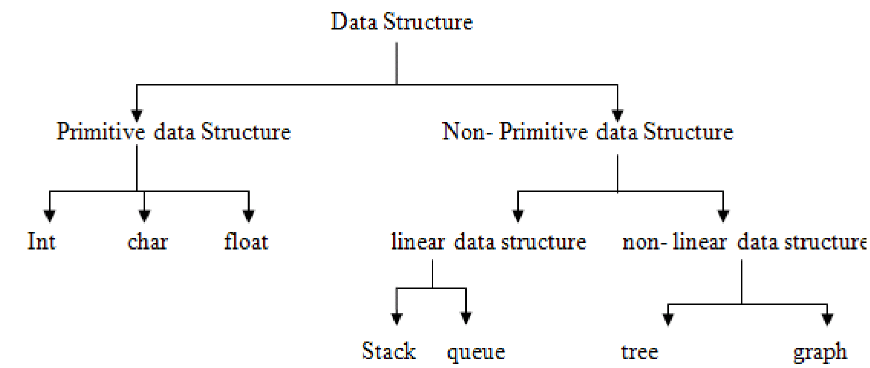
Important Instructions to examiners:

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**
	1)	**Define Big 'O' Notation.**	**2M**
	Ans:	Big O is a mathematical notation that represents time complexity of an algorithm. O stands for order of term.	***(Definition : 2 marks)***
	2)	**Define data structure and give its classification.**	**2M**
	Ans:	A **data structure** is a specialized format for organizing and storing data. i) Data can be organized in many ways and data structures is one of these ways. ii) It is used to represent data in the memory of the computer so that the processing of data can be done in easier way. iii) Data structures is the logical and mathematical model of a particular organization of data	***(Definition: 1 mark, classification: 1 mark)***

Classification:



	3)	**Define searching. Give its type**	**2M**
	Ans:	It is the process of finding a data element in the given data structure. 1. Linear search 2. Binary search	***(Definition:1 mark, types:1 mark)***
	4)	**Define recursion. State any two application where recursion used.** **(**Note: Any other application also to be considered**)**	**2M**
	Ans:	Recursion is the process of calling function by itself. A recursive function body contains function call statement that calls itself repeatedly. **Applications:-** 1. To compute GCD 2. To display Fibonacci series	***(Definition: 1 mark, two applications:1/ 2 mark each)***
	5)	**Define following w.r.t tree** **a) Ancestor** **b) Descendant nodes**	**2M**
	Ans:	**a) Ancestor:** All the nodes that are along the path from root node to a particular node are called as ancestor of that particular node, that is parent nodes are ancestor nodes. **b) Descendant nodes:** All the nodes that are reachable from the root node or parent node are the descendant nodes of that parent node or root node.	***(Definition of each term:1 mark)***

	6)	**Define following w.r.t tree** a) **In-degree** b) **Out - degree**	**2M**
	Ans:	a) **In-degree: - I**n degree of a node is number of edges coming towards the node. b) **Out-degree: -** Out degree of a node is number of edges going out from the node.	***(Definition of each term: 1 mark)***
	7)	**State any four sorting technique.**	**2M**
	Ans:	**1.** Bubble sort **2.** Selection sort **3.** Insertion sort **4.** Radix sort **5.** Shell sort **6.** Quick sort **7.** Merge sort	***(Any four techniques:1/ 2 mark each)***
	8)	**List any four application of graph.**	**2M**
	Ans:	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 four applications:1 /2 mark each)***
	B)	**Attempt any two of the following:**	**8M**
	1)	**What is complexity of an algorithm? Describe time complexity and space complexity.** **[Example optional]**	**4M**
	Ans:	**Complexity of an algorithm:** The complexity of an algorithm is a measure that describes its efficiency in terms of amount of time and space required for an algorithm to process.	***(Definition of complexity:1mark, description of time***

		Time complexity:- Time complexity of an algorithm is the amount of computer time required to execute an algorithm. **Example:** Consider three algorithms given below:- Algorithm A: - a=a+1 Algorithm B: - for x = 1 to n step 1 a=a+1 Loop Algorithm C: - for x=1 to n step 1 for y=1 to n step 1 a=a+1 Loop Frequency count for algorithm A is 1 as a=a+1 statement will execute only once. Frequency count for algorithm B is n as a=a+1 is key statement executes n time as the loop runs n times. Frequency count for algorithm C is n as a=a+1 is key statement executes n2 time as the inner loop runs n times, each time the outer loop runs and the outer loop also runs for n times. **Space complexity:-** Space complexity of an algorithm is the amount of memory required for an algorithm. The space needed by the program is the sum of the following components:- • **Fixed space requirements: -** It includes space for instructions, for simple variables, fixed size structured variables and constants. • **Variable space requirements: -** It consists of space needed by structured variables whose size depends on particular instance of variables. **Example: -** additional space required when function uses recursion.	***complexity:1$^{1/2}$ marks, space complexity:1$^{1/2}$ mark)***
	2)	**Describe binary search algorithm. Give example to search an element using binary search algorithm.**	**4M**
	Ans:	**Binary search algorithm:** Binary search requires sorted list to perform searching. First find the lower index and upper index of an array and calculate mid with the formula (lower+upper)/2.Compare the search element with mid position element. If both are equal then stop the search process. If both are not equal then divide list into two parts. If the search element is less than mid position element then change the upper index value and use first half of the list for further searching process. If the search element is greater than mid position element than change the lower index value and use second half of the list for further searching process. Again find lower and upper index value and calculate mid value. Repeat the process till element is found or lower index value is less than or equal to upper index value.	***(Description:2 marks, Example: 2 marks)***

Example:

Input string:- 10,20,30,40,50,60,70,80,90,100

Search element: - 80 (S)

Array X [10]: used to store elements, lower is lower index of array, upper is upper index of array.

Step 1:-

X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	X[6]	X[7]	X[8]	X[9]
10	20	30	40	50	60	70	80	90	100

Lower=0, upper=9 ; mid=9/2=4.5=4

S! =X [4] i.e. 80! =50

80>50 so lower=lower+1=5

Step 2:

X[5]	X[6]	X[7]	X[8]	X[9]
F	G	H	I	J

lower=5, upper=9; mid=5+9/2=7

S=X [7] i.e 80=80

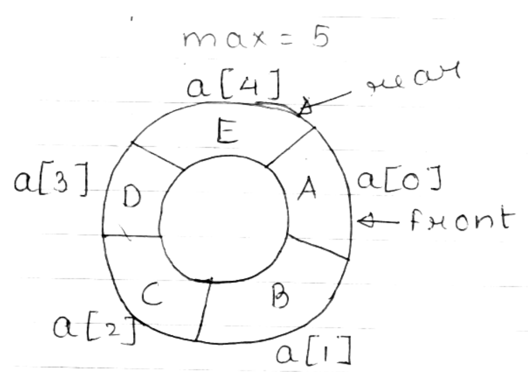
Search successful.

3)	**Describe circular queue. Give its advantages.**	**4M**

Ans:

A circular queue is a linear data structure where it store all elements in a specific order. It has two ends front and rear where front is used to delete an element and rear is used to insert an element. The last location of circular queue is connected to first location of the same. It follows circular path while performing insertion and deletion.

(Description: 3 marks, any one advantage:1 mark)



		Advantage:- • It allows insertion of an element in a queue when queue has empty space in it. Before insertion, it checks for circular queue full. If queue is not full then it performs insert operation to add man element in it.	
2.		**Attempt any four of the following :**	**16**
	a)	**Describe working of inserting sort. Demonstrate working of insertion sort algorithm to sort 6 elements.**	**4M**
	Ans:	In insertion sort, sorting is done on the basis of shift and insert principle. In first pass, 1st index element is compared with o^{th} index element. If o^{th} index element is greater than 1^{st} index element then store 1^{st} index element into a temporary variable and shift o^{th} index element to its right by one position. Then insert temporary variable value in o^{th} position. In pass 2 compare 2^{nd} index element with o^{th} index and then 1^{st} index elements. If required perform shift and insert operations. In each pass fix one position and compare it with all elements placed before it. Continue this process till last position element is compared with all elements placed before it. **Example- Input list: 25, 15,5,24,1,30**	**(Description:2 marks, example:2 marks)**

Pass 1 :

I_1 25 15 5 24 1 30

shift

15 25 5 24 1 30

Pass 2 :

I_1 15 25 5 24 1 30

shift

I_2 5 15 25 24 1 30

Pass 3 :

I_1 5 15 25 24 1 30

I_2 5 15 25 24 1 30

I_3 5 15 25 24 1 30

shift

5 15 24 25 1 30

Pass 4 :

I_1 5 15 24 25 1 30

shift

I_2 1 5 15 24 25 30

I_3 1 5 15 24 25 30

I_4 1 5 15 24 25 30

Pass 5

I_1 1 5 15 24 25 30

At the end of pass 4 the list is in sorted order.

b)	**Find out prefix equivalent of the following expression:** **i) [(A + B)] + C] * D ii) A [(B*C)+D]**	**4M**

Ans:

1) [(A+B)+C]*D

Infix string	Read char	Stack	Prefix string
[(A+B)+C]*D	D		D
	*	*	D
	]	] *	D
	C	] *	CD
	+	+] *	CD
	)	) +] *	CD
	B	) +] *	BCD
	+	+) +] *	BCD
	A	+) +] *	ABCD
	(	+] *	+ABCD
	[	*	++ABCD
			*++ABCD

Prefix conversion:- *++ABCD

ii) A[(B*C)+D]

Infix string	Read char	stack	Prefix string
A[(B*C)+D]	]	]	
	D	]	D
	+	+]	D
	)	) +]	D
	C	) +]	CD
	*	*) +]	CD
	B	*) +]	BCD
	(	+]	*BCD
	[		+*BCD
	A		A+*BCD

Prefix conversion:-A+*BCD

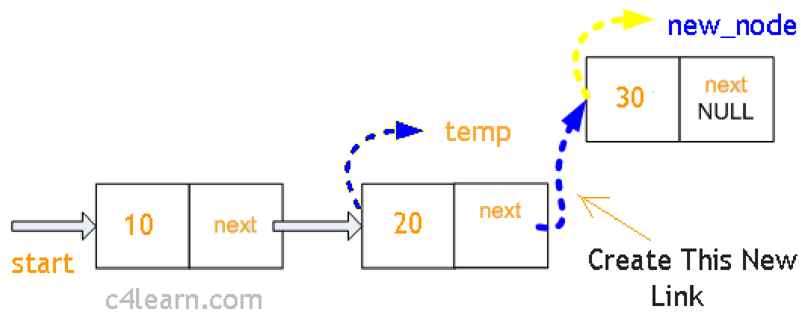
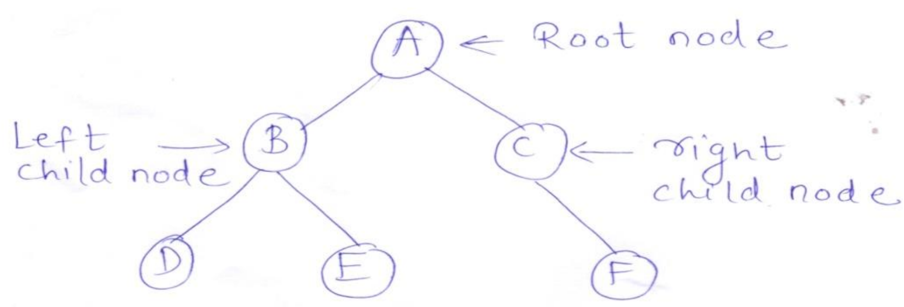
	(c)	**Write an algorithm to insert a new node as the last of a singly linked list. Give example.**	**4M**

Ans:

Inserting node at last in the SLL (Steps):

1. Create New Node
2. Fill Data into "Data Field"
3. Make it's "Pointer" or "Next Field" as NULL
4. Node is to be inserted at Last Position so we need to traverse SLL up to Last Node.
5. Make link between last node and new node
   temp -> link = new_node;

(Algorithm:2 marks, example:2 marks)

		Example:-  	
	(d)	**Describe concept of binary tree. State its application.**	**4M**
	Ans:	A binary tree is a tree where each node from the tree has maximum two child nodes. Each node can have one child, two child nodes or no child node in a binary tree. A child node on a left side of parent node is called as left child node. A child node on a right side of parent node is called as right child node.  **Applications:** 1. To create expression tree. 2. To create Binary search tree. 3. To represent hierarchical data into the memory	***(Description:3 marks, any one application:1mark)***
	e)	**Write a program to insert element in queue.**	**4M**

	Ans:	Implementation insertion on Queue Using array:
		`#include<stdio.h>`
		`#include<conio.h>`
		`#define max 3`
		`int rear=-1;`
		`int front=-1;`
		`int queue[max];`
		`void insert();`
		`void insert()`
		`{`
		`int insert_item;`
		`if(rear==(max-1))`
		`printf("\n queue is full");`
		`else`
		`{`
		`printf("\n enter element to be inserted:");`
		`scanf("%d",&insert_item);`
		`rear=rear+1;`
		`queue[rear]=insert_item;`
		`if(front==-1)`
		`{`
		`front=0;`
		`}`
		`}`
		`}`
		
		`void main()`
		`{`
		`insert();`
		`}`
			(Correct logic :2 marks, correct syntax:2 marks)

	(f)	**Write a program to search an element in an array. Display position of element. [Linear search or binary search program shall be considered.]**	**4M**

	Ans:	**Linear search:-**
		
		`#include<stdio.h>`
		`#include<conio.h>`
		`void main()`
		`{`
		`int a[10]={10,20,30,40,50,60,70,80,90,100};`
		`int i,num;`
		`printf("LINEAR SEARCH");`
		`printf("\n INPUT LIST:-\n");`
		`for(i=0;i<=9;i++)`
			(Correct logic: 2 marks, correct syntax:2 marks)

```
printf("%d\t",a[i]);
printf("\nEnter search element:");
scanf("%d",&num);
for(i=0;i<=9;i++)
{
if(a[i]==num)
{
printf("\n Element found at %d index position ",i);
break;
}
}
if(i==5)
printf("\n Element not found");
getch();
}
```

OR

Binary search:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10]={10,20,30,40,50,60,70,80,90,100};
int i,mid,num,upper=9,lower=0, flag=0;
clrscr();
printf("BINARY SEARCH");
printf("\n INPUT LIST:-\n");
for(i=0;i<=9;i++)
printf("%d\t",a[i]);
printf("\nEnter search element:");
scanf("%d",&num);
while(lower<=upper)
{
mid=(upper+lower)/2;
if(a[mid]==num)
{
flag=1;
printf("\n Element found at %d index position ",mid);
break;
}
if(a[mid]>num)
upper=mid-1;
else
lower=mid+1;
```

		}
		 if(flag==0)
		 printf("\n Element not found");
		 getch();
		 }	

3.		**Attempt any four of the following**	**16**
	a)	**Describe PUSH and POP operation on stack using array representation.**	**4M**

Ans: Stack is a linear data structure which follows Last-In First - Out (LIFO) principle where, elements are inserted (push) and deleted (pop) from only one end called as stack top.

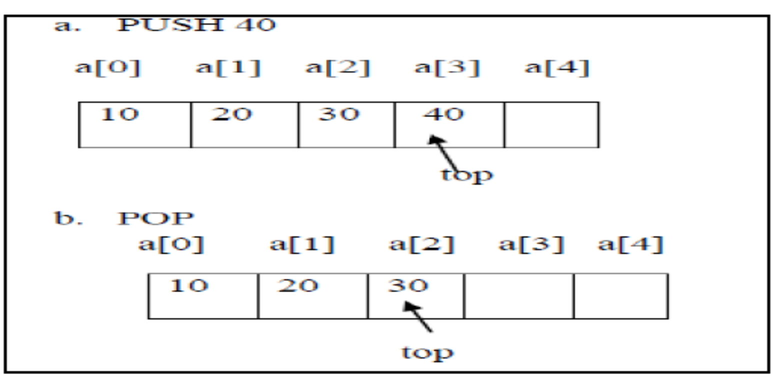
(PUSH operation:2 marks & POP operation: 2 marks)

Push Algorithm:

Step 1: [Check for stack full/ overflow]
If stack top is equal to max-1 then write "Stack Overflow"
return
Step 2: [Increment top]
top= top +1;
Step 3 : [Insert element]
stack [top] = item;
Step 4 : return

Pop Algorithm:

Algorithm:
Step 1: [Check for stack empty/ underflow]
If stack top is equal to -1 then write "Stack Underflow"
return
Step 2: [Copy data]
item=stack[top];
Step 3 : [decrement top]
top = top-1;
Step 4 : return



b) **What is priority queue? Describe working of priority queue with suitable example.** **4M**

Ans:

A priority queue is a queue in which the intrinsic ordering among the elements decides the result of its basic operations i.e. the ordering among the elements decides the manner in which Add and Delete operations will be performed. In a priority queue,

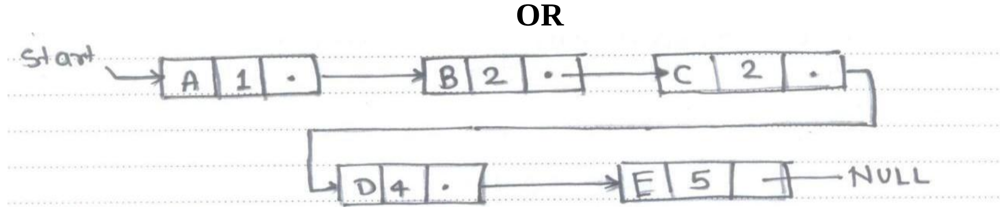
1. Each element is assigning a priority.
2. The elements are processed according to, higher priority element is processed before lower priority element and two elements of same priority are processed according to the order of insertion.

(Priority queue: 1 mark, Working: 2 marks, Example :1 mark)

(Represent either with array or linked list)
Array representation: Array element of priority queue has a structure with data, priority and order. Priority queue with 5 elements:

C,1,4	B,3,2	B,3,5	A,4,1	D,5,3

OR



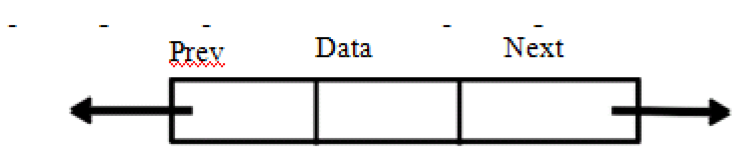
Above figure shows priority. Queue with 5 elements where B & C have same priority number. Each node in above priority queue contains three items.

c) **Describe working of doubly linked list. Write syntax used for double linked list in program** **4M**

Ans:

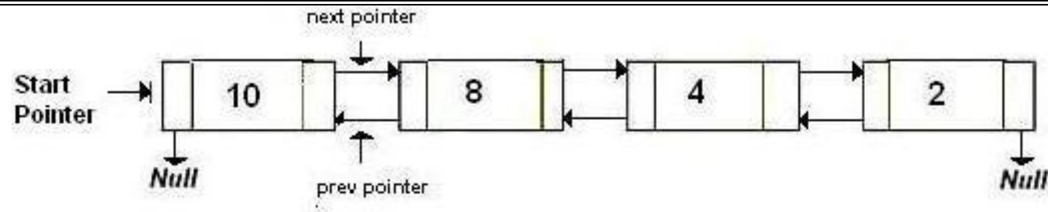
A doubly linked list is a linked list in which each node contains two links- one pointing to the previous node and pointing to the next node.

(Working: 2 marks, Example: 1 mark, Syntax:1 mark)



Each node contains three parts.
1. Data: contains information. E.g.10, 20, etc.
2. Next pointer: Contains address of the next node.
3. Prev pointer: Contains address of the preceding node.

Example:



The syntax for doubly linked list is-

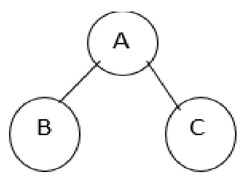
```
Struct node
 {
int data;
Struct node *next, * prev;
}
```

	d)	**Write algorithm for morder traversal for binary tree. Demonstrate with suitable example.** **(**Note: Any Binary Tree Traversal shall be considered**)**	**4M**
	Ans:	**Algorithm for Inorder Traversal:**	***(Algorithm: 2 marks, Example: 2 marks)***

Algorithm for Inorder Traversal:

Step 1: Visit left subtree in inorder.
Step 2: Visit root node.
Step 3: Visit right subtree in inorder

Example:



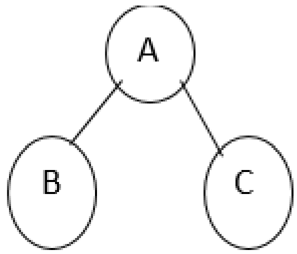
Inorder traversal is: B, A, C.
In this traversal method 1st process left subtree, then root element & then right subtree.

OR

Algorithm for Preorder Traversal:

Step 1: Visit root node.
Step 2: Visit left subtree in preorder.
Step 3: Visit right subtree in preorder.

Example:



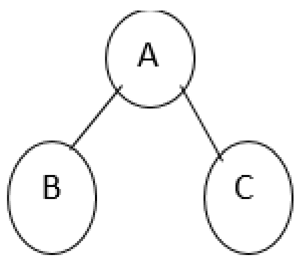
Preorder traversal is: A, B, C.
In this traversal method 1st process root element then left subtree & then right subtree.

OR

Algorithm for Postorder Traversal:

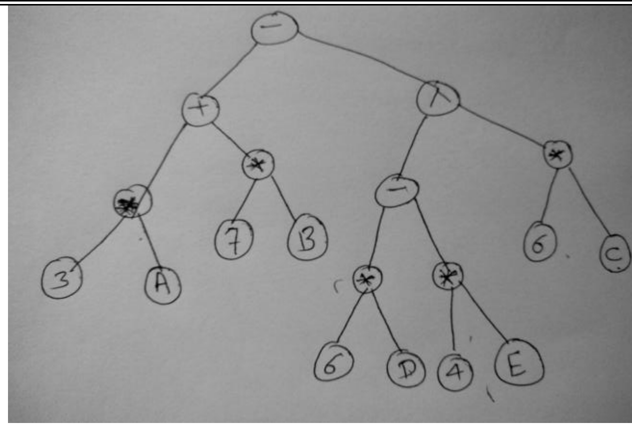
Step 1: Visit left subtree in postorder.
Step 2: Visit right subtree in postorder.
Step 3: Visit root node

Example:



Preorder traversal is: B, C, A.
In this traversal method 1st process left subtree then right subtree & then root element.

	e)	**Draw tree structure for following expression.** **[3A + 7B] – [(6D – 4E) ^ 6C]**	**4M**
	Ans:		***(Correct answer :4 marks)***



	f)	**What is collision resolution techniques? State its types.**	**4M**

Ans:

When the hash function generates the same integer on different keys, it result into collision. Two records cannot be stored in the same location.
A method used to solve the problem of collision is called the collision resolution techniques.

Types:

1. **Open addressing**
   i) Linear probing
   ii) Quadratic probing
   iii) Rehashing
   iv) Chaining

(Defining Collision resolution techniques:2 marks, Listing Types:2 marks)

4.		**Attempt any four of the following :**	**16**
	a)	**Compare Top-down approach v/s Bottom –up approach[any four points].**	**4M**

Ans:

Top-down approach	Bottom-up approach
A top-down approach starts with identifying major components of system or program decomposing them into their lower level components & iterating until desired level of module complexity is achieved .	A bottom-up approach starts with designing most basic or primitive Component & proceeds to higher level components.
In this we start with topmost module & Incrementally add modules that is calls.	Starting from very bottom, operations That provide layer of abstraction are implemented.
Top down approach proceeds from the abstract entity to get to the concrete design	Bottom up approach proceeds from the concrete design to get to the abstract entity.

Top down design is most often used in designing brand new systems	Bottom up design is sometimes used when ones reverse engineering a design; i.e. when one is trying to figure out what somebody else designed in an existing system.
Top-down approach is simple and not data intensive.	Bottom-up approach is complex as well as very data intensive
Top-down approaches are backward-looking.	Bottom-up approaches are forward-looking.
Example is c programming.	Example is C++ programming.

b) **How stack is used in Recursion? Describe with suitable example.** **4M**

Ans:

(Explanation of how stack used in recursion: 2 marks, Example:2 marks)

1. Recursion is calling a function from itself repeatedly. A function call to the recursive function is written inside the body of a function.
2. In the recursive call each time a function executes the same number of statements repeatedly. Each function contain local variables.
3. When a recursive function is called, before executing the same function again, the local variables are saved in the data structure stack. This way in each execution local variables values are copied to the stack.
4. When the recursive function terminates one by one each element is removed from the stack and we get the result.

Example: Factorial of number, Tower of Hanoi. Fibonacci Series

```
int factorial (int no)
{
If(no==1)
 Return 1;
Else
Fact=fact*factorial(no-1);
}
```

Fact= fact* factorial (no-1); This statement gives recursive call to the function.

Each time when factorial function is called stack stores the value of current local variables and then next variable. If factorial of 5 then first time stack will have 5 then in 2nd call 4 … till the last call stack gets the element. Once it is empty all variable values are pop & result is calculated.

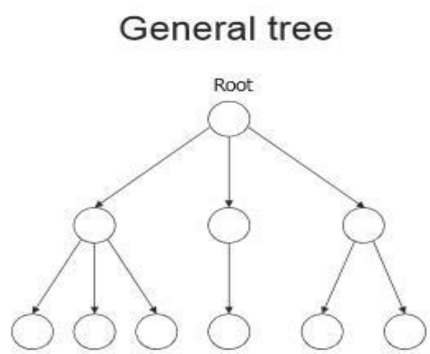
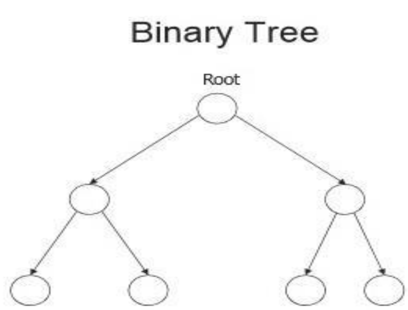
	c)	**Write a code delete an element in queue.**	**4M**

	Ans:	Delete queue(&a, &front, &rear)
		
		Step 1: [check underflow]
		if front=-1 then write "queue empty"
		return
		otherwise go to Step 2
		Step 2: [copy data]
		data=a[front]
		Step 3: [check front & rear pointer]
		if front=rear then front =rear=-1
		otherwise
		front=front+1
		Step 4: End/ return to calling function
		OR
		void deletion()
		{
		if(front==-1 ||front==rear+1)
		{
		printf("Queue is empty\n");
		return;
		}
		item=q[front];
		front=front+1;
		printf("Element deleted is::%d",item);
		}
			(Correct code: 4 marks)

	d)	**Define following terms:** **i) Node** **ii) Null pointer** **iii) Empty list** **iv) Information**	**4M**
	Ans:	i) **Node:** It is a data element which contains info as well as address of next node. ii) **NULL pointer:** It is used to specify end of list. The last element of list contains NULL pointer to specify end of list. iii) **Empty list:** A linked list is said to be empty if head (start) node contains NULL pointer. iv) **Information:** It is also known as data part. It is used to store data inside the node.	*(Each term:1 mark Each)*

	e)	**Write an algorithm to traverse a singly linked list.** *****NOTE: Description with example can also be considered.***	**4M**

	Ans:	**Algorithm to traverse a singly linked list**
		1. if (start==NULL) then display "linked list is empty".
		2. Otherwise Visit each node of linked list and display its data till end of the list q=start // Assign a temporary pointer q to starting node while(q!=NULL) do
		Display q->data // display node information
		q=q->link;
			(Correct stepwise algorithm: 4 marks)

	f)	**Describe general tree and binary tree.**	**4M**
	Ans:	**General Tree:**  1. A general tree is a data structure in that each node can have infinite number of children 2. In general tree, root has in-degree 0 and maximum out-degree n. 3. In general tree, each node have in-degree one and maximum out-degree n. 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} 5. Subtree of general tree are not ordered. **Binary tree:**  	***(General tree:2 marks, Binary tree:2 marks)***

1. A Binary tree is a data structure in that each node has at most two nodes left and right
2. In binary tree, root has in-degree 0 and maximum out-degree 2.
3. In binary tree, each node have in-degree one and maximum out-degree 2.
4. Height of a binary tree is : Height(T) = { max (Height(Left Child) , Height(Right Child) + 1}
5. Subtree of binary tree is ordered.

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

a) **Sort following elements by radix sort algorithm** **8M**

87.3, 2.34, 7.29, 3.59, 45.8, 3.79, 3.20, 422.

{Note: As radix sort cannot be applied on decimal number, consider all number without decimal i.e. 873,234,729,359,458,379,320,422)**}**

Ans: **Sorting of Given Numbers: -**

(Correct sorting on given numbers;8 marks)

Pass 1:

Element	0	1	2	3	4	5	6	7	8	9
873				873						
234					234					
729										729
359										359
458									458	
379										379
320	320									
422			422							

Output of Pass 1: 320,422,873,234,458,729,359,379

Pass 2:

Element	0	1	2	3	4	5	6	7	8	9
320			320							
422			422							
873								873		
234				234						
458						458				
729			729							
359						359				
379								379		

Output of Pass 2: 320,422,729,234,458,359,873,379

Pass 3:

Element	0	1	2	3	4	5	6	7	8	9
320				320						
422					422					
729								729		
234			234							
458					458					
359				359						
873									873	
379				379						

Output of Pass 3: 234,320,359,379,422,458,729,873

OR

87 2 7 3 45 3 3 422

087 002 007 003 045 003 003 422

Pass 1:

Elements	0	1	2	3	4	5	6	7	8	9
087								087		
002			002							
007								007		
003				003						
045						045				
003				003						
003				003						
422			422							

Output of Pass 1: 002, 422, 003, 003, 003, 045, 087, 007

Pass 2:

Elements	0	1	2	3	4	5	6	7	8	9
002	002									
422			422							
003	003									
003	003									
003	003									
045					045					
087									087	
007	007									

Output of Pass 2: 002, 003, 003, 003, 007, 422, 045, 087

Pass 3:

Elements	0	1	2	3	4	5	6	7	8	9
002	002									
003	003									
003	003									
003	003									
007	007									
422					422					
045	045									
087	087									

Output of Pass 3: 002,003,003,003,007,045,087,422

b) **Convert the given infix expression to postfix expression using stack and the details of stack at each step of conversation.**
EXPRESSION P* Q ↑ R – S/T +[U/V] **8M**

Ans:

(Correct Postfix Expression: 8 marks)

SYMBOL SCANNED	STACK	RESULTANT
	[	
P	[	P
*	[*	P
Q	[*	PQ
↑	[*↑	PQ
R	[*↑	PQR
-	[-	PQR↑*
S	[-	PQR↑*S
/	[-/	PQR↑*S
T	[-/	PQR↑*ST
+	[+	PQR↑*ST/-
[	[+[	PQR↑*ST/-
U	[+[	PQR↑*ST/-U
/	[+[/	PQR↑*ST/-U
V	[+[/	PQR↑*ST/-UV
]	[+	PQR↑*ST/-UV/
]	NIL	PQR↑*ST/-UV/+

THE POSTFIX EXPRESSION IS:**PQR↑*ST/-UV/+**

c) **Describe DFS with suitable examples.** **8M**

Ans:

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

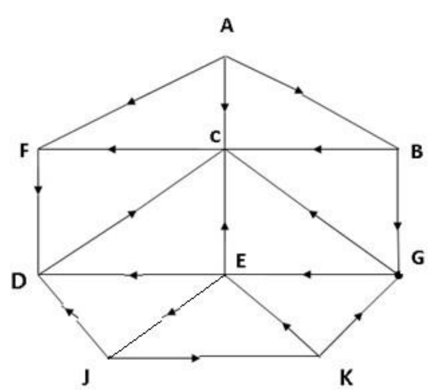
Step3: Push the starting node onto the stack. Mark it as waiting.

Step4: Pop the top node from stack and mark is as visited. Push all its adjacent no des 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** (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 neighbors 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**
e) 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).
f) Pop and print the top element C and then push onto the stack all the neighbors of **C Print C STACK D, E,F**
g) Pop and print the top element **F Print F STACK D,E**

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

h) Note that only neighbor **D** of **F** is not pushed onto the stack, since **D** has already been pushed onto the stack.

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

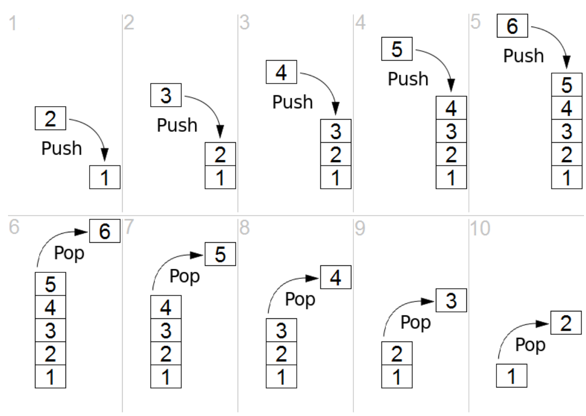
j) The stack is now empty, so the **DFS** of **G** starting at **J** is now complete. Accordingly, the nodes which were printed K, G, C, F, E, D These are the nodes reachable from **J.**

6.		**Attempt any two of the following:**	**16**
	a)	**How stack is useful in reversing a list? write a C program to reverse a list using stack**	**8M**
	Ans:		***(Explanation: 4 marks, program: 4 marks (Any other relevant logic can be considered.)***

Stack is useful to reverse a list. It can be simply done by pushing the individual elements of list one by one on the stack, till end of list is reached and there is no more elements to push on stack. The elements are then popped one by one till the stack is empty.

Eg. Consider the list to reverse as 1, 2, 3, 4, 5, 6.

This can be done using stack as:



Reverse of list is: 6,5,4,3,2,1

Program to reverse a list using stack.

```
#include <stdio.h>
#define MAXSIZE 7
#define TRUE 1
#define FALSE 0
struct Stack
{
   int top;
   int array[MAXSIZE];
} st;
void initialize()
{
```

```
 st.top = -1;
}
int isFull()
 {
    if(st.top >= MAXSIZE-1)
        return TRUE;
    else
        return FALSE;
}
 int isEmpty()
{
 if(st.top == -1)
     return TRUE;
 else
     return FALSE;
}
void push(int num)
{
    if (isFull())
        printf("Stack is Full...\n");
    else
{
        st.array[st.top + 1] = num;
        st.top++;
    }
}
 int pop()
{
    if (isEmpty())
        printf("Stack is Empty...\n");
    else
{
     st.top = st.top - 1;
        return st.array[st.top+1];
    }
}

void printStack()
{
 if(!isEmpty())
{
     int temp = pop();
     printStack();
     printf(" %d ", temp);
     push( temp);
    }
}
```

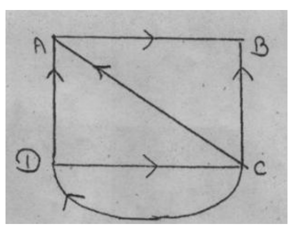
```
void insertAtBottom(int item)
 {
   if (isEmpty())
{
       push(item);
   }
else
 {

       int top = pop();
       insertAtBottom(item);
            push(top);
   }
}

void reverse()
{
  if (!isEmpty())
 {
       int top = pop();
       reverse();
       insertAtBottom(top);
   }
}
int getSize()
{
 return st.top+1;
}

int main()
{
   initialize(st);
   push(1);
   push(2);
   push(3);
   push(4);
   push(5);
   printf("Original Stack\n");
   printStack();
   reverse();
   printf("\nReversed Stack\n");
   printStack();
   getch();
   return 0;
}
```

		Output Original Stack 1 2 3 4 5 Reversed Stack 5 4 3 2 1	
	b)	**Write a program to calculate number node in binary search tree.**	**8M**
	Ans:	See program below	***(Correct program: 8 marks) (Any other relevant logic can be considered.)***

```
#include<stdio.h>
struct tree
{
  struct tree *lchild ;
  int data ;
  struct tree *rchild ;
};
struct tree *root ,*new, *curr ,*prev ;
int ch,n ;
char c ;

main()
{
  do
   {
     clrscr();
     printf("\n---------------------------");
     printf("\n        M E N U     ");
     printf("\n---------------------------");
     printf("\n  1 . C R E A T E ");
     printf("\n  2 . COUNT ");
     printf("\n  3 . E X I T ");
     printf("\n---------------------------");
     printf("\n Enter Your Choice => ");
     scanf("%d",&ch);

     switch( ch )
     {
       case 1 : create();
        break ;
```

```
        case 2 : printf("\nThe number of nodes in tree are :%d\n",count(root));
         getch();
         break ;
 case 3 : exit(0);
    }
  } while(ch != 3 );
return(0);
}

/* FUNCTION TO CREATE A TREE */
int create()
{
  printf("\nEnter the Root Element ");
  scanf("%d",&n);
  root = ( struct tree * ) malloc(sizeof(struct tree ));
  root -> lchild = NULL ;
  root -> data  = n ;
  root -> rchild = NULL ;
  printf("\nDo you want to continue ? ");
  c = getch();
  while ( c == 'y' || c =='Y' )
  {
    printf("\nEnter the next Element ");
    scanf("%d",&n);
    new = ( struct tree * ) malloc(sizeof(struct tree ));
    new -> lchild = NULL ;
    new -> data  = n ;
    new -> rchild = NULL ;
    curr = root ;
    while ( curr != NULL )
    {
      prev = curr ;
      if ( curr -> data > new -> data )
 curr = curr -> lchild ;
      else
 curr = curr -> rchild ;
    }
    if( prev -> data > new -> data )
      prev -> lchild = new ;
    else
      prev -> rchild = new ;

    printf("\nDo you want to continue ? ");
    c = getch();
  }
}
```

		int count(struct tree *p)
		{
		if(p == NULL)
		return(0);
		else if(p->lchild == NULL && p->rchild == NULL)
		return(1);
		else return(1 + (count(p->lchild) + count(p->rchild)));
		}	

	c)	**Consider the graph 'G' in fig.** **i) Find all simple paths from C- A.** **ii) Find all simple paths from D-B.** **iii) Find indeg [B] and outdeg[C].** **iv) Find the adjacency matrix A for graph.** **v) Give the adjacency list representation of graph.** 	**8M**
	Ans:	**i) Find all simple paths from C- A.** **(a) C -> A** **(b) C-> D-> A** **ii) Find all simple paths from D-B.** **(a) D-> C-> B** **(b) D-> C-> A-> B** **(c) D-> A-> B** **iii) Find indeg [B] and outdeg[C].** **(a) indeg [B] : 2** **(b) outdeg [C] : 3**	***(i:1 mark,ii:1 mark,iii:2 marks,iv:2 marks,v:2 marks)***

vi) Find the adjacency matrix A for graph.

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

iv) Give the adjacency list representation of graph.