WINTER– 17 EXAMINATION

Subject Name: Data Structure Using 'C' Model Answer **Subject Code:** 17330

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. No.	Sub Q. N.	Answer	Marking Scheme
1.	A)	**Attempt any six of the following:**	**12 Marks**
	a)	**List any four operations performed on data structure.**	**2M**
	Ans:	**Operations on Data Structure:-** • Insertion • Deletion • Searching • Sorting • Traversing • Merging	**(Any four operations: ½ mark each)**
	b)	**Define the term algorithm. Also list approaches to design an algorithm.**	**2M**
	Ans:	**Algorithm**: It is sequence of steps/instructions that must be followed to solve a problem. In other words, an algorithm is a logical representation of the steps/instructions which should be executed to perform a meaningful task. **Approaches to design an algorithm:** **1.** Top-down approach 2. Bottom-up approach:	**(Definition of algorithm: 1 mark, Two approaches to design an algorithm: ½ mark each)**

c) State any two differences between linear search and binary search. **2M**

Ans:

Linear Search	Binary Search
Key element is compared with each element in list	Key element is compared with mid element only
Simplest method of searching	Comparatively difficult method of searching
Easy to implement	Comparatively difficult to implement
Given list of numbers can be sorted or unsorted order	Given list of numbers must be in sorted order
Linear search only requires equality Comparisons.	Binary search requires an ordering comparison.
Linear search has complexity O(n).	Binary search has complexity O(log n).
Linear search is too slow to be used with large lists due to its O(n) average case performance.	Binary search is considered to be a more efficient method that could be used with large lists.
Linear search only requires sequential Access.	Binary search requires random access to the data.
Efficient method for less no. of elements	Efficient method when elements are large in numbers.

(Any two points of differences: 1 mark each)

d) Define the terms pointer and NULL pointer. **2M**

Ans:

Pointer: It is a variable which holds address of next node in a list.

OR

Pointer: It is a variable which holds a memory address of variable.

NULL pointer: It is used to specify end of the list. The last element of list contains NULL pointer to specify end of list.

OR

Null Pointer: It is a variable which does not hold any memory address i.e. it is pointing nothing.

(Definition of pointer & NULL pointer: 1 mark each)

e) Define tree. State its two types. **2M**

Ans:

Definition of tree: Tree is non-linear data structure with finite number of nodes in which there is one special node called root node and remaining nodes are divided into disjoint set which form the sub trees of the main tree.

Types of tree:

- General tree
- Binary tree
- Binary Search tree
- Expression tree

(Definition of tree: 1 mark, Any two types: ½ mark each)

	f)	**Give complexity of following methods:** **i) Bubble sort** **ii) Radix sort** **iii) Linear search** **iv) Binary search.**	**2M**
	Ans:	**i)** Bubble Sort: **$O(n^2)$** **ii)** Radix sort: **O(nk)** iii) Linear search: **O(n)** **iv)** Binary search : **O(log n)**	**(Complexity of each method: ½ mark each)**
	g)	**State any two applications of graph.**	**2M**
	Ans:	**Applications of graphs:** 1. To represent road map 2. To represent circuit or networks 3. To represent program flow analysis 4. To represent transport network 5. To represent social network 6. Neural networks	**(Any two applications: 1 mark each)**
	h)	**Define following terms:** **i) Height of tree** **ii) Degree of a node.**	**2M**
	Ans:	i) **Height of Tree**: The height of a tree is the maximum level of any node in the tree. ii) **Degree of Node:** It is defined as maximum number of child nodes of any node. **OR** Degree of node is the number of nodes connected to a particular node.	**(Definition of Height of tree & Degree of node : 1 mark each)**
	B)	**Attempt any two of the following:**	**8 Marks**
	a)	**Describe with example, time complexity and space complexity of an algorithm.**	**4M**

	Ans:	**{**Note: Any Relevant example shall be considered}**
		Time Complexity: Time complexity of program or algorithm is amount of computer time that it needs to run to completion. To measure time complexity of an algorithm we concentrate on developing only frequency count for key statements.
		Example:
		#include<stdio.h>
		void main ()
		{
		int i, n, sum, x;
		sum=0;
		printf("\n Enter no of data to be added");
		scanf("% d", &n);
		for(i=0 ; i<n; i++)
		{
			(Explanation of time complexity: 2 marks, space complexity: 2 marks)

```
        scanf("%d" , &x);
        sum=sum+x;
      }
    printf("\n Sum = %d ", sum);
    }
```

Calculation of Computational Time:

Statement	Frequency	Computational Time
sum=0	1	t_1
printf("\n Enter no of data to be added")	1	t_2
scanf("% d", &n)	1	t_3
for(i=0 ; i<n; i++)	n+1	$(n+1)t_4$
scanf("%d" , &x)	n	nt_5
sum=sum+x	n	nt_6
printf("\n Sum = %d ", sum)	1	t_7

Total computational time= $t_1+t_2+t_3+(n+1)t_4 +nt_{6+}nt_5+t_7$

$T= n(t_4+t_5+t_6)+ (t_1+t_2+t_3+t_4+t_7)$

For large n , T can be approximated to

$T= n(t_4+t_5+t_6)= kn$ where $k= t_4+t_5+t_6$

Thus T = kn or $T \propto n$

Space complexity: Space complexity of a program/algorithm is the amount of memory that it needs to run to completion. The space needed by the program is the sum of the following components:-

Example: - additional space required when function uses recursion.

```
void main()
{ int i,,n,sum ,x;
sum=0;
 printf("\n enter no of data to be added");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
 { scanf( "%d",&x);
 sum=sum+x;
 }
printf("\n sum =%d", sum) }
```

Space required to store the variables i,,n,sum and x=2+2+2+2=8 (int requires 2 bytes of memory space)

	b)	**Perform radix sort on the following list to arrange all array elements in ascending order:** **18,253,1000,2,80,75,58**	**4M**
	Ans:	Pass1: In this pass arrange the element according to unit place.	**(Four correct Pass: 1 mark each)**

Element	0	1	2	3	4	5	6	7	8	9
18									18	
253				253						
1000	1000									
2			2							
80	80									
75						75				
58									58	

Output of 1st pass: 1000,80,2,253,75,18,58

Pass 2: In this pass arrange the element according to tens place.

Element	0	1	2	3	4	5	6	7	8	9
1000	1000									
80									80	
2	2									
253						253				
75								75		
18		18								
58						58				

Output of 2nd pass: 1000, 2, 18,253, 58,75,80

Pass 3: In this pass arrange the element according to hundred's place.

Element	0	1	2	3	4	5	6	7	8	9
1000	1000									
2	2									
18	18									
253			253							
58	58									
75	75									
80	80									

Output of 3rd pass: 1000, 2, 18, 58, 75, 80,253

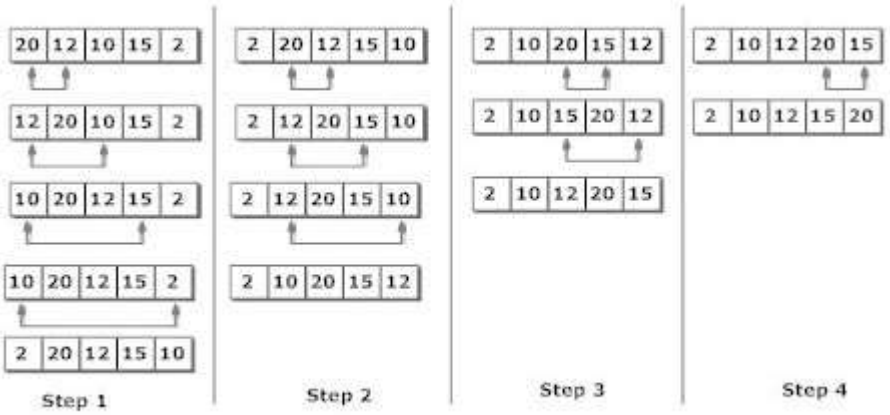
Pass 4: In this pass arrange the element according to thousand's place.

Element	0	1	2	3	4	5	6	7	8	9
1000		1000								
2	2									
18	18									
58	58									
75	75									
80	80									
253	253									

Output of 4th pass: 2,18,58,75,80,253,1000

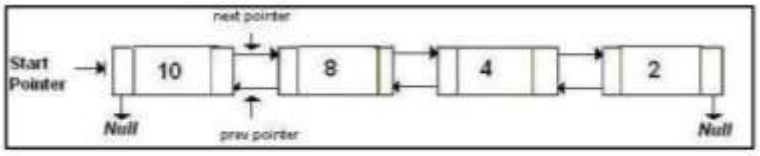
Elements in ascending orders: 2,18,58,75,80,253,1000

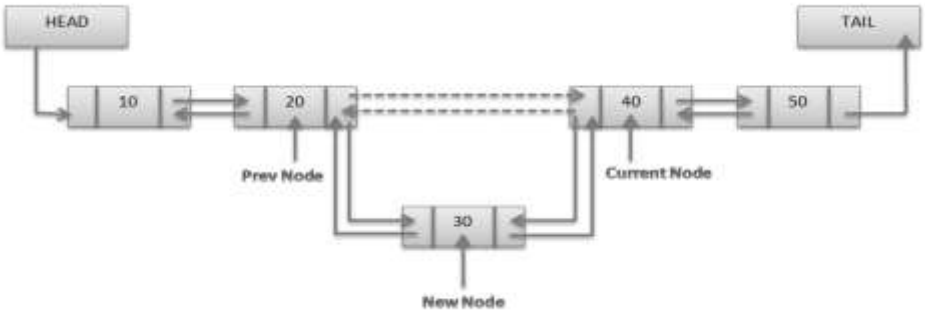
	c)	**Define following terms:** **1) Priority Queue** **2) Dequeues** **3) Queue as an Abstract Data Type** **4) Empty Queue.**	**4M**
	Ans:	1) **Priority Queue:** A priority Queue is a collection of elements where each element is assigned a priority and the order in which elements are added into the queue. 2) **Dequeues:** Dequeue is a special type of data structure in which insertions and	**(Definition of each terms: 1 mark each)**

		deletions will be done either at the front end or at the rear end of the queue. 3) **Queue as an Abstract Data Type:** Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends. In a queue data structure, adding and removing of elements are performed at two different positions. The insertion is performed at one end and deletion is performed at other end. In a queue data structure, the insertion operation is performed at a position which is known as 'rear' and the deletion operation is performed at a position which is known as 'front'. In queue data structure, the insertion and deletion operations are performed based on **FIFO (First In First Out)** principle. 4) **Empty Queue:** If no element is present inside a queue & front & rear is set to -1 then queue is said to be empty.	
2.		**Attempt any four of the following:**	**16 Marks**
	a)	**Describe working of selection sort method with suitable example.**	**4M**
	Ans:	**(**Note: Any relevant example can be considered**)** **Working of Selection sort:** Selection Sort algorithm is used to arrange a list of elements in a particular order (Ascending or Descending). In selection sort, the first element in the list is selected and it is compared repeatedly with remaining all the elements in the list. If any element is smaller than the selected element (for ascending order), then both are swapped. Then we select the element at second position in the list and it is compared with remaining all elements in the list. If any element is smaller than the selected element, then both are swapped. This procedure is repeated till the entire list is sorted. Example: Consider the array elements 20,12, 10, 15, 2  Figure: Selection Sort In first step 20 (first position) is compared with 12 as 20> 12 swap numbers. Now 12 is compared with 10 as 12>10 swap numbers. Now 10 is compared with next element 15 as 10< 15 no need to interchange. Again 10 is compared with 2 as 10>2 swap the numbers. So after first iteration the smallest number is placed at first position. Repeat this procedure till all numbers get sorted.	**(Working of selection sort: 2 marks, Example: 2 marks)**

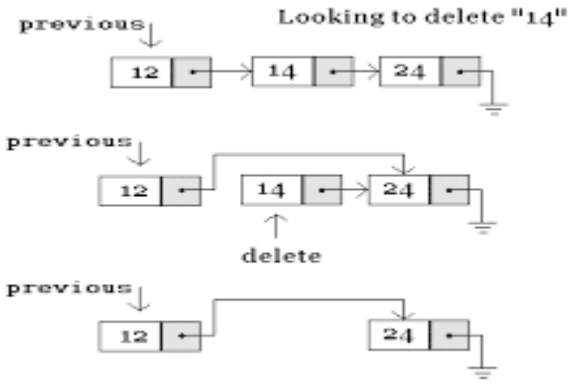
	b)	**Write a 'C' program to display Fibonacci series using recursive function.**	**4M**

	Ans:	`#include<stdio.h>`
		`#include<conio.h>`
		`int Fibonacci(int);`
		`int main()`
		`{`
		`  int n, i = 0, c;`
		`  scanf("%d",&n);`
		`  printf("Fibonacci series\n");`
		`  for ( c = 1 ; c <= n ; c++ )`
		`  {`
		`  printf("%d\n", Fibonacci(i));`
		`  i++;`
		`  }`
		`  getch();`
		`  return 0;`
		`}`
		`int Fibonacci(int n)`
		`{`
		`  if ( n == 0 )`
		`  return 0;`
		`  else if ( n == 1 )`
		`  return 1;`
		`  else`
		`  return ( Fibonacci(n-1) + Fibonacci(n-2) );`
		`}`
			(Correct logic: 2 marks, correct syntax: 2 marks)

	c)	**Describe with example advantage of doubly linked list over linear linked list.**	**4M**
	Ans:	**Advantages of doubly linked list over singly linked list:** 1. Traversal is done in both the directions. In singly linked list only one pointer is used so we can traverse in only one direction. But in doubly linked list we use two pointers next and previous so it easy to traverse in both direction.  **Fig: Doubly Linked List** 2. Insertion and deletion can be easily performed. In doubly linked list we can easily add new node and delete node in doubly linked list. We need to set the previous and next pointer in the list accordingly.	**(Any two advantages description: 2 marks each)**



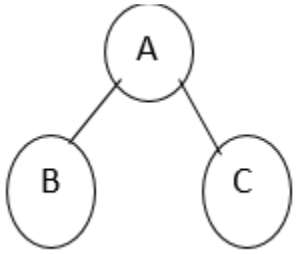
Insertion operation



Deletion operation

3. Searching an element is faster.
   In doubly linked list we can traverse anywhere using next and previous pointer so it is easy to search the element in list.

4. It is easy to reverse the linked list. In doubly linked list using previous pointer we can easily reverse the list by traversing in backward direction.

	d)	**With suitable example, describe preorder traversal of tree. Also write algorithm for preorder traversal.**	**4M**
	Ans:	**Algorithm for Preorder Traversal:** Step 1: Visit root node. Step 2: Visit left subtree in preorder. Step 3: Visit right subtree in preorder. **Example:**  **Preorder traversal is**: A, B, C. In this traversal method 1st process root element then left subtree & then right subtree.	**(Example of Preorder traversal: 2 marks, Algorithm of Preorder traversal: 2 marks)**

	e)	**Write a 'C' program to insert an element in a queue.**	**4M**

	Ans:	**(**Note: Any relevant logic can be considered**)**
		#include <stdio.h>
		#include<conio.h>
		#define MAX 10
		int queue_array[MAX];
		int rear = - 1;
		int front = - 1;
		void main()
		{
		int add_item;
		clrscr();
		if (rear == MAX - 1)
		printf("Queue Overflow \n");
		else
		{
		if (front == - 1)
		{
		front = 0;
		}
		printf("Insert the element in queue : ");
		scanf("%d", &add_item);
		rear = rear + 1;
		queue_array[rear] = add_item;
		printf("Element in queue is inserted Successfully \n");
		}
		}
			(Correct logic: 2 marks, correct syntax: 2 marks)

	f)	**Write a 'C' program to perform bubble sort on array of size N.**	**4M**

	Ans:	**(**Note: Any relevant logic can be considered**)**
		#include <stdio.h>
		void main()
		{
		int a[10], i,j,n,temp;
		clrscr();
		printf("\nEnter size of an array: ");
		scanf("%d", &n);
		printf("\nEnter elements of an array:\n");
		for(i=0; i<n; i++)
		scanf("%d", &a[i]);
		for(i=0; i<n; i++)
		for(j=0; j<n-1-i; j++)
		if(a[j] > a[j+1])
			(Correct logic: 2 marks, correct syntax: 2 marks)

```
{
temp = a[j];
a[j] = a[j+1];
a[j+1] = temp;
}
printf("\n\n Elements After sorting:\n");
for(i=0; i<n; i++)
printf("\n%d", a[i]);
getch();
}
```

3. **Attempt any four of the following :** **16 Marks**

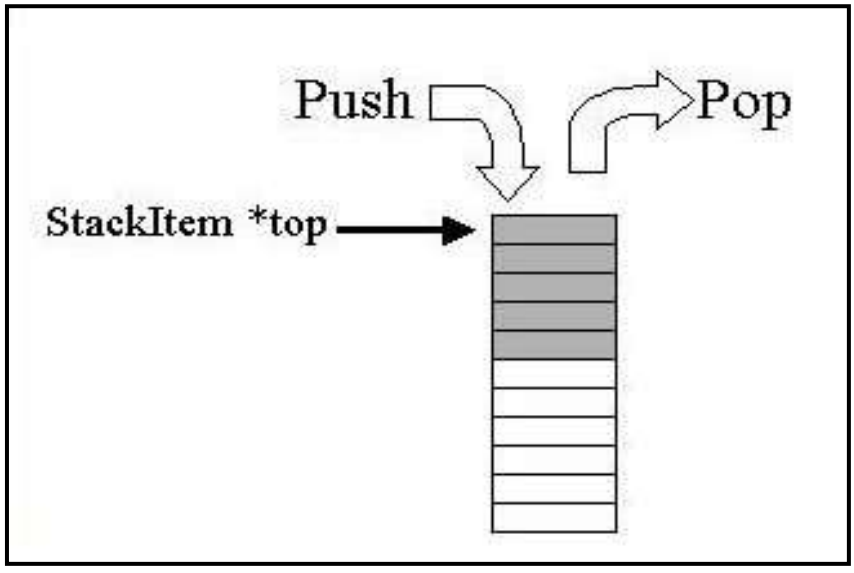
a) **Describe stack as an abstract data type.** **4M**

Ans: **(**Note: Any representation of an ADT containing elements and operation shall be considered**)** **(Description:2marks, stack elements:1 mark ,operations:1 mark)**

Stack is a linear data structure in which data elements are stored in a specific sequence. It works if LIFO manner i.e. Last In First Out. It has one pointer called as top. Data elements are push and pop using top pointer inside stack. Stack as an abstract data type contains stack elements and operations performed on it.

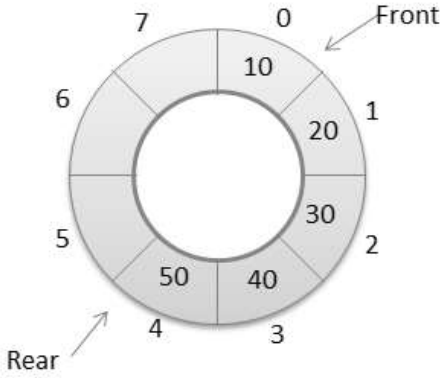
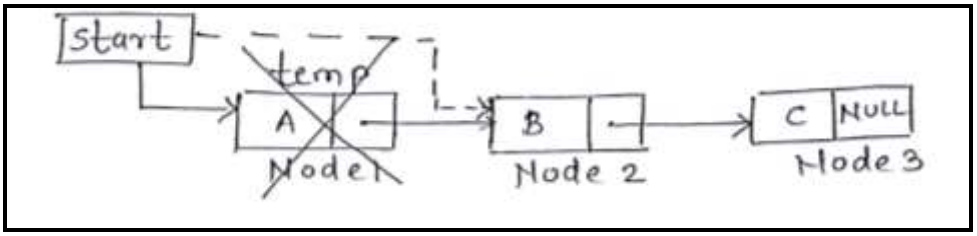
Stack elements are: Array in which data elements are stored and stack top pointer to perform operations on stack.

Diagram:



Stack operations include:

- **Initialize**: set stack to empty
- Checking whether **stack is empty** or not
- Checking if **stack is overflow/full** or not
- Insert a new element. This operation is called as **push.**
- Delete an element from stack. This operation is called as **pop**.

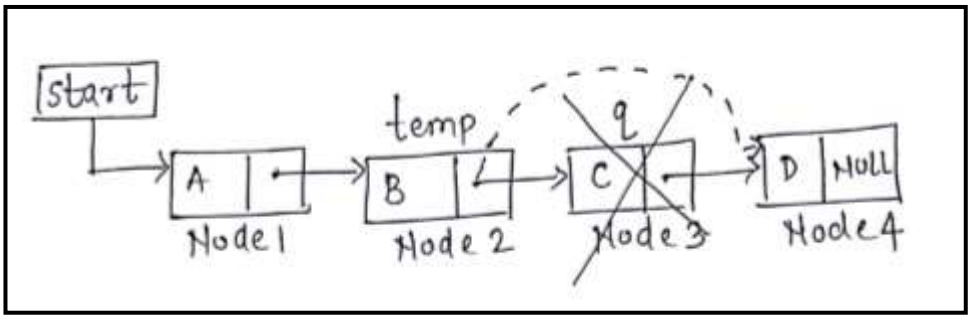
	b)	**Define circular queue. Also describe advantage of circular queue over linear queue.**	**4M**
	Ans:	• A queue, in which the last node is connected back to the first node to form a cycle, is called as circular queue. • It has rear pointer to insert an element and front pointer to delete an element. It works in FIFO manner where first inserted element is deleted first.  **Advantage of circular queue over linear queue:** **1.** In linear queue, before insertion of element 'queue full' condition is checked. If rear pointer points to the maximum-1 position then queue is full even though front pointer is greater than 0. If front pointer is greater than 0 that means space is available at the beginning in which new element can be stored. If rear pointer indicates queue full then insertion cannot be done. So it leads to wastage of space. **2.** Circular queue has advantage of utilization of space over linear queue**.** Circular queue is full only when there is no empty position in a queue. Before inserting an element in circular queue front and rear both the pointers are checked. So if it indicates any empty space anywhere in a queue then insertion takes place.	**(Definition:2 marks, description of advantage :2marks)**
	c)	**Describe how to delete a node from linear linked list.**	**4M**
	Ans:	**(**Note: Correct algorithm or program shall be considered. Any one deletion shall be considered**)** In a linear linked list, a node can be deleted from the beginning of list, from in between positions and from end of the list. **Delete a node from the beginning:-**  	**(Correct description : 4 marks)**

Node to be deleted is node1.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.

OR

Step 1: Create temporary node 'temp'.
Step 2: Assign address of first node to 'temp' pointer.
Step 3: Store address of second node (temp->next) in header pointer 'start'.
Step 4: Free temp.

Delete a node from in between position:-

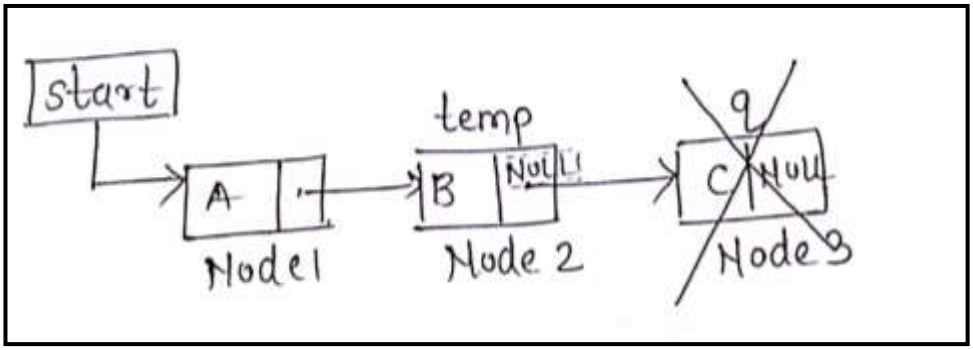


Node to be deleted is node3.Create a temporary node as 'temp' and 'q'. Set 'temp' node with the address of first node. Traverse the list up to the previous node of node 3 and mark the next node (node3) as 'q'. Store address from node 'q' into address field of 'temp' node. Then delete 'q' pointer with free function. Deleting 'q' pointer deletes the node 3 from the list.

OR

Step 1: Create temporary node 'temp', 'q'.
Step 2: Assign address of first node to 'temp' pointer.
Step 3: Traverse list up to previous node of node to be deleted.
Step 4: Mark the node to be deleted 'q'.
Step 5: Store address from node 'q' in address field of 'temp' node (temp->next=q->next).
Step 6: Free q.

Delete a node from the end:-



Node to be deleted is node 3.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

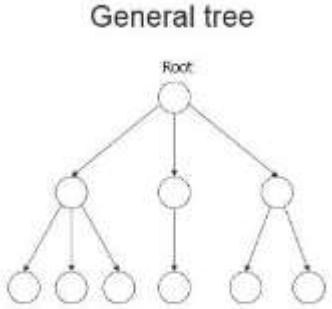
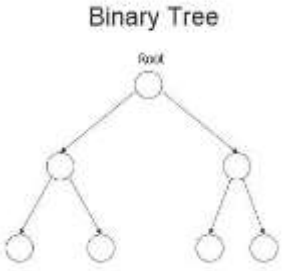
node and mark the last node as ‘q’. Store NULL value in address field of ‘temp’ node and then delete ‘q’ pointer with free function. Deleting q pointer deletes the last node from the list.

OR

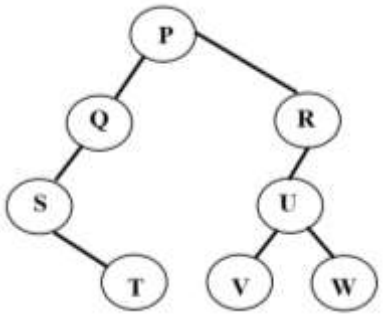
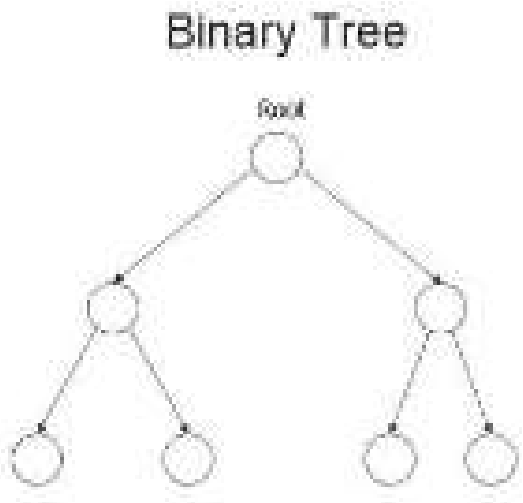
Step 1: Create temporary node ‘temp’,’q’.
Step 2: Assign address of first node to ‘temp’ pointer.
Step 3: Traverse list upto second last node.
Step 4: Mark last node’s address in node ‘q’.
Step 5: store NULL value in address field of second last node (temp->next).
Step 6: Free q.

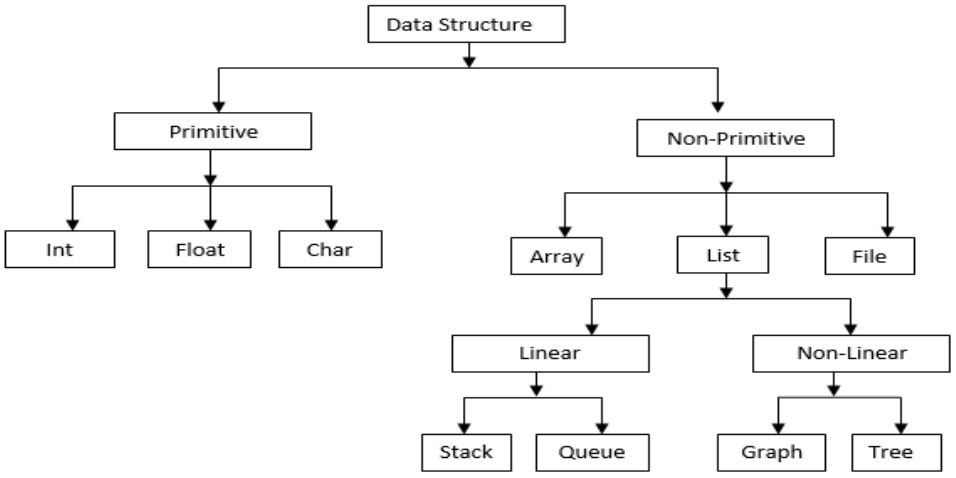
d) Differentiate between general tree and binary tree(Any four points) — **4M**

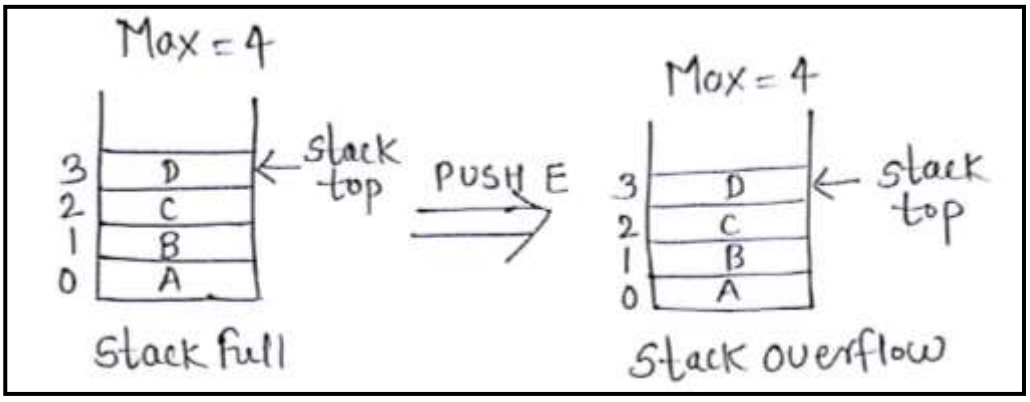
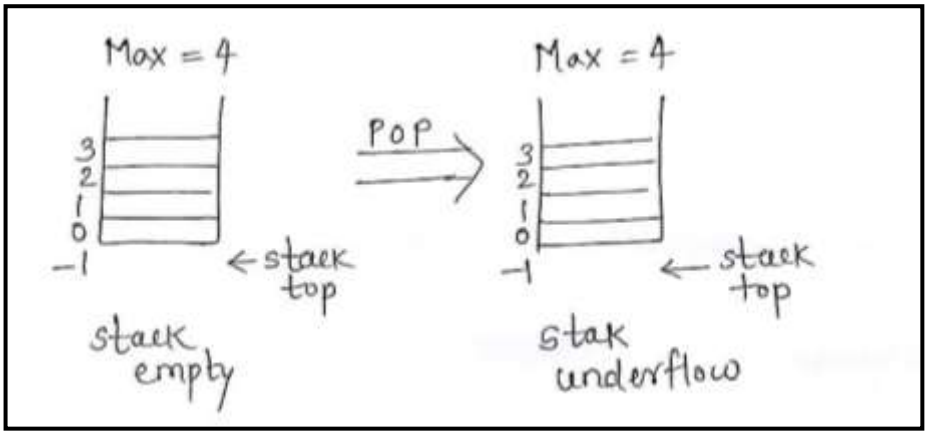
Ans:

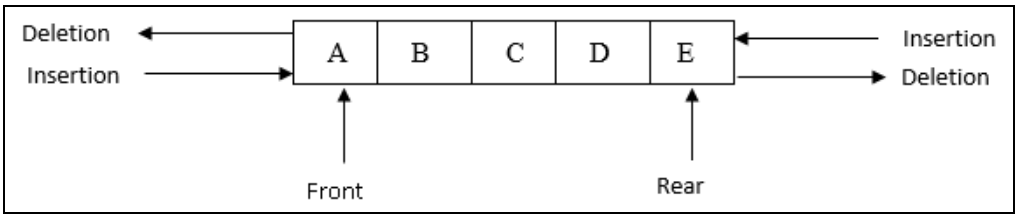
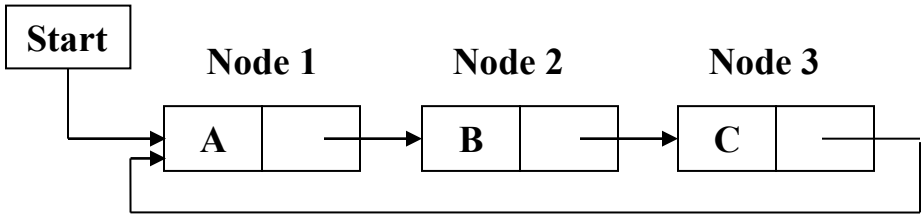
Sr.no	General Tree	Binary Tree
1	A general tree is a **data structure** in that each node can have infinite number of children	A Binary tree is a data structure in that each node has at most **two nodes** left and right
2	In general tree, root has **in-degree 0** and maximum **out-degree n**.	In binary tree, root has **in-degree 0** and maximum **out-degree 2**.
3	In general tree, each **node** have in-degree **one** and maximum out-degree **n**.	In binary tree, each node have in-degree **one** and maximum out-degree **2**.
4	**Height** of a general tree is the length of longest path from root to the leaf of tree. Height(T) = {**max**(height(child1) , height(child2) , … height(child-n)) +1}	Height of a binary tree is : Height(T) = { **max** (Height(Left Child) , Height(Right Child) + 1}
5	Subtree of general tree are **not ordered**	Subtree of binary tree is **ordered**.
		

(Any four points:1mark each)

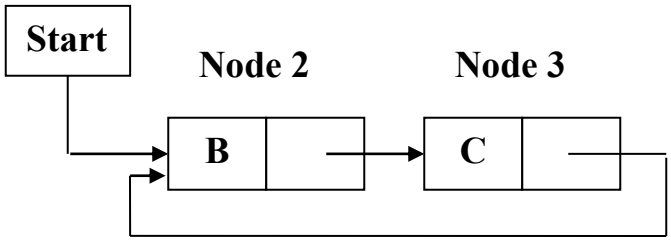
	e)	**Define binary tree. Traverse the following tree in inorder, preorder and postorder.** 	**4M**
	Ans:	**Binary tree:** A Binary tree is a nonlinear data structure in which each non-leaf node can have maximum two child nodes as left child ad right child.  **Tree traversal:-** **Inorder- S T Q P V U W R** **Preorder:- P Q S T R U V W** **Postorder:- T S Q V W U R P**	**(Definition:1 mark, Each Traversal: 1 mark)**
	f)	**Describe concept of hashing with example. Also describe problem of collision in hashing.**	**4M**
	Ans:	Hashing is the process of mapping large amount of data item to a smaller table with the help of hashing function. Hashing is a technique used to compute memory address for performing insertion, deletion and **searching** of an element using hash function. Hash address is calculated using division method, middle square method and Folding method. H(65)=5 Both key's hash address is 5, so we cannot map both the keys to same location. This is referred as hashing collision **Example:-**Division method	**(Description of hashing : 1 mark, example:1 mark ,description of collision:2 marks)**

		H(K)=K mod m Consider key value is 55 m is 10 then the hash address for the key 55 is: H(55) = 55 mode 10 H(55)=5 So 5 is the hash address in which key 55 will be stored in hash table. **Collision: -** A situation in when the calculated hash addresses for more than one key, maps to the same location in the hash table, is called a hash collision. Example:- Consider key values are 55 and 65 m is 10 then the hash address for the key 55 is: H(55) = 55 mode 10 H(55)=5 then the hash address for the key 65 is: H(65) = 65 mode 10	
4.		**Attempt any four of the following :**	**16 Marks**
	a)	**Describe classification of data structure with example of each.**	**4M**
	Ans:	Classification of data structure:-  Data structure (ds) is classified into two categories – primitive data structure and non-primitive data structure. **Primitive data structure:** All basic data types of any language are called as primitive data types. It defines how the data will be internally represented in, stored and retrieve from memory. Example: int, char, float **Non-primitive data structure:** All the data structures derived from primitive	**(Correct classification:2 marks , Example of each:2marks)**

		data structures are called as non-primitive data structures. Example: array, list, files. Non-primitive data structure can be classified into two categories: **(1) Linear Data Structure:** In this type of data structure, all the data elements are stored in a particular sequence. Example: stack, queue **(2) Non-Linear data structure:** In this type of data structure, all the data elements do not form any sequence. Example: graph and tree.	
	b)	**Describe following terms with suitable diagram:** **i) Stack overflow** **ii) Stack underflow.**	**4M**
	Ans:	i) Stack overflow ii) Stack underflow **Stack overflow**: When stack contains maximum number of elements i.e. stack is full and push operation is called to insert a new element then stack is said to be in overflow state.  **Stack underflow:** When there is no elements in a stack i.e. stack is empty and pop operation is called then stack is said to be in underflow state  	**(Description of term :1 mark each, diagram -1mark each)**

	c)	**Describe working of dequeue with suitable example.**	**4M**
	Ans:	Dequeue : It is a linear list in which data elements can be inserted or deleted from either end but not in the middle. The data elements can be inserted or deleted from the front or rear end but no changes can be made elsewhere in the list. Dequeue has two types: 1. Input restricted dequeue: It allows insertions at only one end of the list but allows deletions at both the ends of the list. 2. Output restricted dequeuer: It allows deletions at only one end of the list but allows insertions at both ends of the list. Example:- 	**(Description:2 marks, Example:2 marks)**
	d)	**With example, describe how circular linked list works when a node is deleted from beginning of list.**	**4M**
	Ans:	Circular linked list is a collection of nodes where last node is connected to first node. Address field of last node contains address of first node. **Before Deletion**  Deleting a node from the beginning of the list: Node to be deleted is node 1. If node 1is deleted from the beginning then start node (header node) points to node 2. Deleting node from beginning of the list requires to change address field of last node of the list. So after deleting the first node, find last node from the list by traversing through the list. Then set its last field with the address from header node	**(Correct explanation with example:4 marks)**

After Deletion



Step 1: set header pointer with the address of node 2
Step 2: traverse through the list to find last node from the list.
Step 3: set last node's address field with the address from header node.
Step 4: free first node.

e) Write a 'C' program to insert new node at the end of linear linked list. — 4M

Ans: **(**Note: Any relevant logic shall be considered**)** — **(Correct logic: 2 marks, correct syntax: 2 marks)**

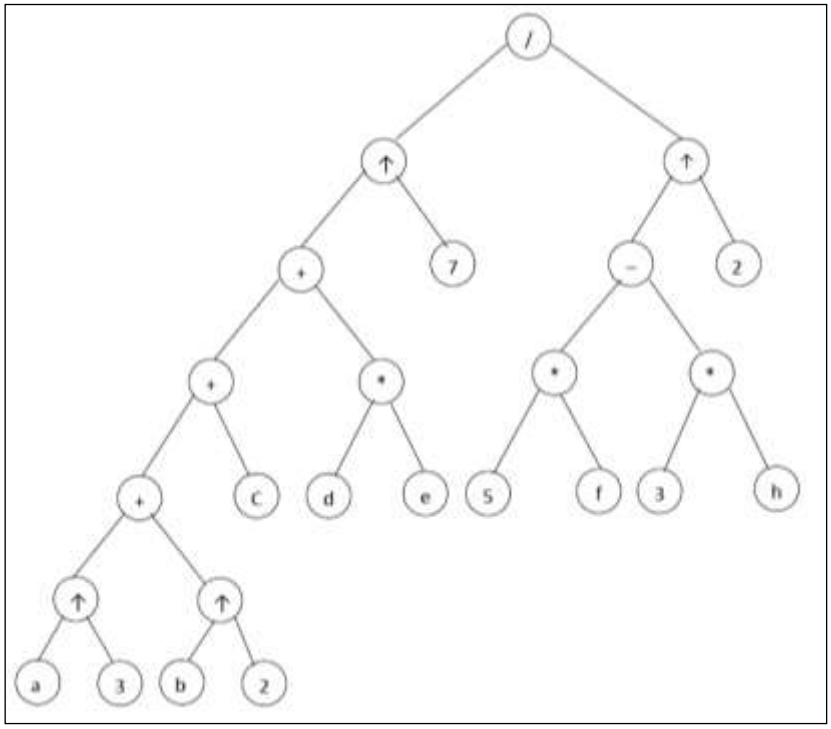
```
#include <stdio.h>
#include <stdlib.h>

struct node {
   int data;         // Data
   struct node *next; // Address
}*head;

void insertAtEnd(int data);

int main()
{
   int n, data;
   printf("\nEnter data to insert at end of the list: ");
   scanf("%d", &data);
   insertAtEnd(data);
   return 0;
}

void insertAtEnd(int data)
{
   struct node *newNode, *temp;
   newNode = (struct node*)malloc(sizeof(struct node));
   if(newNode == NULL)
   {
      printf("Unable to allocate memory.");
   }
   else
   {
      newNode->data = data; // Link the data part
```

```
        newNode->next = NULL;
        temp = head;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = newNode; // Link address part
        printf("DATA INSERTED SUCCESSFULLY\n");
    }
}
```

f) **Draw an expression tree for following expression:**

$(a^3 + b^2+c+de)^7 \ / \ (5f-3h)^2$

Ans:



(Correct Expression Tree: 4 marks)

5. **Attempt any two of the following :** **16 Marks**

a) **Describe working of binary search method. Give stepwise procedure to search 65 in the following list:**
List: 23,12,5,29,10,65,55,70 **8M**

Ans;

Working:

1. Binary search algorithm locates the position of an element in a sorted array.
2. Binary search works by comparing an input value to the middle element of the array.
3. The comparison determines whether the element equals the input, less than the input or greater.
4. When the element being compared to equals the input the search stops and typically returns the position of the element.
5. If the element is not equal to the input then a comparison is made to determine

(Working: 4 marks, Stepwise procedure for searching: 4 marks)

whether the input is less than or greater than the element.

6. Depending on which algorithm then update the beginning or end of array and calculate the middle of array.
7. Algorithm again compare the middle elements with the input value
8. This process continues till either the input value is found or the search condition become false.

Searching Element in Given List:

List: 23, 12, 5, 29, 10, 65, 55, 70

Pre-condition for Binary search is array elements must be in ascending order.

The given list is not sorted.

Sorted List A= {5, 10, 12, 23, 29, 55, 65, 70}

Search element (k) = 65

i. Low=0 high=7

Mid= (0+7)/2 = 3

A[mid] =a [3] =23

65>23

k>a [mid]

ii. Low=mid+1 High=7 Mid= (4+7)/2=5

A[mid] =a [5] =29

65>29

iii. Low=6 high=7

Mid= (6+7)/2 = 6

A[mid] =a [6] =65

A[mid] =k

Therefore key element is found at 6th position, no. of comparison required = 3.

Search is successful

b) **Convert following infix expression into a postfix expression. Show all steps.** **8M**

(P+(Q * R – (S/T↑ U)* V)* W)

Ans:

Sr. No	Symbol Scanned	Stack	Expression
1	(	(	
2	P	(	P
3	+	(+	P
4	(	(+(	P
5	Q	(+(	PQ
6	*	(+(*	PQ

(Correct conversion: 8 marks, step wise marks shall be given)

7	R	(+(*	PQR	
8	-	(+(-	PQR*	
9	(	(+(-(	PQR*	
10	S	(+(-(	PQR*S	
11	/	(+(-(/	PQR*S	
12	T	(+(-(/	PQR*ST	
13	↑	(+(-(/↑	PQR*ST	
14	U	(+(-(/↑	PQR*STU	
15	)	(+(-	PQR*STU↑/	
16	*	(+(-*	PQR*STU↑/	
17	V	(+(-*	PQR*STU↑/**V**	
18	)	(+	PQR*STU↑/V*-	
19	*	(+*	PQR*STU↑/V*-	
20	W	(+*	PQR*STU↑/V*-W	
21	)		PQR*STU↑/V*-W*+	

Postfix Expression for Given Equation is :- PQR*STU↑/V*-W*+

c) Describe Depth First Search traversal of graph with example. 8M

Ans:

Depth First Search (DFS)

The aim of DFS algorithm is to traverse the graph in such a way that it tries to go far from the root node. Stack is used in the implementation of the depth first search. Back tracking used in this algorithm

Algorithm

Step1: Start

Step2: Initialize all nodes as unvisited

Step3: Push the starting node onto the stack. Mark it as waiting.

Step4: Pop the top node from stack and mark is as visited. Push all its adjacent nodes into the stack &mark them as waiting.

Step 5 .Repeat step 4

until stack is empty.

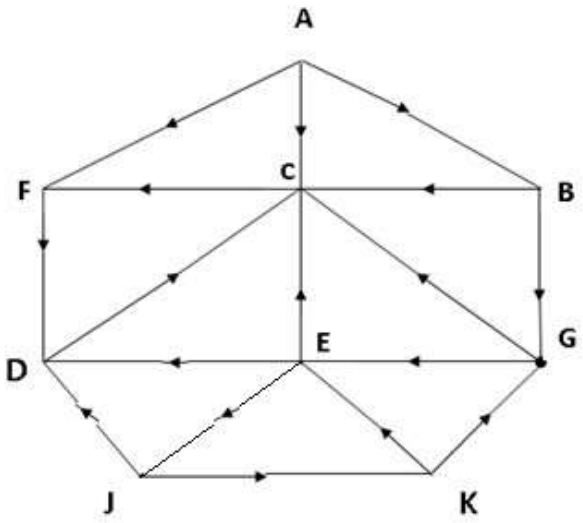
Step 6: Stop

For example, consider the following graph **G** as follows:

Suppose we want to find and print all the nodes reachable from the node **J**

(Description : 4 marks, Example : 4 marks; any relevant example shall be considered)

(Including **J** itself). The steps for the **DFS** will be as follows:



a) Initially, push **J** onto stack as follows: stack: J

b) Pop and print the top element **J** , and then push onto the stack all the neighbors of **J** as follows:
Print J STACK D, K

c) Pop and print the top element K, and then push onto the stack all the unvisited neighbours of k
Print K STACK D, E, G

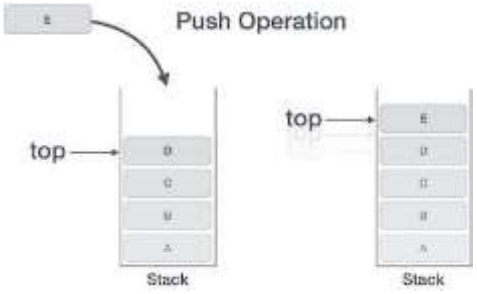
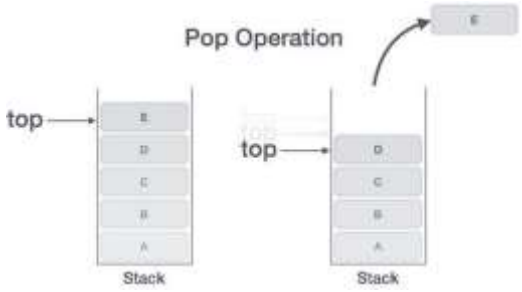
d) Pop and print the top element G, and then push onto the stack all the neighbors of **G. Print G STACK D, E, C**
Note that only **C** is pushed onto the stack, since the other neighbor, **E** is not pushed because **E** has already been pushed onto the stack).

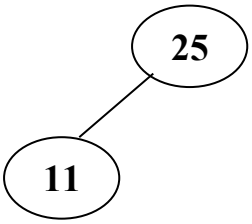
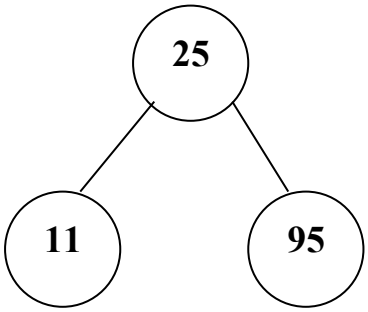
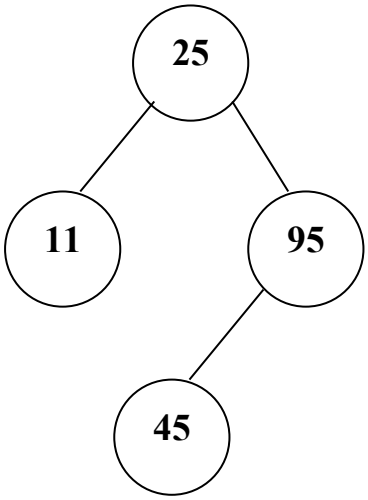
e) Pop and print the top element C and then push onto the stack all the neighbors of **C Print C STACK D, E, F**

f) Pop and print the top element **F Print F STACK D, E**
Note that only neighbor **D** of **F** is not pushed onto the stack, since **D** has already been pushed onto the stack.

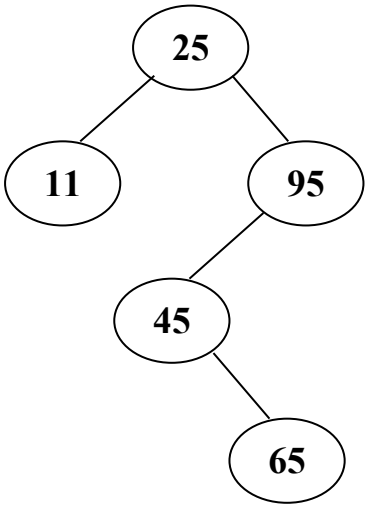
g) Pop and print the top element E and push onto the stack all the neighbors of **D Print E STACK: D,**

h) Pop and print the top element D, and push onto the stack all the neighbors of **D Print D STACK : empty**
The stack is now empty, so the **DFS** of **G** starting at **J** is now complete. Accordingly, the nodes which were printed,
J, K, G, C, F, E, D
These are the nodes reachable from **J**

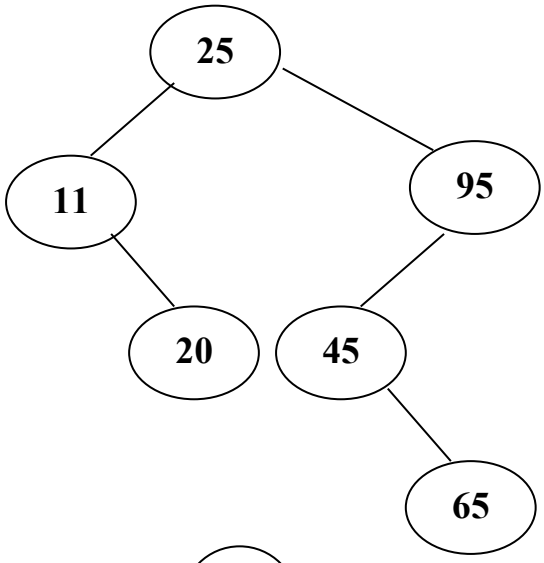
6.		**Attempt any two of the following :**	**16 Marks**
	a)	**Describe push and pop operations on stack. Also write algorithm for push and pop operations.**	**8M**
	Ans:	1) **PUSH:** The process of adding new element to the top of the stack is called PUSH operation. Initially stack top is set to -1 as array started with 0 to indicate empty stack. As the new element is added to the top, for every push operation the top is incremented by one.  2) **POP**: The process of deleting an element from top of the stack is called POP operation. As deletion takes place from the top of the stack. After every POP operation the top of the stack is decremented by one.  **Algorithm for push operation** First check for overflow i.e. if top =max-1 (maximum size of an array), then push operation cannot be performed. If there is no overflow then increment the top by 1 and insert new element at top position. 1. If TOP = =MAX – 1, then: (a) Display "stack overflow" (b) Exit 2. TOP = TOP + 1 3. STACK [TOP] = ITEM **4.** Exit **Algorithm for pop operation** First check for underflow i.e. if top is -1 means there is no element in the stack then pop operation cannot be performed. If there is no underflow then decrement the top by 1. It removes an element placed at top of the stack. 1. If TOP==-1, then 2. Display "The Stack is empty" (a) Exit 3. Else remove the Top most elements	**(Description Push operation and Pop operation: 2 marks each, Push Algorithm: 2 marks, POP Algorithm: 2 marks)**

		4. DATA = STACK [TOP] 5. TOP = TOP – 1 6. Exit.	
	b)	**i) Create a binary search tree using following elements : 25,11,95,45,65,20,22,78,5,10,98,76.** **ii) Write an algorithm to insert a node in a binary search tree.**	
	Ans:	**i) Binary Search Tree:** Insert 25 :-  Insert 11:-  Insert 95:-  Insert 45:- 	**(Tree creation: 4 marks)**

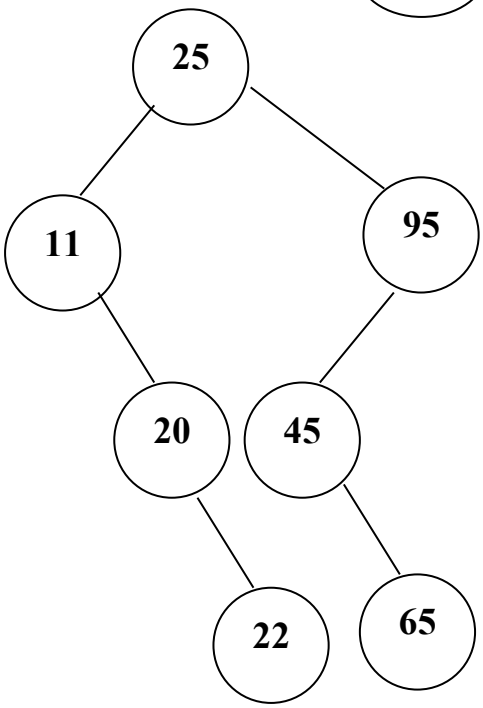
Insert 65:-



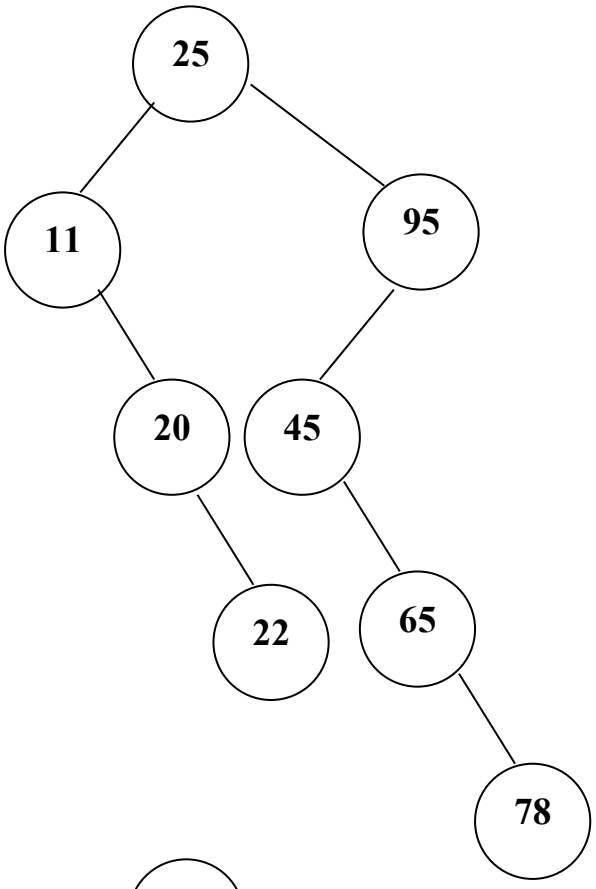
Insert 20:-



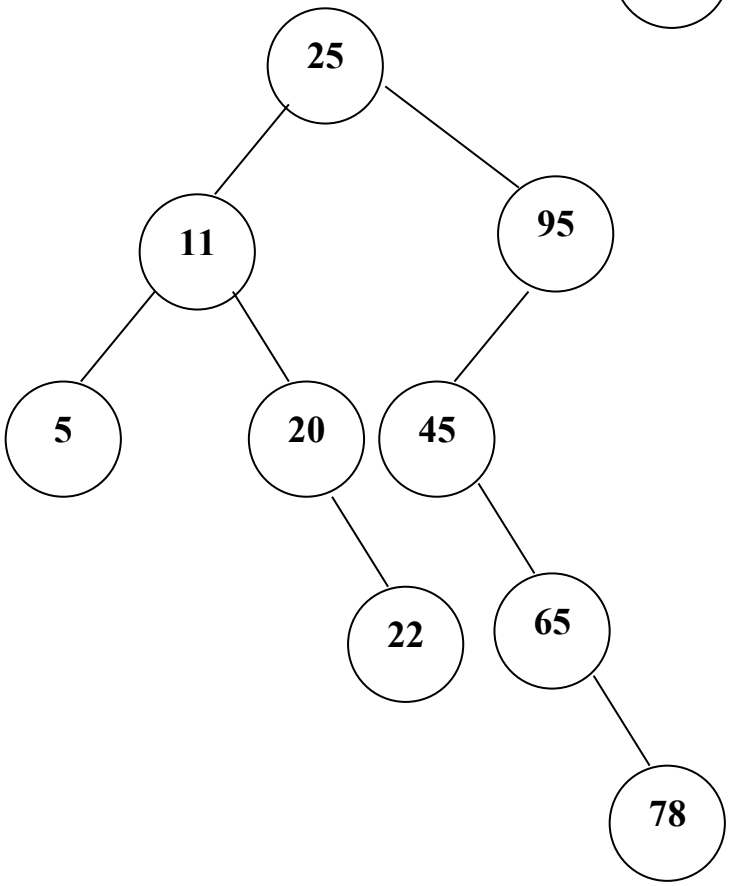
Insert 22:-



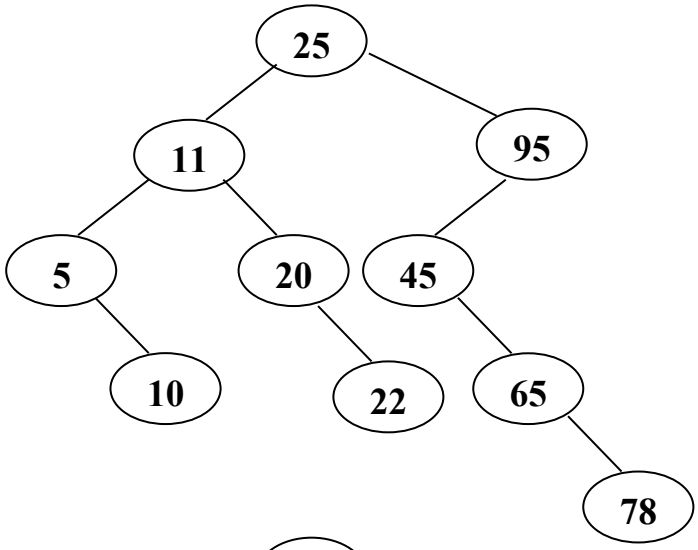
Insert 78:-



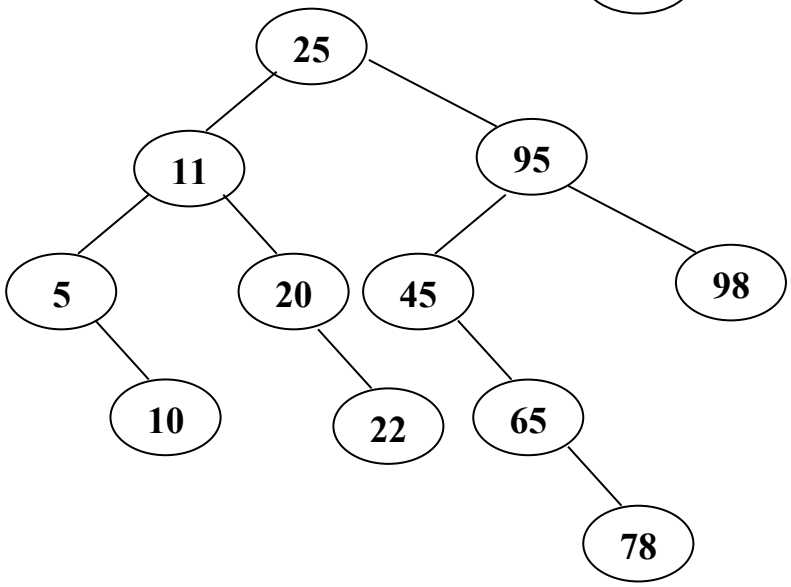
Insert 5 :-



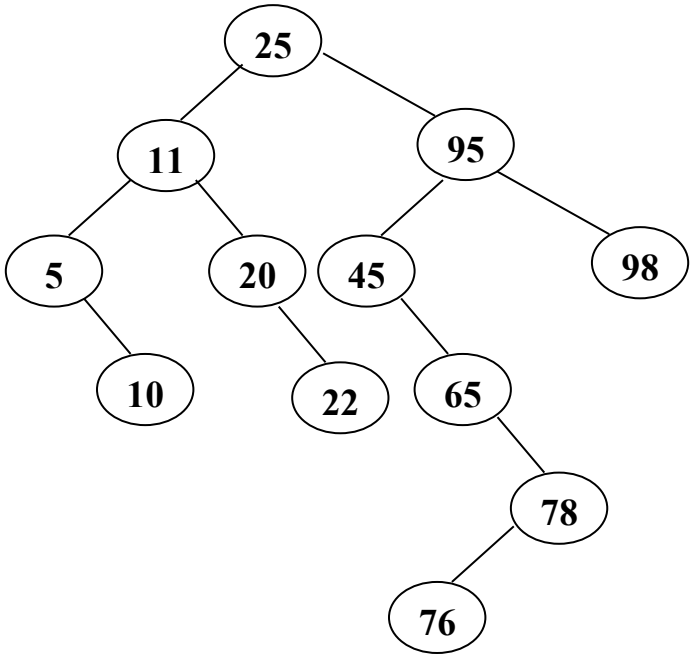
Insert 10:-

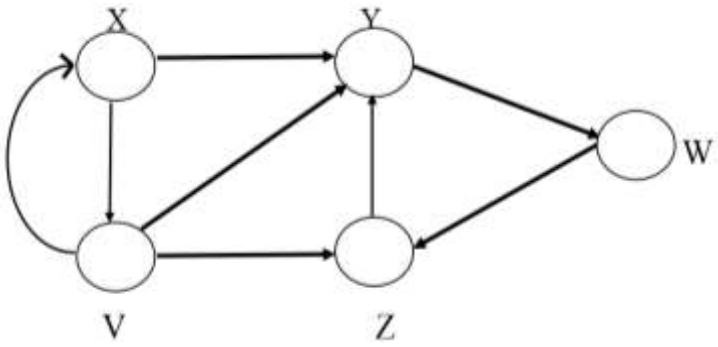


Insert 98:-



Insert 76:-

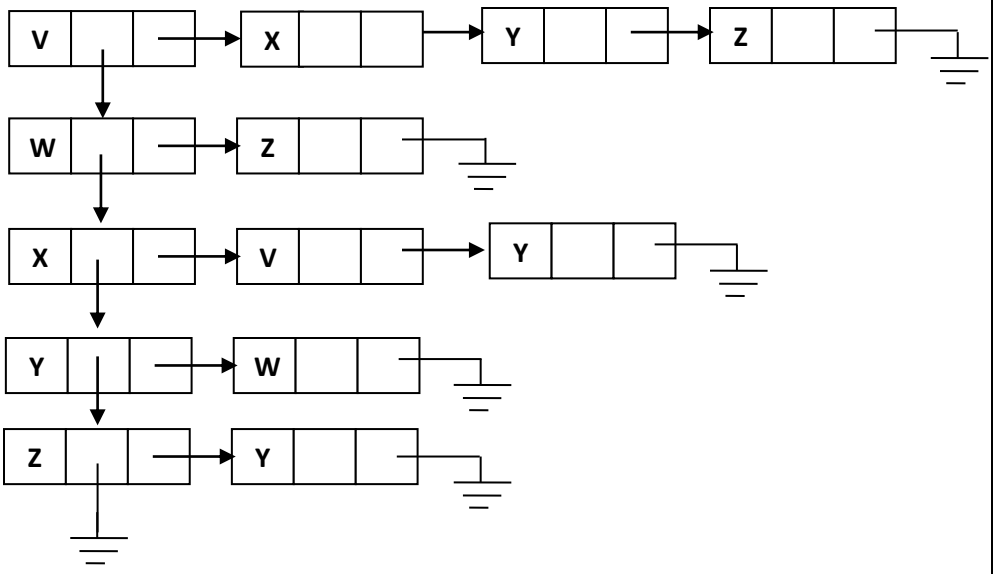


		ii) Algorithm: In a binary search tree, the insertion operation is performed with O(log n) time complexity. In binary search tree, new node is always inserted as a leaf node. The insertion operation is performed as follows... Step 1: Create a newNode with given value and set its left and right to NULL. Step 2: Check whether tree is Empty. Step 3: If the tree is Empty, then set set root to newNode. Step 4: If the tree is Not Empty, then check whether value of newNode is smaller or larger than the node (here it is root node). Step 5: If newNode is smaller than or equal to the node, then move to its left child. If newNode is larger than the node, then move to its right child. Step 6: Repeat the above step until we reach to a leaf node (i.e., reach to NULL). Step 7: After reaching a leaf node, then isert the newNode as left child if newNode is smaller or equal to that leaf else insert it as right child.	**(Algorithm: 4 marks)**
	c)	**Consider the graph 'G' in following figure :**  **i) Find all simple path from X to Z.** **ii) Find indegree and outdegree of nodes Y to Z.** **iii) Find adjacency matrix A for the above graph.** **iv) Give adjacency list representation of above graph.**	
	Ans:	**{{**Note:- Any sequence for adjacency matrix shall be considered**}}** **i) All simple path from X to Z.** a) X → Y → W → Z b) X → V → Y → W → Z c) X → V → Z **ii) Indegree and outdegree of nodes Y to Z.** In-degree(y) = 3 Out-degree(y) = 1 In-degree(z) = 2 Out-degree(z)= 1	**(All Path from X to Z: 2 marks, In-degree & out-degree of Y & Z: 2 marks, Adjacency matrix: 2 marks, Adjacency list: 2 marks)**

iii) Adjacency matrix A for the above graph.

	V	*W*	*X*	*Y*	*Z*
V	***0***	***0***	***1***	***1***	***1***
W	***0***	***0***	***0***	***0***	***1***
X	***1***	***0***	***0***	***1***	***0***
Y	***0***	***1***	***0***	***0***	***0***
Z	***0***	***0***	***0***	***1***	***0***

iv) Adjacency list representation of above graph.



Adjacency List

Node	Adjacency List
V	X, Y, Z
W	Z
X	V , Y
Y	W
Z	Y