SUMMER– 18 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.	Answers	Marking Scheme
1.	(A)	**Attempt any five :**	**20 Marks**
	a)	**State the need of data structure. Write any four operations perform on data structure.**	**4M**
	Ans:	**Need of data structure:** • Data structures are an important way of organizing information or data in a computer. • It has a different ways of storing & organizing data in a computer. • It helps to store data in logical manner. • It allows collection of data to grow & shrink dynamically over time & to organize the information so that one can access it using efficient algorithms. • Specific data structures are essential ingredients of many efficient algorithms, & they make possible management of huge amount of data, such as large collections of databases. **Operations perform on data structure:** • Insertion • Deletion • Searching • Sorting • Traversing • Merging	**(Need: 2marks, Any four application :1/2 mark each)**

b) Differentiate between binary search and linear search (Any four points). **4M**

Ans: {{**Note: any other relevant point shall be considered**}} **(Any four points:1 mark each)**

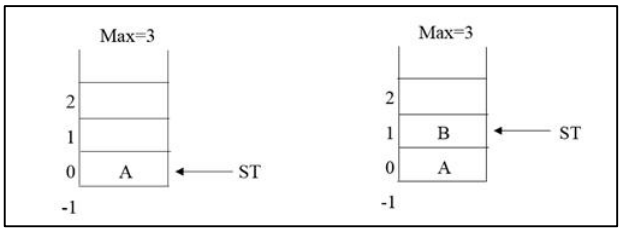
Linear Search	Binary Search
Linear search compares each element with search element, while searching an element in the list.	Binary search computes mid element. It compares mid element with search element, while searching element in the list.
Linear search works on both sorted and unsorted input list	Binary search works only on sorted list.
Linear search take more time to search as it compares with each element.	Binary search requires less time to search as it compares only mid element with search element. If they are not equal then it reduces the list for comparison.
linear search has complexity O(n)	Binary search has complexity O(log n)
Linear search supports sequential access of elements.	Binary search supports random access of elements.
Good technique for small set of elements	Binary search is recommended when number of elements are large

c) Explain four primitive operations on stack. **4M**

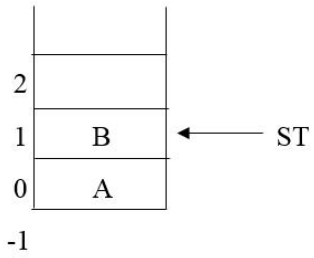
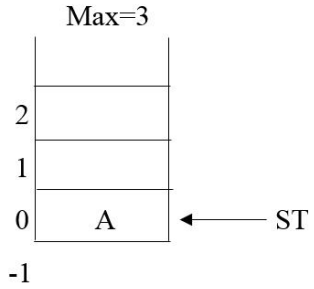
Ans:

1. push() : This operation is used to insert an element in a stack. Inserting an element in stack requires increment of stack top by one position and then store data element in it.

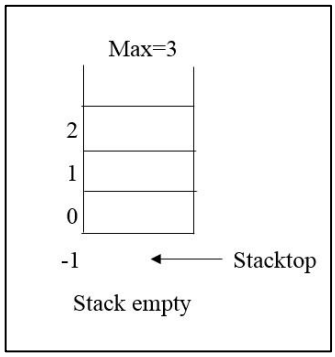
Push B



2. pop(): This operation is used to remove an element from stack. Removing an element from stack requires decrement of stack top by one position.

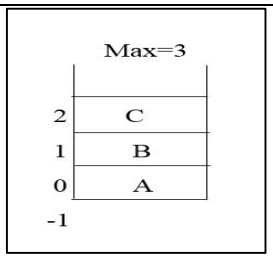
Before Pop	After Pop
	

3. isempty(): This operation is used to check whether stack is empty or not. It checks stack top position. If stack top is -1 then stack is empty.

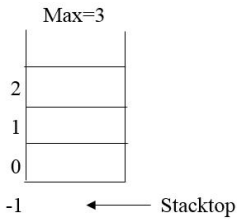


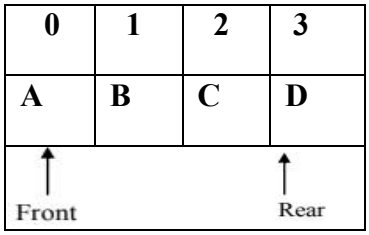
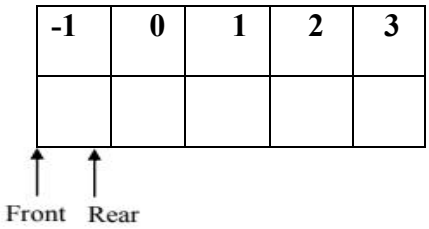
4. isfull(): This operation is used to check whether stack is full or not. It checks stack top position. If stack top is maximum size -1 then stack is full.

(Any four operations: 1 mark each)

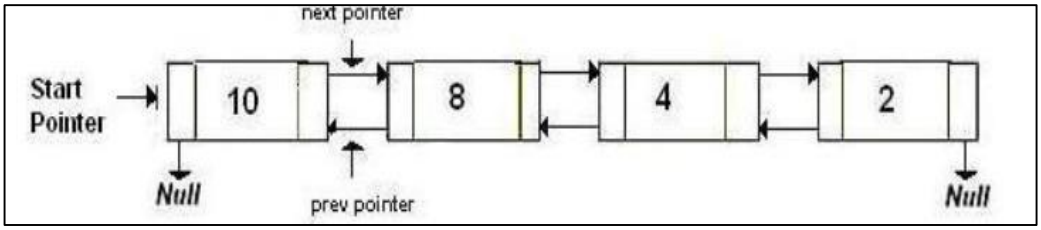


5. initialize(): This operation is used to initialize stack top to –1 value.



	d)	**Explain queue full and queue empty condition with suitable example.**	**4M**
	Ans:	**Queue full**:-A queue is full when its rear pointer points to max -1 position. Max is maximum number of elements in a queue. If rear pointer is not equal to max-1 then a new element can be added to a queue. **Example:-**  **Queue empty**: A queue is empty when its front pointer points to -1 position. When front pointer is -1 then one cannot delete any data from a queue. **Example:**  	**(Explanation of each term: 1mark, example of each: 1mark)**
	e)	**Describe doubly linked list with suitable example.**	**4M**
	Ans:	A doubly linked list is a linked list in which each node contains two links i.e double links. Each node in the list contains two pointers that store address of previous node and next node. It can traverse in any direction. It can access both the data elements. **Node structure:** prev. pointer | Data | next pointer Each node contains three parts. **1. prev. pointer:** It **c**ontains address of the previous node. **2. Data:** It contains information. Example: 10, 20, etc. **3. next pointer:** It **c**ontains address of the next node.	**(Description: 2marks, Example: 2 marks)**

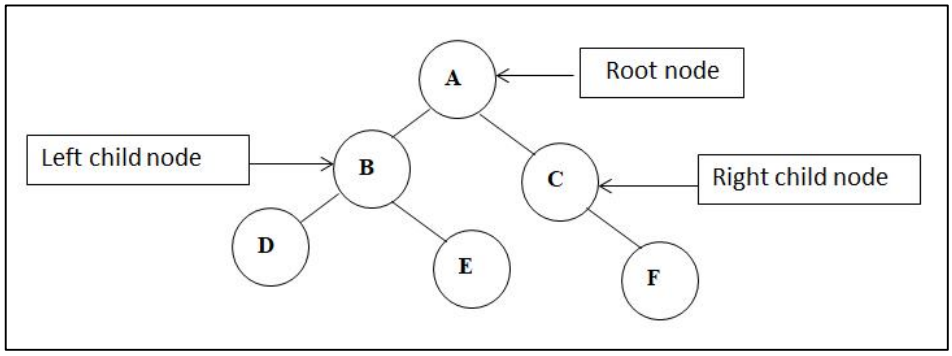
Example:



In the above example, each node has three fields. Previous pointer (first field) stores the address of previous node and next pointer (third field) stores address of next node. Second field stores information. First node from the list contains Null value in prev. pointer. Similarly last node from the list contains Null value in next field. Apart from these two, the other remaining nodes in between contains address of previous node in prev. pointer and address of next node in next pointer.

	f)	**Explain binary tree with an example.**	**4M**
	Ans:	A binary tree is a tree in which each non-leaf 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.	**(Explanation:2marks, Example:2 marks)**

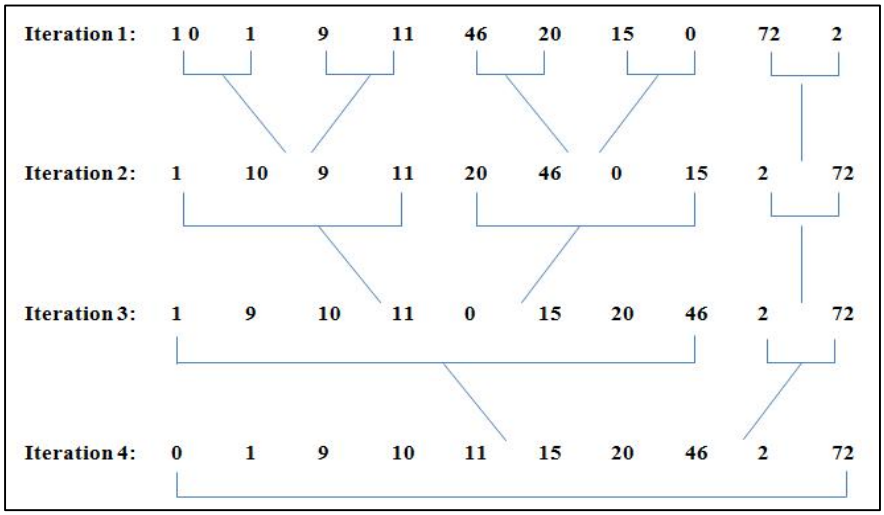
Example:



In the above example, Nodes A, B, C are non-leaf nodes. Node A and B has two child nodes whereas node C has only one child node. Nodes D, E, F are leaf node as they don't have any child node.
Node A is root node of tree. Node B is left child of root and node C is right child of root.

	g)	**Define hashing. Explain any one hashing method.**	**4M**
	Ans:	**Definition:** Hashing is a technique used to compute memory address for performing insertion, deletion and searching of an element using hash function. **Hashing methods:-** • Division method • Mid square method • Folding method **1. Division method:** In this method hash address is calculated by dividing the key value by a prime number or a number without small divisor. Formula: H(K)=K(mod m) or H(K)=K(mod m)+1 K- Specify unique key value. m- Specify is a prime number or number without small divisors. Example:- H(3205)= 3205 mod 97=4 **2. Middle square method:** In this method hash address is calculated by taking two digits from middle of square of key value. Formula: H(K)=I I-specify digits after deleting digits from both ends of K^2 Example:- H(3205)=$(3205)^2$=10272015=72 **3. Folding method:** In this method hash address is calculated by partitioning key into multiple parts and performing addition. Formula: H(K)=$K_1+K_2+...K_n$ Example:- H(3205)=32+05=37	**(Definition: 2marks, Explanation of any one method: 2marks)**
2.		**Attempt any four:**	**16 Marks**
	a)	**Describe working of merge sort with example.**	**4M**
	Ans:	All the elements from the input list are divided into groups. Each group is sorted independently and merged together in each iteration. These iterations continue performing divide, sort and merge procedure till all elements are placed in one single group. The final group can be sorted with any other sorting method to get a sorted list.	**(Description:2marks, example:2 marks)**

Example:- input list: 10,1,9,11,46,20,15,0,72,2



In the above example, input list is divided into group of two elements in iteration 1 and sorted in that group. In iteration 2, two groups from iteration 1 are merged together to form group of four elements. Again each group is sorted and merged together in iteration 3. In iteration 3, two groups of 4 elements are merged together to form group of eight elements. Then they are sorted and merge together to form a group of ten elements in iteration 4. Now all these elements are in a single group so they are sorted together with any other sorting algorithm.

Sorted list is: 0,1,2,9,10,11,15,20,46,72.

b) Write an algorithm for performing push operation on stack. **4M**

Ans:

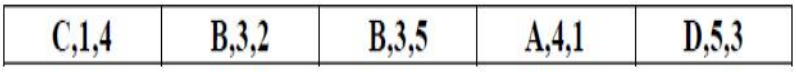
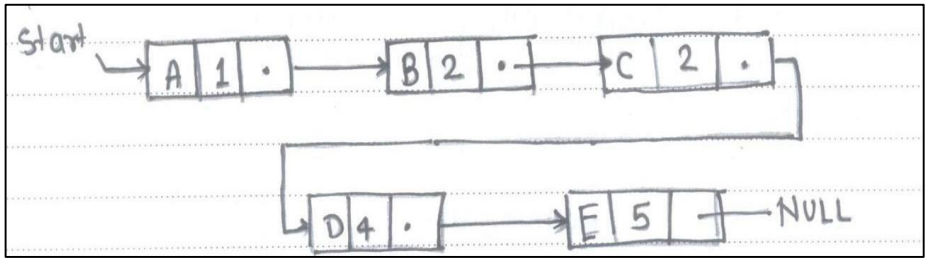
Push algorithm: - Max is maximum size of stack. **(Correct algorithm: 4marks)**

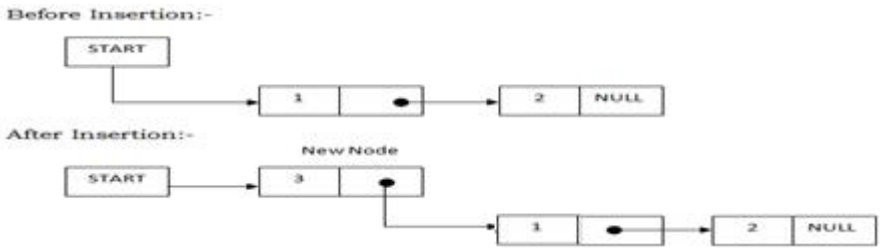
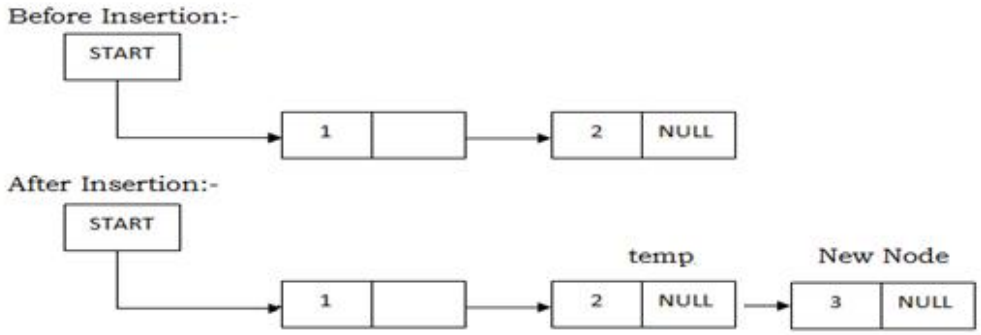
Step 1: [Check for stack full/ overflow]
If stack_top is equal to max-1 then
Display output as "Stack Overflow" and return to calling function
Otherwise
Go to step 2

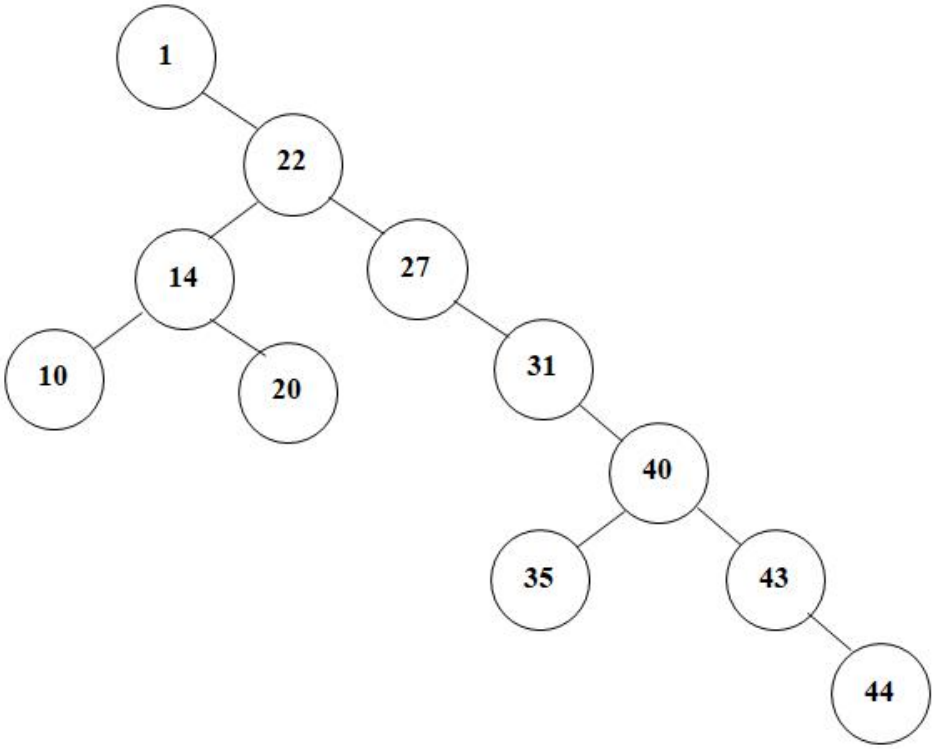
Step 2: [Increment stack_top]
Increment stack top pointer by one.
stack_top=stack_top +1;

Step 3: [Insert element]
stack [stack_top] = item;

Step 4: return to calling function

	c)	**Explain concept of priority queue with example.**	**4M**
	Ans:	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. The rules for processing the elements of priority queue are: 1) An element with 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 (FCFS). One of the examples of priority queue is a queue used in operating system. The operating system has to handle a large number of jobs. These jobs have to be properly scheduled. The operating system assigns priorities to each type of job. The jobs are placed in a queue and the job with the highest priority will be executed first. **Example:** **(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 is as shown below:-  In the above diagram, each structure element has three members as information, priority and order in which element is arrived in the list. **OR** **Linked representation:**  In the above diagram priority queue with 5 elements is shown. Each node in above priority queue contains three fields: i. Information field INFO ii. A priority number PR No iii. Link Next	**(Explanation-2marks, Example:2 marks)**

	d)	**Explain insertion at the beginning and at end operations on linked list with example.**	**4M**
	Ans:	**Inserting node at the beginning:** 1. Create a New Node with two fields as "Data" and "Next". 2. Store information (data) into "Data Field". 3. Store address from start node (first node address) in its "Next field". 4. Make the new node as first node by storing its address start pointer.  **Inserting node at the end:** 1. Create a New Node with two fields as "Data" and "Next". 2. Store information (data) into "Data Field". 3. Store NULL value in "Next field". 4. Traverse the list up to the last node (temp). 5. Store address of New Node inside "Next" field of temp node to make the New Node last node in the list. 	**(Explanation with example of each: 2marks)**
	e)	**Construct Binary Search Tree:** **1, 22, 27, 14, 31, 40, 43, 44, 10, 20, 35.**	**4M**
	Ans:	Binery search tree	**(Correct construction : 4marks)**



	f)	**Write algorithm for breadth first search.**	**4M**
	Ans:	**Algorithm for BFS**: Step 1. Initialize all nodes to ready state. Step 2. Insert starting node in a queue and change its state to waiting state. Step 3. Repeat steps 4 to 6 till the queue becomes empty. Step 4. Remove front node N from queue and change its status to be visited. Add this node N to list of reachable nodes and add its origin to origin list. Step 5. Insert all adjacent nodes of N at the rear end of the queue and change their status to waiting state. Step 6. From the origin find path from source node to destination node or from the queue element list find all nodes that are reachable. Step 7. Stop.	**(Correct steps of algorithm: 4 marks)**
3.		**Attempt any two:**	**16 Marks**
	a)	**Sort number in ascending order using bubble sort.** **19, 2, 27, 3, 7, 5, 31.**	**8M**
	Ans:	**{{**Note : Pass 4 onwards optional**}}** **Pass 1:** 2,19,27,3,7,5,31 2,19,27,3,7,5,31 2,19,3,27,7,5,31 2,19,3,7,27,5,31 2,19,3,7,5,27,31 **Pass 1 Completed** **Pass 2:**	**(Correct Sort: 8 marks, Stepwise marks shall be given)**

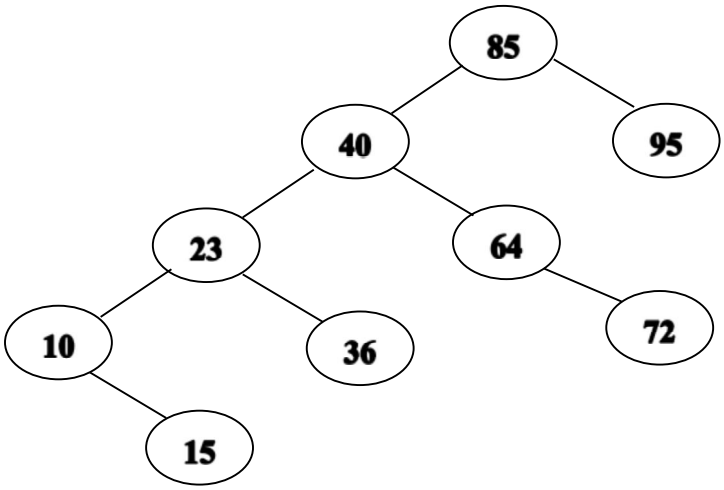
2,19,3,7,5,27,31
2,3,19,7,5,27,31
2,3,7,19,5,27,31
2,3,7,5,19,27,31
2,3,7,5,19,27,31
Pass 2 Completed
Pass 3:
2,3,7,5,19,27,31
2,3,7,5,19,27,31
2,3,5,7,19,27,31
Pass 3 Completed
Pass 4:
2,3,5,7,19,27,31
Pass 4 Completed
Pass 5:
2,3,5,7,19,27,31
Pass 5 Completed
Pass 6:
2,3,5,7,19,27,31
Pass 6 Completed

b) Convert the given infix expression in to postfix expression. Write all steps for conversion (a * b + c/d) * (e + f ↑ g). **8M**

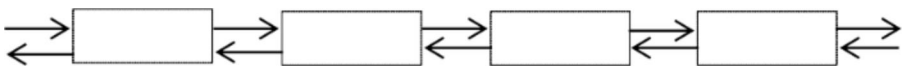
Ans:

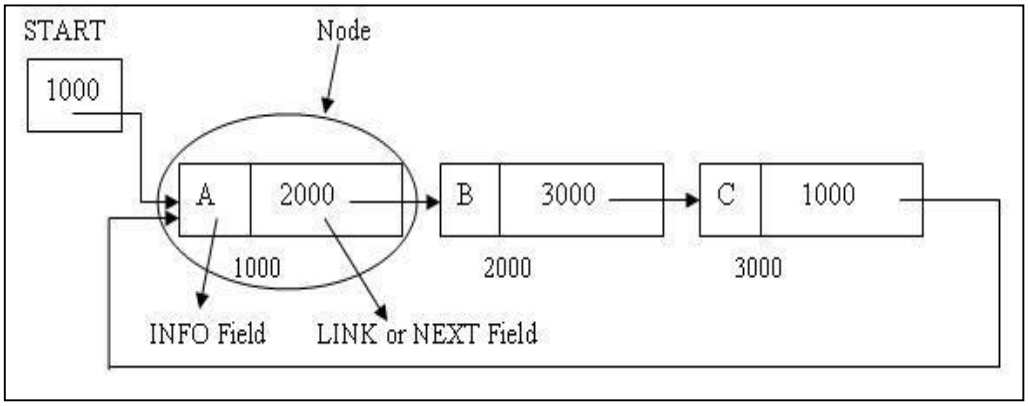
Sr. No	Symbol Scanned	Stack	Postfix Expression
1	(	(	
2	(	((	
3	a	((	a
4	*	((*	a
5	b	((*	ab
6	+	((+	ab*
7	c	((+	ab*c
8	/	((+/	ab*c
9	d	((+/	ab*cd
10	)	(	ab*cd/+
11	*	(*	ab*cd/+

(Correct Expression :8 marks, Stepwise marks shall be given)

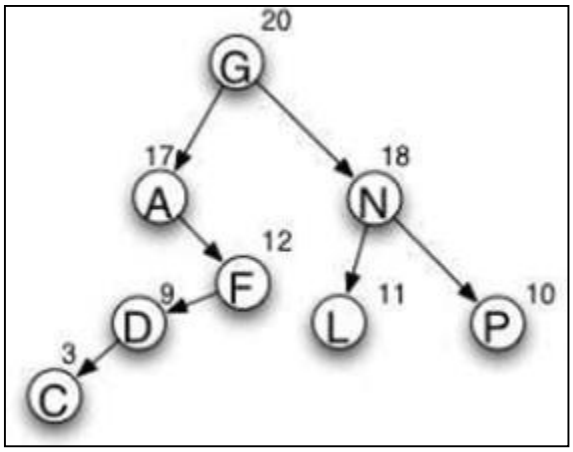
			12	(	(*(	ab*cd/+	
			13	e	(*(	ab*cd/+e	
			14	+	(*(+	ab*cd/+e	
			15	f	(*(+	ab*cd/+ef	
			16	↑	(*(+↑	ab*cd/+ef	
			17	g	(*(+↑	ab*cd/+efg	
			18	)	(*	ab*cd/+efg↑+	
			19	)	nil	ab*cd/+efg↑+*	
	c)	**Define the term tree traversal. Traverse the following tree in Inorder, Preorder and Postorder.** 				**8M**
	Ans:	**Definition:** Tree traversal is the process of visiting each node in a tree, such as a binary tree or binary search tree, exactly once. **Inorder Traversal:** 10,15,23,36,40,64,72,85,95 **Preorder Traversal:** 85,40,23,10,15,36,64,72,95 **Postorder Traversal:** 15,10,36,23,72,64,40,95,85				**(Defination :1 mark, Inorder:3 marks, Preorder:2 mark, Postorder: 2 marks)**

4.	**a)**	**Attempt any four:**	**16 Marks**
	a)	**Define algorithm. How it is analysed?**	**4M**
	Ans:	**{{**Note: Any other relevant answer shall be considered**}}** **Algorithm:** An algorithm is a step by step set of instructions designed to perform a specific task. **There are different types of time complexities which can be analyzed for an algorithm:** **Best Case Time Complexity:** It is measure of minimum time that algorithm will require for input of size "n". Running time of many algorithms varies not only for inputs of different sizes but also input of same size. For example in running time of some sorting algorithms, sorting will depend on ordering of input data. Therefore if input data of "n" items is presented in sorted order, operations performed by algorithm will take least time. **Worst Case Time Complexity:** It is measure of maximum time that algorithm will require for input of size "n" Therefore if various algorithms for sorting are taken into account & say "n" input data items are supplied in reverse order for any sorting algorithm, then algorithm will require n^2 operations to perform sort which will correspond to worst case time complexity of algorithm. **Average Case Time Complexity:** The time that an algorithm will require to execute typical input data of size "n" is known as average case time complexity. We can say that value that is obtained by averaging running time of an algorithm for all possible inputs of size "n" can determine average case time complexity. Computation of exact time taken by algorithm for its execution is very difficult. Thus work done by algorithm for execution of input of size "n" defines time analysis as function f(n) of input data items.	**(Definition- 1 mark, Description of algorithm analysis - 3 marks)**
	b)	**Find the position of element '29' using binary search method. Show all steps. A= {11,5,21,3,29,17,2,43}.**	**4M**
	Ans:	Pre-condition for Binary search is array elements must be sorted in ascending order. In given example array elements are not sorted. Applying Bubble sort we sort the elements of array.	**(Correct position with all correct steps: 4 marks,**

		Sorted array A= {2, 3, 5, 11, 17, 21, 29, 43} Step 1) Low=0, high=7, k=29 Mid= (0+7)/2 = 3 A[mid] =a [3] =11 29>11 k>a [mid] Step 2) Low=mid+1 High=7 Mid= (4+7)/2=5 A[mid] =a [5] =21 29>21 k>a [mid] Step 3) Low=mid+1 Mid= (6+7)/2=6 A[mid] =a [6] =29 A[mid] =k Therefore key element is found at 6th position, no. of comparison required = 3. Search is successful	**Stepwise marks shall be considered)**
	c)	**Explain the concept of dequeue with example.**	**4M**
	Ans:	1. A double-ended queue or dequeue is an abstract data structure that implements a queue for which elements can only be added to or removed from the front (head) or back (tail). 2. It is also often called a head-tail linked list. 3. Dequeue is a special type of data structure in which insertions and deletions will be done either at the front end or at the rear end of the queue. 4. The operations that can be performed on dequeues are a. Insert an item from front end b. Insert an item from rear end c. Delete an item from front end d. Delete an item from rear end e. Display the contents of queue Rear Front  Front Rear	**(Explanation: 3 marks, Diagram:1 mark)**

	d)	**Describe advantage of circular linked list over linear linked list with example.**	**4M**
	Ans:	In Circular Linked List last node does not contain NULL pointer. Instead the last node contains a pointer that has the address of first node and thus points back to the first node. **Example:** It is shown below:  **Advantages:** • The entire list can be traversed starting from any end which is not possible in single linked list • In singly linked list to delete desired node, it is necessary to give the address of first node of the list. This necessity results from the fact that in order to delete desired node. The predecessor of this node has to be found. • Maximum utilization of space allotted to linked list.	**(Any 2 advantages : 2 marks, example:2 marks)**
	e)	**Describe weight balanced tree and height balanced tree with example.**	**4M**
	Ans:	**Weight balanced trees:** Weight-balanced tree is a binary tree which is balanced based on knowledge of the probabilities of searching for each individual node. Within each subtree, the node with the highest weight appears at the root.	**(Weight Balanced Tree:2 marks, Height Balanced Tree:2 marks)**

Example:



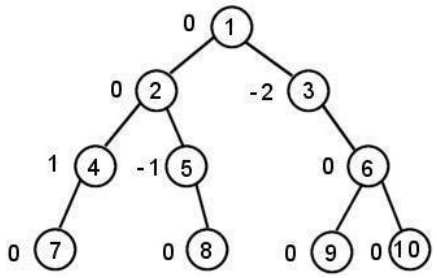
In the diagram to the right, the letters represent node values and the numbers represent node weights. Values are used to order the tree, as in a general binary search tree. The weight may be thought of as a probability or activity count associated with the node. In the diagram, the root is G because its weight is the greatest in the tree. The left subtree begins with A because, out of all nodes with values that come before G, A has the highest weight. Similarly, N is the highest-weighted node that comes after G.

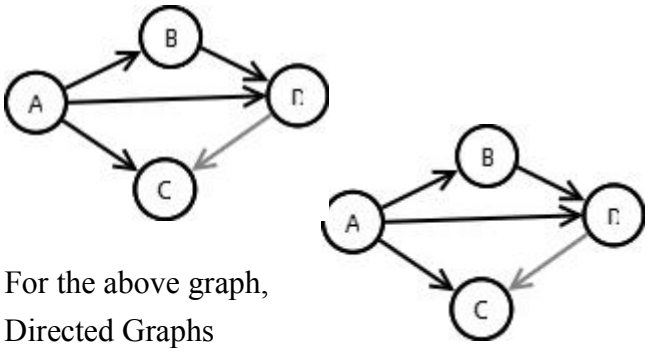
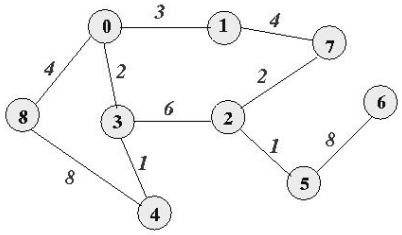
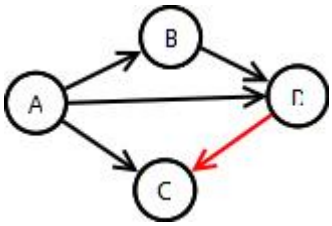
Height balanced trees

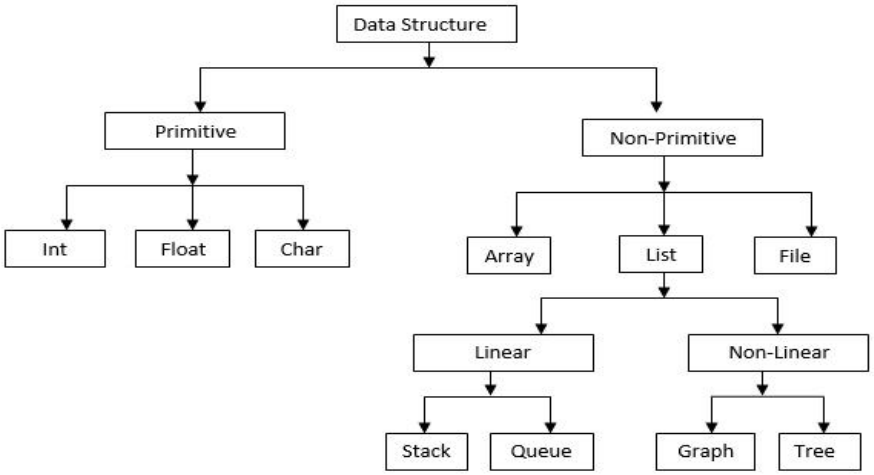
AVL trees are binary search trees, which have the balance propriety. The balance property is true for any node and it states: "the height of the left subtree of any node differs from the height of the right subtree by 1".
The binary tree is balanced when all the balancing factors of all the nodes are -1,0,+1.

Formally, we can translate this to this: | hd – hs | ≤ 1, node X being any node in the tree, where hs andhd represent the heights of the left and the right subtrees.

Example:



	f)	**Define following terms w.r.t graph-indegree of node, directed graph, weighted graph, predecessor.**	**4M**
	Ans:	**In-degree of a node:** Total number of edges coming towards a node is said to be in-degree of that node.  For the above graph, in-degree of A is 0 and B is 1 Directed Graphs A graph whose edges are directed (i.e. have a direction) is called as directed graph. It is also called digraph Edge drawn as arrow Edge can only be traversed in direction of arrow Example: E = {(A,B), (A,C), (A,D), (B,C), (D,C)} **Weighted graph** A graph whose edges are assigned a non-negative numerical values is called as weighted graph. The weight of an edge can represent: Cost or distance = the amount of effort needed to travel from one place to another Capacity = the maximum amount of flow that can be transported from one place to another **Example:**  **Predecessor** Predecessor of a node is the node which is connected to it with an incoming edge.  Predecessor of node B is A and Predecessor of C is A and D	**(In-degree of a node: 1 mark, directed graph: 1 mark, weighted graph: 1 mark, predecessor: 1 mark)**

5.		**Attempt any four :**	**16 Marks**
	a)	**Describe with example classification of data structure.**	**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 retrieves from memory. **Example:** int, char, float **Non-primitive data structure**: All the data structures derived from primitive 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.	**(Correct classification: 2 marks, Example of each: 2marks)**
	b)	**Describe working of radix sort along with example.**	**4M**
	Ans:	**Radix Sorting**: - In this method, ten buckets (0-9) are used to sort elements of an input list. All the elements are sorted according to their digit position from each element. In pass one each element is placed inside the bucket with respect its unit position digit. After placing all elements inside the buckets, read those from 0th bucket to 9th bucket. In pass 2, elements are placed in buckets with respect to	**(Radix sort Explanation: 2 marks, Any suitable**

10th position digit from each element. In each pass one position is considered to arrange all the elements in bucket. At the end of each pass elements are collected from buckets and given as input to the next pass. Total number of passes required for sorting is equal to maximum number of digits present in the largest number from the input list. Last pass gives sorted list after reading all elements from 0th bucket to 9th bucket.

Example: 18,253, 1000,2 ,80,75,58

Pass1: In this pass arrange the element according to unit place.

	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.

	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

Example: 2 marks)

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

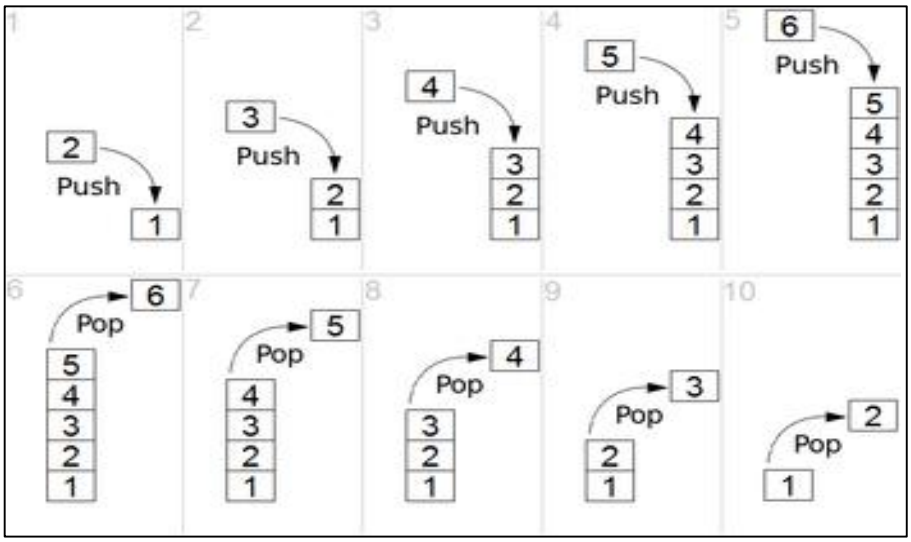
	0	1	2	3	4	5	6	7	8	9
1000	1000									
2	2									
18	18									
253			253							
58	58									
75	75									
80	80									

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

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

	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)	**Describe how recursion is used in reversal of list.**	**4M**
	Ans:	A simple application of stack is reversal of list. To reverse a list, the elements of list are pushed onto the stack one by one as the element read from first to last. Once all elements are pushed on the stack they are popped one by one. Since the element last pushed in comes out first, hence reversal of string occurs. Consider following example where a list contains elements as {1, 2, 3, 4, 5, 6}. Every push operator will push an element on top of stack. Once all elements are pushed one can pop all elements and save it which results in to reversing of list as {6, 5, 4, 3, 2, 1}.  	**(Correct Explanation: 4 marks)**
	d)	**Distinguish between stack and queue (Any four points).**	**4M**
	Ans:		**(Any four points: 1 mark each)**

Stack	Queue
1. In Stack insertion and deletion operations are performed at same end.	1. In Queue insertion and deletion operations are performed at different end.
2.In stack the element which is inserted last is first to delete so it is called Last In First Out	2.In Queue the element which is inserted first is first to delete so it is called First In First Out
3.In stack only one pointer is used called as Top	3.In Queue two pointers are used called as front and rear
4.In Stack Memory is not wasted	4. In Queue memory can be wasted/ unusable in case of linear queue.

		5. Stack of books is an example of stack	5. Students standing in a line at fees counter is an example of queue	
		6.Application:Recursion,Polish notation	6. Application: In computer system for organizing processes. In mobile device for sending receiving messages.	
	e)	Write algorithm for 'search' operation in an unsorted linked list.		**4M**
	Ans:	**{{**Note: Any set of correct steps shall be considered**}}** Algorithm SEARCH (INFO, LINK, START, ITEM, LOC) LIST is a linked list in memory. This algorithm finds the location LOC of the node where ITEM first appears in LIST or sets LOC =NULL. 1) Set PTR := START 2) Repeat Step 3 while PTR !=NULL 3) If ITEM = INFO[PTR], then Set LOC: = PTR, and exit; Else Set PTR: = LINK [PTR]. (PTR now points to the next node) [End of if loop] 4) [Search is unsuccessful.] set LOC:= NULL. 5) **Exit.**		**(Correct Algorithm: 4 marks)**
	f)	**State any four applications of graph. Describe any one in detail.**		**4M**
	Ans:	**Applications of graph:** 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 1. **Social Network Graphs:** to tweet or not to tweet. Graphs that represent who knows whom, who communicates with whom, who influences whom or other relationships in social structures. An example is the twitter graph of who follows whom. These can be used to determine how information flows, how topics become hot, how communities develop, or even who might be a good match for who, or is that whom. 2. **Transportation networks:** In road networks vertices are intersections and edges are the road segments between them, and for public transportation networks vertices are stops and edges are the links between them. Such networks are used by many map programs such as Google maps, Bing maps and now Apple IOS 6 maps (well perhaps without the public transport) to		**(Any four application : ½ mark Each, Any one application : 2 marks)**

		find the best routes between locations. They are also used for studying traffic patterns, traffic light timings, and many aspects of transportation. 3. **Neural networks:** Vertices represent neurons and edges the synapses between them. Neural networks are used to understand how our brain works and how connections change when we learn. The human brain has about 1011 neurons and close to 1015 synapses. 4. **Utility graphs:** The power grid, the Internet, and the water network are all examples of graphs where vertices represent connection points, and edges the wires or pipes between them. Analysing properties of these graphs is very important in understanding the reliability of such utilities under failure or attack, or in minimizing the costs to build infrastructure that matches required demands. 5. **Network packet traffic graphs:** Vertices are IP (Internet protocol) addresses and edges are the packets that flow between them. Such graphs are used for analysing network security, studying the spread of worms, and tracking criminal or non-criminal activity. **6. Graphs in compilers**: Graphs are used extensively in compilers. They can be used for type inference, for so called data flow analysis, register allocation and many other purposes.	
6.		**Attempt any two :**	**16 Marks**
	a)	**Define recursion. Write any two advantages of recursion. Write 'C' program to calculate the factorial of number using recursion.**	**8M**

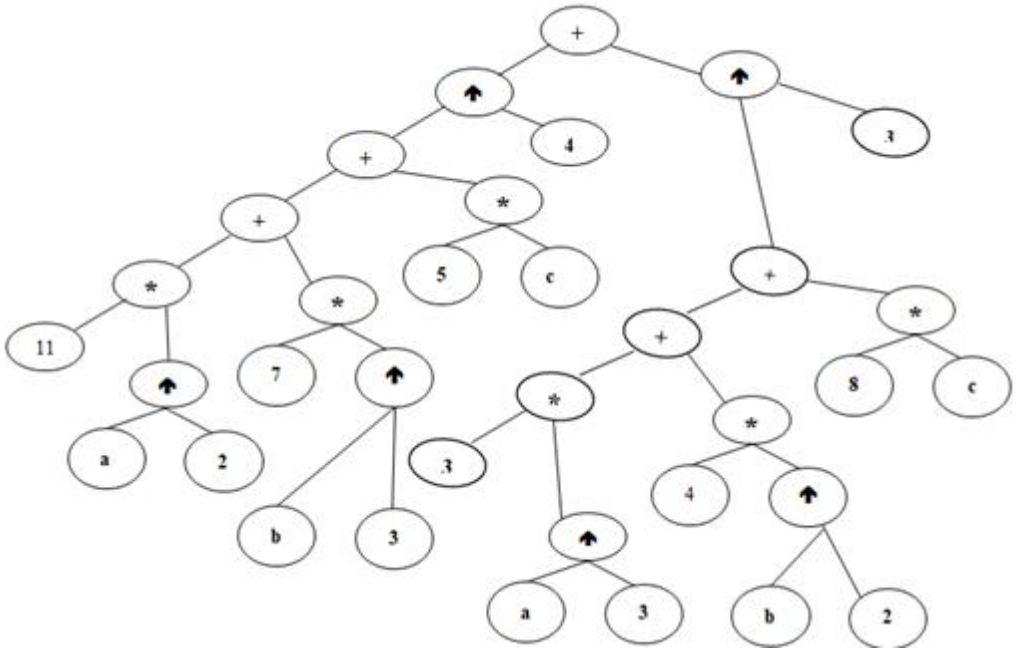
	Ans:	**Definition:** Recursion is the process of calling function by itself. A recursive function body contains function call statement that calls itself repeatedly.
		
		Advantages:
		
		• The main benefit of a recursive approach to algorithm design is that it allows programmers to take advantage of the repetitive structure present in many problems.
		• Complex case analysis and nested loops can be avoided.
		• Recursion can lead to more readable and efficient algorithm descriptions.
		• Recursion is also a useful way for defining objects that have a repeated similar structural form.
		• Using recursion, the length of the program can be reduced.
		
		Program:
		#include<stdio.h>
		#include<conio.h>
		int fact(int n);
		void main()
		{
		int n;
		clrscr();
		printf("\nThe factorial of % is = %d",n,fact(n));
		getch();
			(Definition: 2 marks, any two advantages : 1 mark each, Factorial program: 4 marks)

```
}
int fact(int n)
{
if(n==1)
return 1;
else
return(n*fact(n-1));
}
```

b) Define expression tree. Draw tree structure for following expression: $(11a^2 + 7b^3 + 5c)^4 + (3a^{3+} 4b^2 + 8c)^3$ — **8M**

Ans:

Definition: Expression trees are a special kind of binary tree used to evaluate certain expressions.

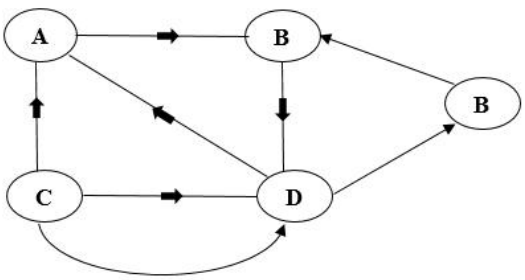


(Definition: 1mark, Correct expression tree: 7 Marks)

c) For following graph give — **8M**

i) adjacency matrix representation

ii) adjacency list representation.

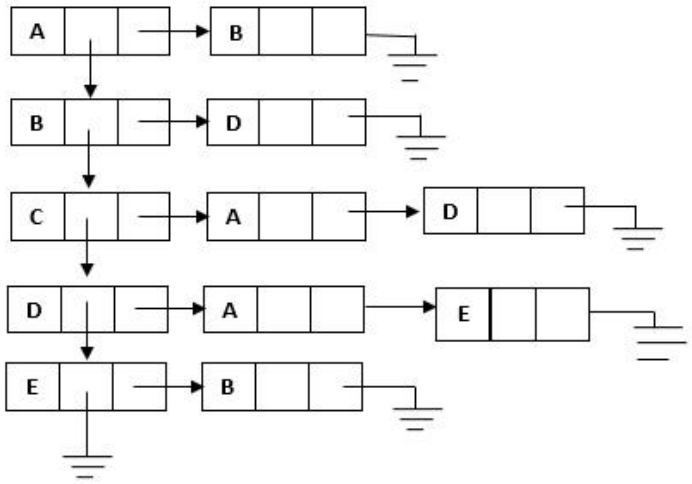


Ans:

i) Adjacency matrix A for the above graph.

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

ii) Adjacency list representation of above graph.



Adjacency List

Node	Adjacency List
A	B
B	D
C	A , D
D	A,E
E	B

(Adjacency matrix representation : 4 marks Adjacency list representation: : 4 marks)