SCHEME OF EVALUATION

(Scoring indicators)

Revision: 2015 Course Code :4133

Course Tittle: Data Structures

Question No.	Scoring indicator **PART-A**	Split Up Score	Sub Total	Total
I (1)	Insertion,Deletion, Search ,Traverse ,Sort etc...	2	2	10
(2)	A double-ended queue is a special type of data structure in computer programming. In this abstract data type, elements can be added from both the front and the back of the queue. Due to this property, it is also known as a head-tail linked list.	2	2	
(3)	A list contains elements of same type arranged in sequential order and following operations can be performed on the list. get() ,insert(),remove() ,removeAt() and replace().	2	2	
(4)	Single Threaded Binary tree and Double Threaded Binary tree	2	2	
(5)	A graph is a set of objects (called vertices or nodes) that are connected together. The connections between the vertices are called edges or links.	2	2	
PART-B				
II (1)	(A) 4 (B) 37	3 3	6	30
(2)	(1)Stack is used to pass parameter Between function. (2)Implementation of recursion in programming languages (3)Revering a string	6	6	

	(4)Infix expression to post fix expression conversion (5)Post fix Expression evaluation (6)Infix expression to prefix expression conversion (7)Any other types of application such as Last in First out operation manner			

(3)	CircularQueueInsert(Q,N,F,R,x)
	{ if((R==N)&(F==1))
	Printf(Overflow);
	Exit()
	Else
	{ if(R==N)
	R=1
	Else
	R=R+1
	Q[R]=x
	If(F==0)
	F=1;
	}}
		6	6	

(4)	structNode {
	intdata;
	structNode* link;
	};
	structNode* top;
	
	voidpush(intdata)
	{
	structNode* temp;
	temp = (structNode*)malloc(sizeof(structNode));
	
	if(!temp) {
	printf("\nHeap Overflow");
	exit(1);
	temp->data = data;
	
	temp->link = top;
	top = temp;
	}
		6	6	

```
voidpop()
{
    structNode* temp;

    if(top == NULL) {
        printf("\nStack Underflow");
        exit(1);
    }
    else{
        temp = top;
        top = top->link;
        temp->link = NULL;
        free(temp);
    }
}
```

(5)	To delete a node from BST, there are three possible cases to consider: **Case 1:** Deleting a node with no children: simply remove the node from the tree. **Case 2:** Deleting a node with two children: call the node to be deleted *N*. Do not delete *N*. Instead, choose either its in-order *successor* node or its in-order *predecessor* node, *R*. Copy the value of *R* to *N*, then recursively call delete on *R* until reaching one of the first two cases. If you choose in-order successor of a node, as right sub tree is not NULL (Our present case is node has 2 children), then its in-order successor is node with least value in its right sub tree, which will have at a maximum of 1 sub tree, so deleting it would fall in one of the first 2 cases. **Case 3:** Deleting a node with one child: remove the node and replace it with its child.	2 2 2	6	
(6)	<u>DFS</u> . <u>BFS</u> 1)Recursive procedure used .Iterative procedure used 2)Stack is the Data structure.Queue is the Data structure 3)Pre order style traversal .Level order style traversal 4)Some vertex pushed more than one into stack all vertex inserted exactly once into queue 5)Some vertex not pushed on to the stack . all vertex inserted into queue 6)Termination condition is stack is empty Termination condition is queue is empty	3 3	6	
(7)	Bubble sort is a sorting algorithm that works by repeatedly			

<table>
<tr><td></td><td>stepping through lists that need to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. This passing procedure is repeated until no swaps are required, indicating that the list is sorted. Bubble sort gets its name because smaller elements bubble toward the top of the list.Bubble sort is also referred to as sinking sort or comparison sort.
<pre>voidbubbleSort(intarr[], intn)
{
   inti, j;
   for(i = 0; i < n-1; i++)

       for(j = 0; j < n-i-1; j++)
           if(arr[j] > arr[j+1])
              swap(&arr[j], &arr[j+1]);
}</pre></td><td>2

4</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="5"><u>PART-C</u></td></tr>
<tr><td>III (a)</td><td>Algorithm to convert Infix To Postfix

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

1. Push “(“onto Stack, and add “)” to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
    2. Add operator to Stack.
    [End of If]
6. If a right parenthesis is encountered ,then:</td><td>10</td><td>15</td><td>60</td></tr>
</table>

	1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered. 2. Remove the left Parenthesis. [End of If] [End of If] 7.END.			
(b)	ABC * DE/F-G*-H*+	5		
OR				
IV (a)	Evaluation rule of a Postfix Expression states: 1. While reading the expression from left to right, push the element in the stack if it is an operand. 2. Pop the two operands from the stack, if the element is an operator and then evaluate it. 3. Push back the result of the evaluation. Repeat it till the end of the expression. Algorithm 1) Add) to postfix expression. 2) Read postfix expression Left to Right until) encountered 3) If operand is encountered, push it onto Stack [End If] 4) If operator is encountered, Pop two elements i) A -> Top element ii) B-> Next to Top element iii) Evaluate B operator A ,push B operator A onto Stack 5) Set result = pop 6) END	10	15	
(b)	Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called **'Ring Buffer'**. In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we can not insert	3		

	the next element even if there is a space in front of queue Disadvantage of Linear/Static Queue- When any element is inserted in linear queue then rear will be increased by 1. Let, assume after insertion operations rear is shifted to last position in queue. It means, now queue is full. Now if a new element is inserted then overflow condition will occur.	2		
V (a)	• head and tail are two pointers, *where* head points to first node of linked list and tailpoints the las node of the linked list. • A Node contains two parts 1. item - item contains data item (it may be a number, string or any object like structure). 2. next - next contains address of the next node. • Last Node of the linked list contains NULL in the next part.	10	15	

(b)	`void addAtHead(List * lp,int item){`
	`Node * node;`
	`node = createNode(item);`
	
	`if(lp->head ==NULL)`
	`{`
	`lp->head = node;`
	`lp->tail = node;`
	`}`
	`else`
	`{`
	`node->next = lp->head ;`
	`lp->head = node ;`
	`}`
	`}`
		5		

	A linked list is a linear data structure, in which the elements			

	are not stored at contiguous memory locations. The elements in a linked list are linked using pointers **Applications of linked list in computer science –** 1. Implementation of stacks and queues 2. Implementation of graphs 3. Dynamic memory allocation : 4. Maintaining directory of names 5. Performing arithmetic operations on long integers 6. Manipulation of polynomials by storing constants in the node of linked list			
OR				

VI (a)	**struct** Node{
	int data;
	    Node *next;
	    };
	void Queue::insert(**int** n){
	  Node *temp=**new** Node;
	if(temp==NULL){
	cout<<"Overflow"<<endl;
	return;
	}
	  temp->data=n;
	  temp->next=NULL;
	
	if(front==NULL){
	    front=rear=temp;
	}
	else{
	    rear->next=temp;
	    rear=temp;
		10	15	

(b)	}
	cout<<n<<" has been inserted successfully."<<endl;}
	void Queue :: deleteitem()
	{
	if front==NULL){
	cout<<"underflow"<<endl;
	return;
	}
	
	cout<<front->data<<" is being deleted "<<endl;
	if(front==rear)//if only one node is there
	front=rear=NULL;
	else
	front=front->next;}
	void Queue::display(){
	if(front==NULL){
	cout<<"Underflow."<<endl;
	return;
	}
	Node *temp=front;
	while(temp){
	cout<<temp->data<<"";
	temp=temp->next;
	}
	}
		5		

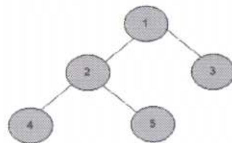
VII (a)	Few disadvantages of linked lists are : 1. They use more memory than arrays because of the storage used by their pointers. 2. Difficulties arise in linked lists when it comes to	10	15	

reverse traversing. For instance, singly linked lists are cumbersome to navigate backwards and while doubly linked lists are somewhat easier to read, memory is wasted in allocating space for a back-pointer.

3. Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential access.
4. Nodes are stored incontiguously, greatly increasing the time required to access individual elements within the list, especially with a CPU cache.

....

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

Inorder Traversal:

Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

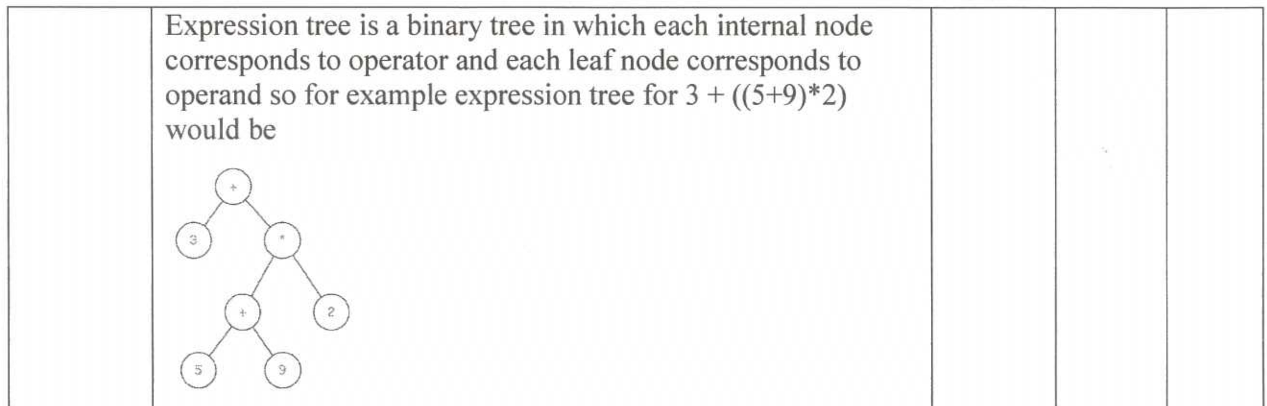
```
voidprintInorder(structnode* node)
{
  if(node == NULL)
     return;
  printInorder(node->left);
  printf("%c ", node->data);
  printInorder(node->right);
}
```

Uses of Inorder
In case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder traversal s reversed can be used.
Eg: Inorder traversal for the above-given figure is 4 2 5 1 3.

Preorder Traversal:

(b)	Algorithm Preorder(tree)
	1. Visit the root.
	2. Traverse the left subtree, i.e., call Preorder(left-subtree)
	3. Traverse the right subtree, i.e., call Preorder(right-subtree)
	voidprintPreorder(**struct**node* node)
	{
	if(node == NULL)
	return;
	printf("%c ", node->data);
	printInorder(node->left);
	printInorder(node->right);
	}
	Uses of Preorder
	Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree.Eg: Preorder traversal for the above given figure is 1 2 4 5 3.
	Postorder Traversal:
	Algorithm Postorder(tree)
	1. Traverse the left subtree, i.e., call Postorder(left-subtree)
	2. Traverse the right subtree, i.e., call Postorder(right-subtree)
	3. Visit the root.
	voidprintPostorder(**struct**node* node)
	{
	if(node == NULL)
	return;
	printInorder(node->left);
	printInorder(node->right);
	printf("%c ", node->data);
	}
	Uses of Postorder
	Postorder traversal is used to delete the tree. Postorder traversal is also useful to get the postfix expression of an expression tree.Eg: Postorder traversal for the above given figure is 4 5 2 3 1.
	Expression Tree:
		5		

Expression tree is a binary tree in which each internal node corresponds to operator and each leaf node corresponds to operand so for example expression tree for 3 + ((5+9)*2) would be



OR

VIII (a)	Binary Search Tree is a node-based binary tree data structure which has the following properties:	10	15	

The left subtree of a node contains only nodes with keys lesser than the node's key.
The right subtree of a node contains only nodes with keys greater than the node's key.
The left and right subtree each must also be a binary search tree.

Insertion of a key

A new key is always inserted at leaf. We start searching a key from root till we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.

```
     100                               100
    /   \        Insert 40            /    \
  20     500    --------->          20      500
 /  \                              /  \
10   30                           10   30
                                         \
                                          40
```

Insertion algorithm

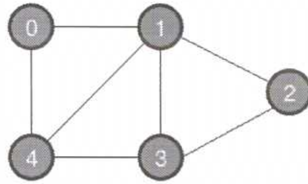
```
structnode* insert(structnode* node, intkey)
{
    if(node == NULL) returnnewNode(key);

    if(key < node->key)
        node->left  = insert(node->left, key);
    elseif(key > node->key)
        node->right = insert(node->right, key);
        returnnode;
}
```

(b)	Preorder Traversal:
	
	`Algorithm Preorder(tree)`
	`   1. Visit the root.`
	`   2. Traverse the left subtree, i.e., call Preorder(left-subtree)`
	`   3. Traverse the right subtree, i.e., call Preorder(right-subtree)`
	
	voidprintPreorder(**struct**node* node)
	{
	  if(node == NULL)
	    return;
	printf("%c ", node->data);
	  printInorder(node->left);
	  printInorder(node->right);
	}
	Uses of Preorder
	Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree.Eg: Preorder traversal for the above given figure is 1 2 4 5 3.
		5		

IX (a)	A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph can be defined as, *A Graph consists of a finite set of vertices(or nodes) and set of Edges which connect a pair of nodes.* Following two are the most commonly used representations of a graph. **1.** Adjacency Matrix **2.** Adjacency List There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use. **Adjacency Matrix:** Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.	4	15	

Following is an example of an undirected graph with 5 vertices.



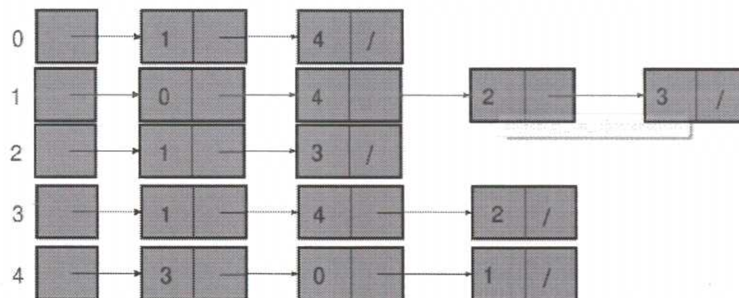
The adjacency matrix for the above example graph is:

V	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

4

Adjacency List:

An array of lists is used. Size of the array is equal to the number of vertices. Let the array be array[]. An entry array[i] represents the list of vertices adjacent to the *i*th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs. Following is adjacency list representation of the above graph.



(b) 7

QuickSort

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

Always pick first element as pivot.
The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Algorithm for recursive QuickSort function :

```
quickSort(arr[], low, high)
{
    if (low < high)
    {
        pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

code for partition()

This function takes last element as pivot, places the pivot element at its correct position in sorted array, and places all smaller (smaller than pivot) to left of pivot and all greater elements to right of pivot */

```
partition (arr[], low, high)
{
  pivot = arr[high];


  i = (low - 1)
  for (j = low; j <= high- 1; j++)
  {
 if (arr[j] <= pivot)
    {
       i++;
       swap arr[i] and arr[j]
    }
  }
  swap arr[i + 1] and arr[high])
  return (i + 1)
}
```

OR

X (a) | 10 | 15

Binary Search: Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Recursive implementation of Binary Search

We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
4. Else (x is smaller) recur for the left half.

Algorithm

```
intbinarySearch(intarr[], intl, intr, intx)
{
    if(r >= l) {
        intmid = l + (r - l) / 2;

        if(arr[mid] == x)
            returnmid;
        if(arr[mid] > x)
            returnbinarySearch(arr, l, mid - 1, x);
        returnbinarySearch(arr, mid + 1, r, x);
    }
```

(b) | 5

A graph is a non-linear data structure, which consists of vertices(or nodes) connected by edges(or arcs) where edges may be directed or undirected.Some application are

1.In **Computer science** graphs are used to represent the flow of computation.
2.Google maps uses graphs for building transportation systems, where intersection of two(or more) roads are considered to be a vertex and the road connecting two vertices is considered to be an edge, thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.

	3.In **Operating System**, we come across the Resource Allocation Graph where each process and resources are considered to be vertices. Edges are drawn from resources to the allocated process, or from requesting process to the requested resource. If this leads to any formation of a cycle then a deadlock will occur.			