, produces an integer which can be used as an index for the key in the hash table. The main aim of a hash function is that elements should be relatively randomly and uniformly distributed. Hash function produces a unique set of integers within some suitable range. Such function produces no collisions. But practically speaking, there is no hash function that eliminates collision completely. A good hash function can only minimize the number of collisions by spreading the elements uniformly throughout the array.

1. Division Method
Division method is the most simple method of hashing an integer *x*. The method divides *x* by *M* and then use the remainder thus obtained. In this case, the hash function can be given as
h(x) = x mod M
The division method is quite good for just about any value of *M* and since it requires only a single division operation, the method works very fast. However, extra care should be taken to select a suitable value for *M*.
Generally, it is best to choose *M* to be a prime number because making *M* a prime increases the likelihood that the keys are mapped with a uniformity in the output range of values. Then M should also be not too close to exact powers of 2. If we have, h (k) = x mod 2kthen the function will simply extract the lowest *k* bits of the binary representation of *x*

Example: Calculate hash values of keys 1234 and 5462.
Setting m = 97, hash values can be calculated as
h(1234) = 1234 % 97 = 70
h(5642) = 5642 % 97 = 16

2. Mid Square Method

Mid square method is a good hash function which works in two steps.
Step 1: Square the value of the key. That is, find k^2
Step 2: Extract the middle *r* bits of the result obtained in Step 1.
The algorithm works well because most or all bits of the key value contribute to the result. This is because all the digits in the original key value contribute to produce the middle two digits of the squared value. Therefore, the result is not dominated by the distribution of the bottom digit or the top digit of the original key value.

Example: Calculate the hash value for keys 1234 and 5642 using the mid square method. The hash table has 100 memory locations.
Note the hash table has 100 memory locations whose indices vary from 0-99. this means, only two digits are needed to map the key to a location in the hash table, so $r = 2$.
When k = 1234, k2 = 1522756, h (k) = 27
When k = 5642, k2 = 31832164, h (k) = 21
Observe that 3rd and 4th digits starting from the right are chosen.

3. Folding Method

The folding method works in two steps.
Step 1: Divide the key value into a number of parts. That is divide k into parts, *k1, k2, ...,kn,* where each part has the same number of digits except the last part which may have lesser digits than the other parts.
Step 2: Add the individual parts. That is obtain the sum of *k1 + k2 + .. + kn*. Hash value is produced by ignoring the last carry, if any.

Note that the number of digits in each part of the key will vary depending upon the size of the hash table.

Example: Given a hash table of 100 locations, calculate the hash value using folding method for keys- 5678, 321 and 34567.
Here, since there are 100 memory locations to address, we will break the key into parts where each part (except the last) will contain two digits.
Therefore,

Key	5678	321	34567
Parts	56 and 78	32 and 1	34, 56 and 7
Sum	134	33	97
Hash Value	34 (ignore the last carry)	33	97

f) Explain Atomic type with example.
(Explanation 2 marks, example 2 marks)

A data structure is required by a memory block which has two parts.

- Data storage
- Address storage.

An atomic type data is a data structure that contains only data items and not the pointers.
Such list of data items is maintained using a list node, containing pointer to these atomic nodes and types indicator indicating type of atomic node to which it point. Whenever a test node is inserted in the list, its address is stored in the next free element of the list of pointers.

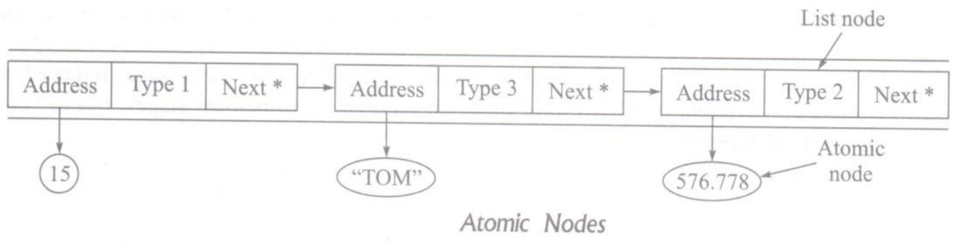


Figure shows a list of atomic nodes maintained using list of nodes. In each node, type represents the type of data stored in the atomic node to which the list node points. 1 stands for integer type, 2 for real number and 3 for character type or any different assumption can be made at implementation level to indicate different data types.

Q.4. Attempt any two questions from the following:

a) Write a program in 'C' language for selection sort.(Accurate program 08 marks)

(Note: program can be written differently depending on the same logic. It may be without separate function)Given solution is only code for separate function. Same logic from the code can be used in single program without function.

```
void selection()
{
printf("\n*****SELECTION SORT*****\n");
for(i=0;i<n;i++)
{
        for(j=i+1;j<5;j++)
                {
                If(a[i]>a[j])
                {
                temp=a[i];
                a[i]=a[j];
                a[j]=a[i];
                }
                }
}
printf("\n The sorted array=");
for(j=0;j<n;j++)
printf("\t %d",a[j]);
}
```

b) Write a menu driven program in 'C' language for Queue having menu: Store, retrieve and Display Queue.(Note: 2 marks insert-store,2 marks retrieve,2 marks display and 2 marks main function with menu)
(Single function with retrieval and display can be considered as code for retrieve and display function)

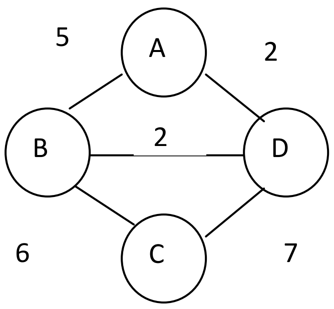
```
#include<stdio.h>
#include<conio.h>
#define max 10
int front=0,rear=-1,i,choice,c=1;;
```

```
char a[max];

void insert();
void display();
voidretrieve_q();
void exit();
void main()
{
clrscr();
printf("the front represents the first customer and rear represents the last\n");
while(1)
{
printf("1.insert\n");
printf("2.display\n");
printf("3.Retrive\n");
printf("4.exit\n");
printf("enter your choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
insert();
break;
case 2:
display();
break;
case 3:
retrieve_q();
break;
case4:
exit();
break;
default:
printf("\nwrong choice");
}
}
}
void insert()
{
char add;
printf("\nThis function adds a element  at the end of the queue");
if(rear==(max-1))
{
printf("\n queue is full");
}
else
{
printf("\nEnter new element to be inserted:");
scanf("%s",&add);
rear=rear+1;
a[rear]=add;
}
}
void retrieve_q()
{
if(rear==-1)
{
printf("\nqueue is empty");
}
else
{
printf("\nThe elements  retrieved from the queue are: ");
for(i=front;i<=rear;i++)
{
printf("%d element retrived is %d\n",c,a[i]);
c++;
}
}
return(0);
}

void display()
{
if(rear==-1)
{
printf("\nqueue is empty");
}
else
{
printf("\nThe elements  in the queue are: ");
for(i=front;i<=rear;i++)
{
printf("%c\n",a[i]);
}
}
return(0);
}
```

c) Explain shortest path algorithm with example.
(Explain 4 marks, any example 4 marks)

Algorithm:
Step 1: Decide source and destination nodes i.e. Vi,Vj.
Step 2: Find all possible paths from Vi to Vj.
Step 3: Calculate weight for each possible path listed in step 2.weight of path is sum of weight of edges along the path.
Step 4: Compare weight of all path and finalize minimum weight path as a shortest path between Vi to Vj.
Step 5: stop.

Example:



Step 1: Findshortest path between source and destination nodes between A and B.
Step 2: Find all possible paths from A & B
1.A-B
2.A-D-B
3.A-D-C-B
Step 3: Calculate weight for each possible path listed in step 2.weight of path is sum of weight of edges along the path.
1.A-B=5
2.A-D-B=4
3.A-D-C-B=15
Step 4: Compare weight of all path and finalize minimum weight path as a shortest path between A & B.
Selected path: A-D-B

Q.5. Attempt any four question from the following:

a) Explain Radix sort with example.(2 marks-Explain or algorithm& 2 marks on example)

Algorithm:

1. Arrange 10 buckets for numbers or 26 buckets for alphabets.
2. Place each number in respective bucket depending on the position of digit in the number.
3. Rearrange numbers or alphabets from all buckets into an array.
4. Perform above steps 2 &3 till all elements are placed in sorted order.

The number of passes will depend on the length of the number having maximum digits.

Suppose we want to sort the following set of three digits numbers:

Example:

Number	0	1	2	3	4	5	6	7	8	9
345						345				
654					654					
924					924					
123				123						
567								567		
472			472							
555						555				
808									808	
911		911								

Number	0	1	2	3	4	5	6	7	8	9
911		911								
472								472		
123			123							
654						654				
924			924							
345					345					
555						555				
567							567			
808	808									

Number	0	1	2	3	4	5	6	7	8	9
808									808	
911										911
123		123								
924										924
345				345						
654							654			
555						555				
567						567				
472					472					

Sorted array:123,345,472,555.567,654,808,911,924.

b) Write a program to sort an array of ten elements with bubble sort.
(4 marks for correct code)

(Note: program can be written differently depending on the same logic. It may be without separate function)given solution is only code for separate function. Same logic from the code can be used in single program without function.

```
bsort(int a[],int n)
{
inti,j,temp;
printf("\n\n BUBBLE SORT");
for(i=0;i<10;i++)
 {
for(j=0;j<10-i;j++)
 {
if (a[j]>a[j+1])
{
temp=a[j];
a[j]=a[j+1];
a[j+1]=temp;
}
}
}
}
```

c) Write the procedure for inserting and deleting an element from queue.(2 marks-insertion & 2 marks- deletion)

Insert procedure:
1. Check queue full i.e. check rear position. If it is full then display message and return to calling function. Otherwise go to step 2.
2. Increment Rear by 1.
3. Insert element with rear pointer.
4. Check position of front. If inserted element is first element then set front to 0.
5. Return to calling function

Delete procedure:
1. Check for queue empty i.e. check front position. If it is empty then display message and return to calling function. Otherwise go to step 2.
2. Check front and rear positions. If front and rear both are equal then show queue empty otherwise increment front by one.
3. Return to calling function

d) Define information, Next, Null pointer and Empty list w.r.t. link list.(each term-1mark)

Information: Each node in linked list contains information field that stores data.

Next: Each node in linked list contains next field that stores address of next node.

NULL: the last node of the linked list contains NULL value in next field of node to indicate end of list.

Empty list: A list without any node is empty list.

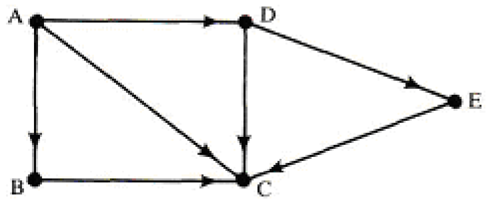
e) For the given Binary tree, perform in order preorder and post order traversal.
(Four marks for correct answer)

In-order: DBFEAGCLJKH
Pre-order: ABDEFCGHJLK
Post-order: DFEBGLKJHCA

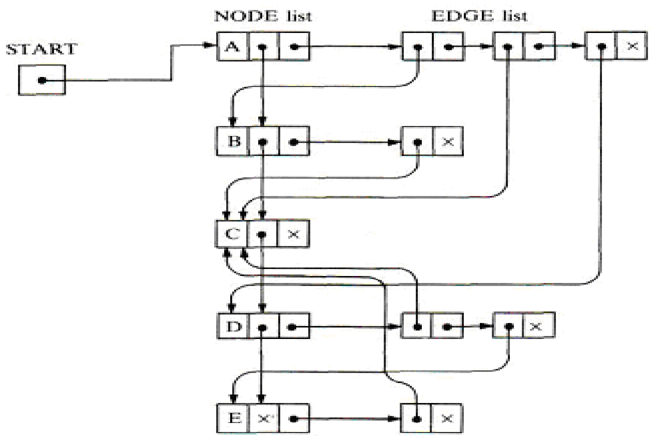
f) Explain the link representation of a Graph with suitable example.(2 marks-explanation & 2 marks-example

G is usually represented in memory by a linked representation, also called as an adjacent structure. In this each vertex of graph is represented as a node of linked list. Each node contains data and address field. Each node list contains 3 fields such as information to store the data, address field to store the address of next node and address field to store the address of adjacent node. Edge list contain node with 2 address field one pointing to a node in node list and other points to next node.

Consider the following the graph G.



Link representation of above graph:



Q.6. Attempt any four questions from the following:

a) Define Searching and explain linear search with example.(1 Mark on definition, 1 Mark-principle & 2 Mark-example)

Searching:

Searching is a process of finding an element in a given list of elements.

Linear search:-

Consideran array is used to store elements. For searching an element in array, compare theitem with each element in array one by one. That is, first we compare array [0] with the search item. If they are equal then display message and stop searchning.otherwise compare search item with next element in sequence. Perform comparison till element is found or search reaches to last element.

Example-

a: 11, 22, 30, 33, 40, 44, 55, 60, 661, 77, 80, 88, 99

Search item 40

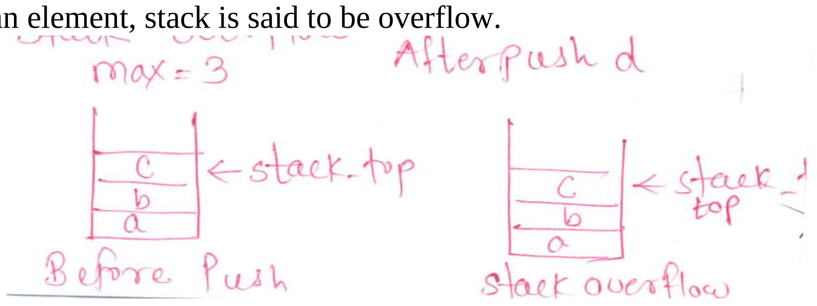
1> compare a[0]=11 with 40	not found
2> compare a[1]=22 with 40	not found
3> compare a[2]=30 with 40	not found
4> compare a[3]=33 with 40	not found
5> compare a[4]=40 with 40	**found**

Element found at4th position.

b) Write a procedure to push an element on stack. Also give meaning of stack overflow term.(2 M-push & 2 M-stack overflow)

This **procedure push** an ITEM onto a stack.
1. [stack already filled]
If TOP = MAXSTK, then: Print: OVERFLOW, and Return.
2. Set TOP:= TOP + 1. [increment TOP by 1]
3. Set STACK[TOP] := ITEM. [ITEM in new TOP position
4. Return.
The operation of adding (pushing) an item onto a stack is called as PUSH.

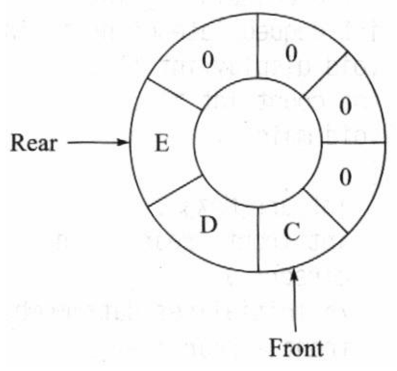
Stack overflow:-
While inserting an element push operation is called. When stack is full and push operation is called to insert an element, stack is said to be overflow.



c) Draw and explain circular Queue in detail.(define with diagram-2mrks& insert delete operations 2 marks.)

Circular queues are the queues implemented in circular form rather than in a straight line. **Circular queues overcome the problem of unutilized space in linear queue implemented as an array.**

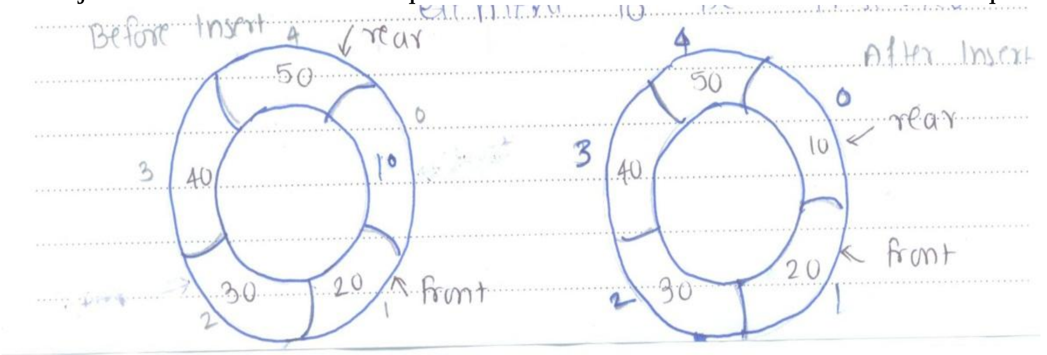
Circular queue contains two pointers such as front and rear. Rear is used to insert an element and front is used to delete an element.



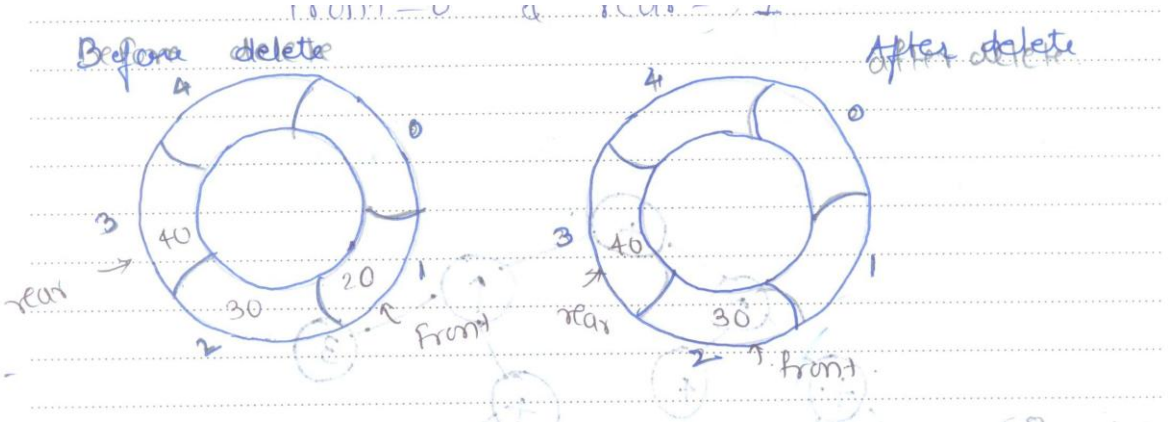
Any relevant explanation can be considered.

Operations on circular queue:

1. **Insert:-**Insert operation in circular queue inserts an element at the position pointed by rear pointer. Before inserting an element check for circular queue full. If front and rear pointers are adjacent to each other then queue is full and we cannot insert element in queue.

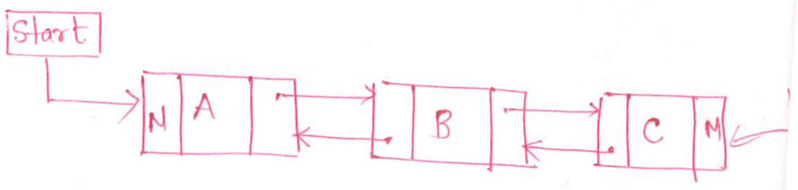


2. Delete:-delete operation is used to delete an element pointed by front pointer. Before deleting check for queue empty.



d) **With suitable diagram, explain 'searching' of a node in Doubly Linked List.(2 M-diagram & 2 M-searching explanation)**

Diagram:-



Procedure for searching :(student may explain with any doubly linked list diagram as shown above)

Search element =C

1. Set ptr=start.ptr points to first node.
2. Compare information field of node 1 with search element. If both are equal then display message and stop searching otherwise goto step 3.
3. Increment ptr to point next node and follow step 2.

4. Traverse the list starting from node 1 to the last node till element found or search reaches to last node.

e) Define following term with respect to graph: (each term- 1 Mark)

i. In-degree of anode
ii. Directed graph
iii. Weighted graph
iv. Predecessor

A **directed graph** G, called a graph in which each edge e in G is assigned a direction, or in other words, each edge e is identified with an ordered pair (u, v) of nodes in G.

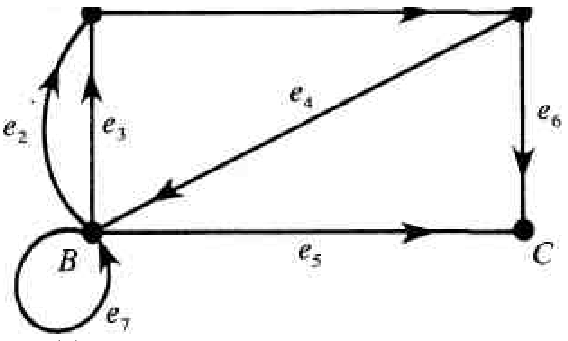
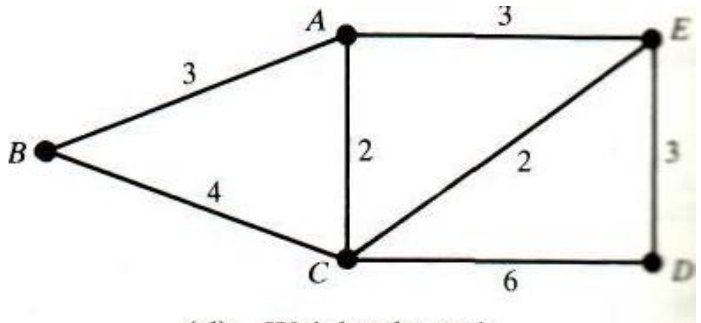


Fig (a)

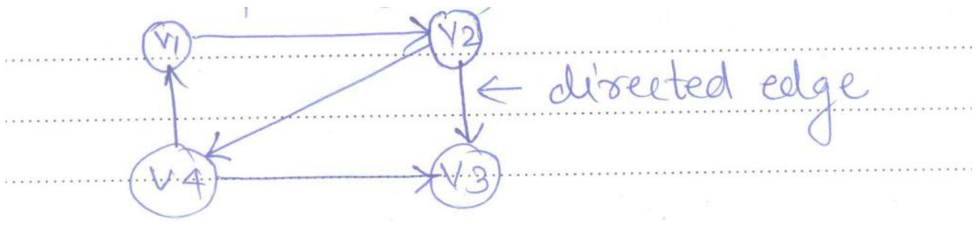
In degree: The in degree of a vertex is the number of edges for which v is head i.e. number of edged coming towards that vertex.
In above figure in-degree of C is 2.

Weighted Graph
If the weight is assigned to each edge of the graph then it is called as Weighted graph.



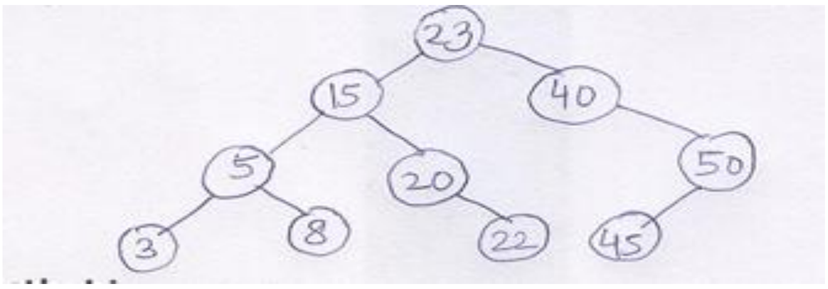
(d) Weighted graph.

Predecessor: Suppose G is a directed graph with a directed edge e = (u, v).
u is a predecessor of v, and v is a successor or neighbor of w.
In figure below, edge (V1-V2) indicates V1 as predecessor

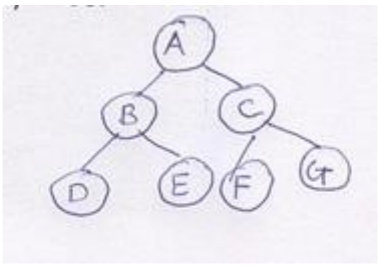


f) Enlist the types of Binary Tree and give their example.
(Any four binary tree types=each 1 Mark)

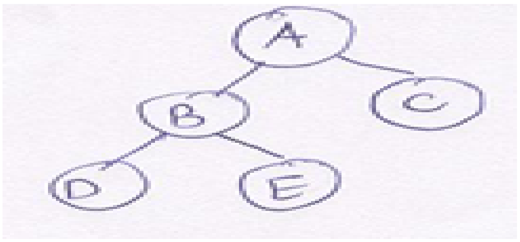
1. **Binary Search tree:-**It is a tree where all the element values of left sub tree are less than the root of tree and all the element values of right sub tree are greater than the root.



2. **Complete binary tree:** It is a tree where all leaf nodes are at the same level.



3. **Strictly binary tree:** it is a tree where each node from the tree is either a leaf node or it has two child nodes.



4. **Expression binary tree:** It is a tree where it represents mathematical expression with operator as a root and operands as child nodes.