



**OCTOBER 2020: IN SEMESTER ASSESSMENT B Tech III SEMESTER
TEST – 1**

UE19CS202 – DATA STRUCTURES AND ITS APPLICATIONS

Time: 2 Hrs

Answer All Questions

Max Marks: 60

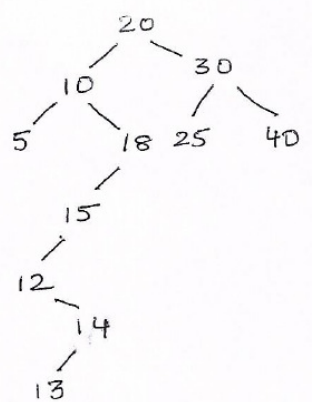
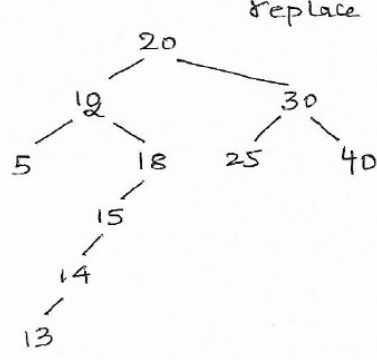
Scheme and Solution

1.	a)	<p>What is a Linked List? Give two reasons where in the Array is preferred over a Linked List</p> <p>A Linked List is linear data structure in which each element is not stored at contiguous locations. The elements are linked using their addresses</p> <p>Two Reasons over array being preferred over linked list</p> <p>Random access is faster in array than in linked list.</p> <p>Less space taken in array since no pointers.</p> <p>2 marks for definition of linked list, 1 mark each for reasons of array being preferred over linked list</p>	4
	b)	<p>Write a function is_sorted to check if a singly linked list is sorted in the ascending order or not. Return 1 if it is sorted else return 0. The address of the first node is passed to the function.</p> <pre>Issorted(struct node *p) { while(p->next!=NULL) – 1 mark { If (p->data > p->next->data) ---- 2 marks return 0 p=p->next; } return 1; -- 1 mark }</pre>	4
	c)	<p>What does the following code do?</p> <pre>sample(struct node *p) //pointer to the first node { If(p==NULL) return 0; return(p->data+ sample(p->next)) }</pre> <p>Returns the sum of all the nodes in the list - 2 marks</p>	2
2.	a)	Given a pointer to the first node of a doubly linked list, write a function to insert a	4

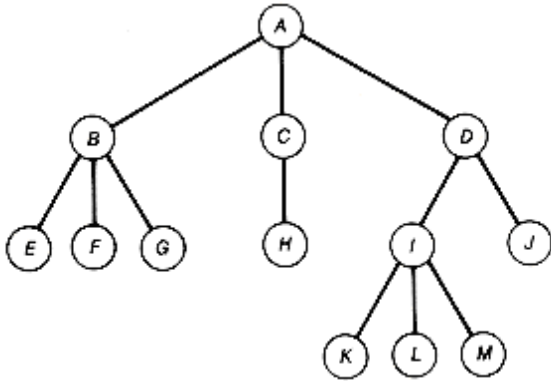
	<p>node with a data x after a node with data y. The function should take care of all the boundary conditions.</p> <pre> insert_after(struct node *p, int x, int y) { temp=malloc(sizeof(struct node)); 1 mark temp->data=x; temp->next=temp->prev=NULL; while(p->next!=NULL)&&(p->data!=y) --- 1 mark p=p->next; If(p->data==y) { If(p->next==NULL) // x is the last node - 1 mark { p->next = temp; temp->prev=p; } } else //somewhere in middle 1 mark { temp->next = p->next; p->next->prev=temp; p->next=temp; temp->prev=p; } } </pre>	
b)	<p>Given the address of the first node of a circularly doubly linked list, write a function to delete the last element.</p> <pre> delete_last(struct node **p) { struct node *t; t=*p q=*p while(q->next!=t) -- 1 mark q=q->next; if(q->next ==t)//only one node *p=NULL else ----- 3 mark { q->next->prev=q->prev; q->prev->next=q->next; } free (q); } </pre>	4

	c)	<p>How is the polynomial of a single variable represented as doubly linked list? Write the structure of the node.</p> <p>Each term of a polynomial is represented as a node defined below ----2 marks</p> <p>structure of a node</p> <pre>struct node { int coeff; int power_x; struct node *next, *prev }</pre>	2
3.	a)	<p>Using push, pop and is_empty functions , write a function to set i to the bottom element of the stack, leaving the stack unchanged (Hint : Use another auxiliary stack)</p> <p>For eg if the stack s1 contains {1, 2, 3 ,4} where 1 being the top of the stack , the function should set i=4, and the stack should remain unchanged.</p> <p>Do not write push, pop and isempty functions</p> <pre>set(int * S1 , int *S2) // S1 is the input stack and S2 is the auxillary stack { while (!empty (s1)) --2 marks { h = pop (s1); push (s2, h); } i = h; while (!empty (s2)) -- 2 marks { h = pop (s2); push (s1, h); } }</pre>	4
	b)	<p>Convert the following infix expressions to post fix and prefix</p> <p>A + (((B - C) * (D - E) + F) / G) \$ (H - J)</p> <p>Note : \$ is the exponent operator</p> <p>Postfix = ABC-DE-*F+G/HJ-\$+ 2 marks</p> <p>Prefix = +A\$/*-BC-DEFG-HJ 2 marks</p>	4
	c)	<p>Given a function that checks if an expression is balanced with respect to the parenthesis, Give an example where in the function incorrectly prints balanced. (push, pop and isempty are functions of the stack)</p> <pre>void balance_parentheses(char *exp) { while (exp[i]!='\0') { ch=exp[i]; if(ch== '(') // left paren push(ch) // push into the stack</pre>	2

		<pre> else if (ch== ')' && !isempty()) // right paren pop(); // pop from the stack else { printf("unbalanced parentheses") exit(); } print "balanced parentheses" } } </pre> <p>For the Input : ((())</p>	
4.	a)	<p>Write a function called AlternateSplit to split an integer queue into two queues which contain alternate elements of the original queue. Use only queue functions Insert, remove and isempty .For eg. if original queue Q is (1,2,3,4,5) then Q1 is (1,3,5) and Q2 is (2,4). You do not have to write insert, remove and isempty functions.</p> <pre> AlternateSplit(int *Q, int *Q1, int *Q2); Void AlternateSplit(int *Q, int *Q1, int*Q2) { While (!isEmpty(Q)) 1 mark { insert (Q1, remove(Q)); 1 mark If (!isEmpty(Q)) 1 mark insert (Q2,remove(Q)); 1 mark } } </pre>	4
	b)	<p>Write a function rem_left (removes the left element) function of a double ended queue implemented as a doubly linked list. (front and rears are pointers to the first and last node of the queue)</p> <pre> int rem_left(struct dequeue *p) { struct node *q; int x; if(p->front == NULL)//checking if list is empty 1 mark return -1; q=p->front; x=q->data; //is this the only node ? 1 mark if(p->front==p->rear) p->front=p->rear =NULL; else ----- 2 mark { p->front=q->next; } } </pre>	4

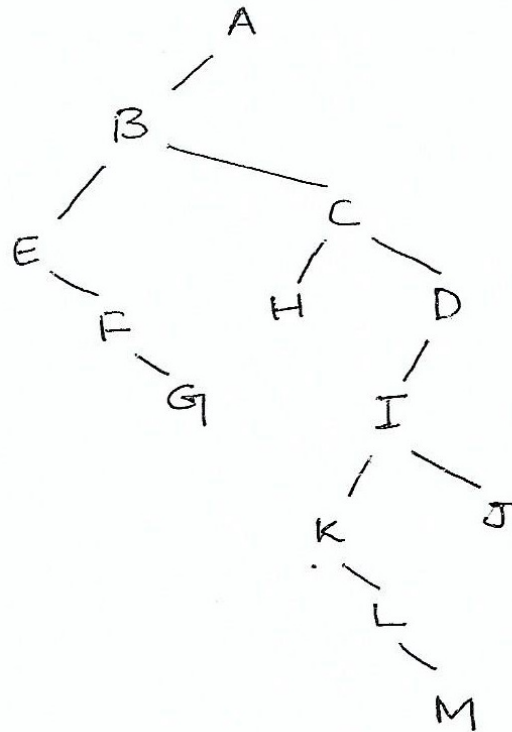
		<pre> p->front->prev=NULL; } free(q); return x; } </pre>	
	c)	<p>Write a recursive function to find the average of the elements in an array</p> <pre> average(int * a, n, int i) //a is pointer to an array, i is the index of the element, n is the number of elements { int s=0; if(i==n-1) return (a[i]/n) 1 mark else s=a[i]/n + average(a,n,i+1) 1 mark return s } </pre>	2
5.	a)	<p>Create a binary search tree for the following numbers. 20,30,10,18, 25,40,15,12,14,13,5 Show the tree after deletion of 10.</p>  <p>After deletion of 10</p>  <p>replace 10 with its inorder successor ie 12 it can also be replaced with its predecessor ie 5</p>	4
		<p>2 mark for tree creation and 2 mark for deletion of a node</p>	

	<p>b) The height of the tree is the path from the root to the deepest leaf node. Write a function to find the height of a binary tree implemented by an array</p> <pre> int height(int *t, int i) { int l,r; if(t[i]==-1) 1 mark return -1; if((t[2*i]==-1) && (t[2*i+1]==-1)) return 0; l=height(t,2*i); 2 marks r=height(t,2*i+1); if(l>r) return(l+1); 1 mark return(r+1); } </pre>	4
	<p>c) Implement a recursive function to find the sum of elements of a binary tree implemented by dynamic allocation.</p> <pre> int sum(struct tnode *t) { if(t==NULL) return 0; int l=sum(t->left); 1 mark int r=sum(t->right); return (l+r+t->data); 1 mark } </pre>	2

6	<p>a) Convert the following 3-ary tree to a binary tree</p>  <pre> graph TD A((A)) --- B((B)) A --- C((C)) A --- D((D)) B --- E((E)) B --- F((F)) B --- G((G)) C --- H((H)) D --- I((I)) D --- J((J)) I --- K((K)) I --- L((L)) I --- M((M)) </pre> <p>Write the steps for the inorder traversal of an n-ary tree, and show the corresponding inorder traversal of the above 3-ary tree</p> <p>Steps for inorder traversal of the n-ary tree</p> <ol style="list-style-type: none"> 1. Traverse in inorder the forest formed by the subtrees of the first tree, if any 2. Visit the root of the first tree in the forest 3. Traverse in inorder the forest formed by the remaining trees in the forest, if any 	4
---	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---

Inorder traversal of the above tree : E, F, G, B, H, C, K, L, M, I, J, D, A

Binary tree

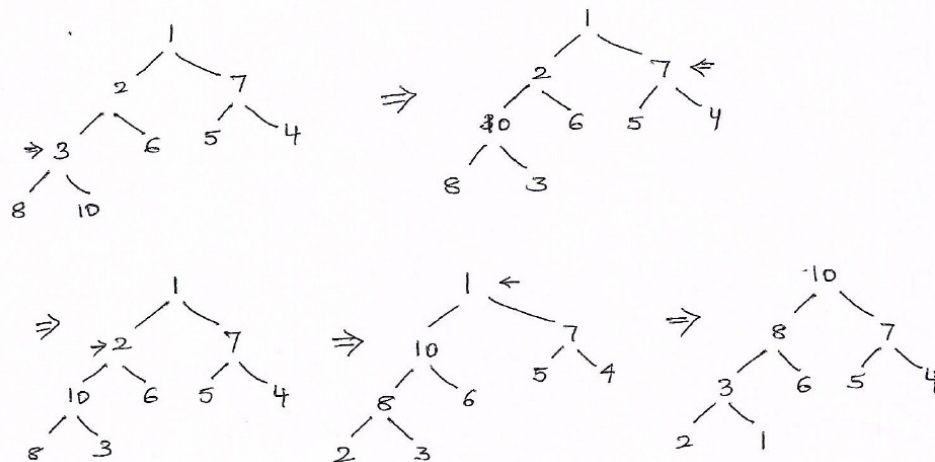


2 mark for binary tree conversion

1 mark each for steps for inorder traversal and printing the inorder traversal

- b) Create a maximum heap using bottom up approach for the following set of elements 1,2,7,3,6,5,4,8,10

4



2 marks for the conversion of root to heap, remaining marks for the other subtree conversions.

- c) Write a function to return the pointer to inorder successor of a node of an

2

		<p>inthreaded binary tree.</p> <pre>struct tnode *inordersuccessor(struct tnode *t) { if(t->rthread==true) 1 mark return(t->right); t=t->right; while(t->lthread==false) 1 mark t=t->left; return t; }</pre>	
--	--	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--