

**B.TECH.**  
**THIRD SEMESTER THEORY EXAMINATION 2010-11**

**ECS-302.**

**DATA STRUCTURES USING C**

**Time: 3 Hours**

**Total Marks: 100**

**Note:** (i) Attempt all questions.

1. Attempt any four parts of the following:

$$(5 \times 4 = 20)$$

- (i) Determine the formula to find the address location of an element in three dimensions array, suppose each element takes four bytes of space and elements are stored in column major order.

Ans. Let  $A[i][j][k]$  be an array of three dimension then the address can be calculated in column-major order by the following formula:

$$\text{LOC } A[i][j][k] = \text{Base}[A] + [(E_3 L_2 + E_2) L_1 + E_1] \times 4$$

Where  $L_1, L_2$  and  $L_3$  are the length of the three dimension array and  $E_1, E_2$  and  $E_3$  are the effective indices of respective subscripts.

- (ii) Write and explain the algorithm to find the 7th smallest element in an array.

Ans. To find 7th smallest element in array, first of all you have to sort the array elements using any sorted algorithm then you find the 7th element in array.

Findelement (A, pos)

```
{  n = sizeof [A]
  post = 7
  for j = 2 to n
  {  key = A[j]
    i = j - 1
    while (i > 1) and (A[i] > key)
    {
      A[i + 1] = A[i]
```

```
      i = i - 1
    }
    A[i + 1] = key
  }
  print A[pos]
}
```

- (iii) Write a program using C to create an array of 10 elements. Take 10 elements in the array pass them to a function which prints the elements in reverse order.

Ans. #include<stdio.h>

```
main()
{
  int a[10], i;
  printf("In energy array elementing in");
  for (i = 0, i < 10; i++)
    scanf("%d", &a[i]);
  reverse function (a);
  getch( );
}

reverse function (int a[ ])
{
  int i, b[10], j=0.
  for (i=g; i >=0, i--)
  {
    b[j]=a[i];
    j=j + 1'
  }
}
```

```

for (i=0; i<10; i++)
a[i]=b[i];
for (i=0; i<10; i++)
print f("%d", a[i];
}

```

(iv) Write an algorithm to find the location of an element in the given linked list. Is the binary search will be suitable for this search? Explain the reason.

Ans. Algorithm to find the location of element in a given linked list i.e., to search an element

Search (head, key)

```

{
    -int i = 0
    temp = head
    while (temp ≠ NULL)
    {
        if (temp → info = key)
        {
            print "Element found at location" exit (0),
            else
            {
                temp = temp → info;
                i = i + 1
            }
        }
    }
    print "Element not found"
}

```

(v) How a polynomial equation can be represented through link list? Explain the method to add two given polynomial equations using link list.

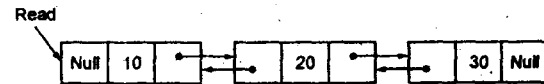
Ans. Please See Q. 2(c) (i), 2008-09

(vi) Define the double link list. Write an algorithm to delete an element from the existing link list.

Ans. A doubly linked list is a two way linked list in which each node contains three fields:

- An address of previous node, used to define the links between its predecessor.
- A data element, used to store information value
- An address of next node, used to define the links between its successor

It is graphically represented as follow:



Algorithm to delete an element

delete-node (head, element)

```

{
    temp = head
    while (temp → info ≠ element)
    {
        temp = temp → next
    }
    temp → next → prev = temp → prev;
    temp → prev → next = temp → next;
    free (temp)
}

```

2. Attempt any four parts: (5 × 4 = 20)

(a) Write an algorithm to evaluate an expression given in postfix notation.

Ans. Please See Q. 1(e), 2006-07.

(b) Define the recursion. Write a recursive and non recursive program to calculate the factorial of the given number.

Ans. Recursive function: A function which calls itself is called a recursive function.

A recursive program of factorial no. is

≠ include < statio. h)

main

```

{
    int num, fact;
    scanf ("%d", num);
    fact = factorial (num);
}

```

```

    printf ("Factorial = %d", fact);
    getch( );
}

int factorial (int x)
{
    if (x = 1)
        return 1,
    else
        return x * factorial (x - 1);
}

A non-recursive program to of factorial no=include
<stdio.h>

main ( )
{
    int num, fact = 1, i;
    scanf ("%d", &num);
    for(i = 1; i <= num; i++)
        fact = fact * i;
    printf ("factorial = %d", fact);
}

```

(c) What do you mean by priority queue? Explain the types to maintain the priority queue.

**Ans. Priority queue:** A priority queue is a collection of elements in which each element has been assigned a priority such that the order in which elements are deleted and processed comes from the following rules:

1. An element of higher priority is processed before any elements of lower priority.
2. If the two elements have the same priority then the elements are processed according to the order in which they were added to the queue.

The types to maintain the priority queue are:

Max priority queue

Min priority queue

Max-priority queue supports the following operation:

insert (s, x)  
Maximum (s)  
Extract max(s)  
Increase key(s, x, k)

Min-priority queue supports the following operation:

Insert (s, x)  
Minimum (s)  
Extract-min (s)  
Decrease key (s, x, k)

(d) What is a circular queue ? Write a C function to insert an item in the circular queue.

**Ans.** Please See the solution of 2(b), 2008–09

(e) What is the Tower of Hanoi problem ? Explain the solutions of the Tower of Hanoi problem where the number of disks are 4 and number of pegs are 3.

**Ans. Tower of Hanoi problem:** The tower of hanoi problem is recursive problem which is based on three pegs and a set of disk of sizes. Suppose three pegs labeled *P*, *Q* and *R*, and on peg *P* there are finite number of *n* disk is increasing order of size i.e., biggest at the bottom and smallest at the top. The problem is to move disks from peg *P* to peg *R* using peg *Q* whenever it is required. So the rules which are required before moving a disk are:

- (i) Only one disk may be moved at a time.
- (ii) Only the top disk on any peg may be moved to any other peg.
- (iii) A larger disk can not be placed on a smaller one.

Lets us consider the three pegs, *P*, *Q*, and *R* and number of disk is 4.

1. Move from *P* to *Q*
2. Move from *P* to *Q*
3. Move from *Q* to *R*
4. Move from *P* to *Q*
5. Move from *R* to *P*

6. Move from  $R$  to  $Q$
7. Move from  $P$  to  $Q$
8. Move from  $P$  to  $R$
9. Move from  $Q$  to  $R$
10. Move from  $Q$  to  $P$
11. Move from  $R$  to  $P$
12. Move from  $Q$  to  $R$
13. Move from  $P$  to  $Q$
14. Move from  $P$  to  $R$
15. Move from  $Q$  to  $R$

(f) Convert the following infix expression into postfix expression:

$$A*(B+D)/E-F*(G+H/K)$$

Ans.  $A*(B+D)/E-F*(G+H/K)$

$$A*(BD+)/E-F*(G+H/K)$$

$$A*(BD+)/E-F*(G+K/H)$$

$$A*(BD+)/E-F*(GHK/+)$$

$$A*(BD+)/E-F*(GHK/)$$

$$(ABD+*)/E-F*(GHK/+)$$

$$(ABD+*)/E-F*(GHK/+)$$

$$(ABD+*E)-F*(GHK/+)$$

$$(ABD+*E)-(FGHK/+*)$$

$$ABD+*E/FGHI/+*-$$

3. Attempt any two parts of the following:

$$(10 \times 2 = 20)$$

- (a) (i) Determine in the terms of level 1 the maximum number of possible nodes at level 1 in the binary tree and also determine the maximum total number of possible nodes in a binary tree if the root node is at level 1.

Ans. The term level 1 determines the possible number of node of level 1.

The maximum possible number of nodes of level 1 are

If level 1 is a root then the node is one.

If level 1 is not a root then the nodes are two.

The total possible nodes at level 1 is exactly one if level 1 is at root.

(ii) How a node can be deleted from the binary search tree? Explain the methods.

Ans. In binary search tree, deletion depends upon the number of children of the deleted node and discussed below." If node  $N$  contains a key value and we want to delete  $N$  then:

1. If node  $N$  has no child then node  $N$  is deleted from BST by replacing the location of node  $N$  in the parent node by the NULL pointer.
2. If node  $N$  has one child then node  $N$  is deleted from BST by replacing the location of node  $N$  in parent node by location of the child of node  $N$ .
3. If node  $N$  has two children then node  $N$  is deleted from BST by first deleting in order successor of node  $N$  from BST using step 1 or step 2 and then replacing node  $N$  by inorder successor of node  $N$ .

(b) Write the function in C to insert and delete a node in an existing binary search tree.

Ans. Please See Q. 3(b) 2009-10.

(c) The preorder and inorder traversal of binary tree is given below, construct the tree—

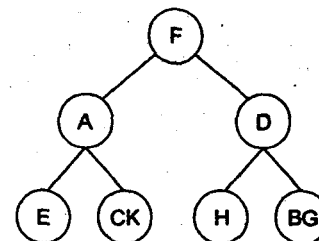
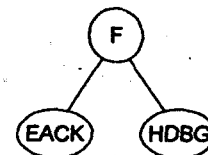
Preorder: FAEKCDHGB

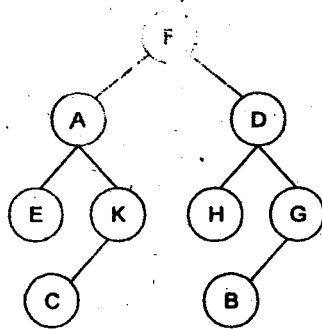
Inorder: EACKFHDBG

Ans. The given sequence are:

Preorder: FAEKCDHGB

Inorder: EACKFHDBG





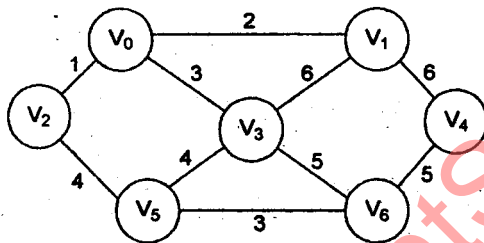
4. Attempt any two parts of the following:

(10 × 2 = 20)

- (a) Write and explain the breadth first search and depth first graph traversal algorithm. What are their complexities?

Ans. Please See Q. 5(a), 2005-06

- (b) Define the spanning tree. Write the Prim's algorithm to find the minimum cost spanning tree of the following:



Ans. A spanning tree of a graph is just a subgraph that contains all the vertices without any cycle.

Prim's algorithm:

```

MST - PRIM (G, W, S)
{
  for each u ∈ V(G)
  {
    Key[u] = ∞
    π[u] = NIL
  }
  Key[s] = 0
  Q = V[G]

```

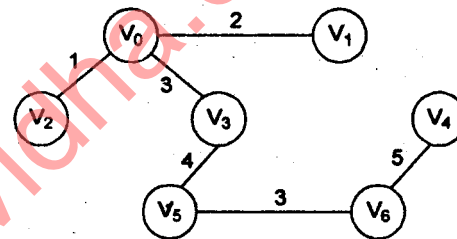
while (Q ≠ NULL)

```

{
  u = Extract = min(Q)
  for each v ∈ adj[u]
  {
    if v ∈ Q and w(u, v) < key[v]
    π[v] = u
    key[v] = w(u, v)
  }
}

```

Minimum cost spanning tree for a given graph is

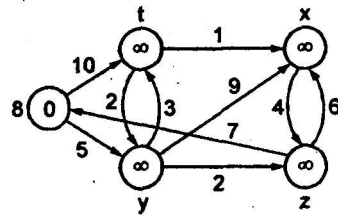


- (c) Describe the Dijkstra's algorithm for finding shortest path with help of suitable example.

Ans. A min-priority queue of vertices, keyed by the values of vertices. (Initially it will contain source vertices 's' also and it is the first node).

1. Initialize-single-source (G,S).
2.  $S \leftarrow \phi$
3.  $Q \leftarrow V[G]$
4. while  $Q \neq \phi$
5. do  $\leftarrow \text{EXTRACT. MIN}(Q)$
6.  $S \leftarrow S \cup \{U\}$
7. for each verted  $v \in \text{Ad}[u]$
8. do RELAX ( $u, v, w$ )

**Example 1.** Consider the following graph and using Dijkstra Algorithm find the shortest path.

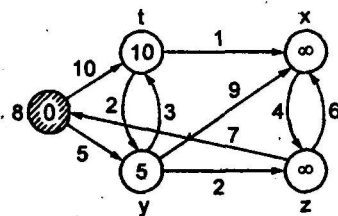


**Step 1:**  $S = \{s\}$

$Q = \{s, t, x, y, z\}$

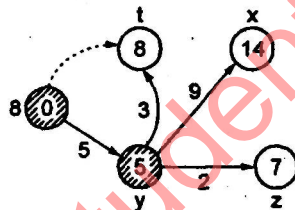
$u = s$

$S = \{s\}$



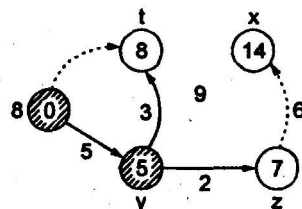
**Step 2:**  $u = y$

$S = \{s, y\}$



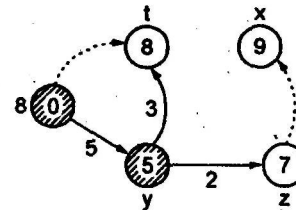
**Step 3:**  $u = z$

$S = \{s, y, z\}$



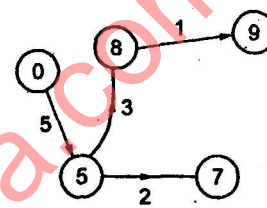
**Step 4:**  $u = t$

$S = \{s, y, z, t\}$



**Step 5:**  $u = x$

$S = \{s, y, z, t, x\}$



So, 'S' represents the shortest path from a single source  $s$ .

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

**(10 × 2 = 20)**

**(a) Write the algorithm for bubble sort and insertion sort. Explain their best case and worst case complexities.**

**Ans. Bubble sort (A, n)**

```
{
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (A[j] > A[j + 1])
            {
                temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
            }
        }
    }
}
```

```

}
}

```

Time complexity of bubble sort is always  $O(n^2)$ .  
i.e., in best and worst both cases is  $O(n^2)$  Insertion sort.

```

insertion sort (A, n)
{
    for j = 1; j < n; j ++
    {
        key = A[j]
        i = j - 1;
        while ((i > 1) and (A[i] > key))
        {
            A[i + 1] = A[i]
            i = i - 1
        }
        A[i + 1] = key
    }
}

```

Time complexity of insertion sort:

**Best case:** If the inner loop will not execute i.e.,  
th elements are already in sorted order then the time  
complexity is  $O(n)$ .

**Worst case:** If the inner loop will execute maximum  
time then the time complexity is  $O(n^2)$ .

(b) Write the algorithm for the merge sort.  
Explain its complexities. Sort the following  
using merge sort method:

75, 10, 20, 70, 80, 90, 100, 40, 30, 50.

**Ans. Merge sort:** The sorting algorithm  
Mergesort produces a sorted sequence by sorting  
its two halves and merging them. With a time  
complexity of  $O(n \log(n))$  Merge sort is optimal.

**Algorithm:**

```

void merge sort (int lo, int hi)
{
    if (lo < hi)
    {
        int m = (lo + hi) / 2;

```

```

mergesort(lo, m);
mergesort(m + 1, hi);
merge(lo, m, hi);
}
}

void merge(int lo, int m, int hi)
{
    int i, j, k;
    for (i = lo; i < hi; i++)
        b[i] = a[i];
    i = lo; j = m + 1; k = lo;
    while (i <= m and j <= hi)
        if (b[i] <= b[j])
            a[k++] = b[i++];
        else
            a[k++] = b[j++];
    while (i <= m)
        a[k++] = b[i++];
}

```

The given elements are

75, 10, 20, 70, 80, 90, 100, 40, 30, 50

**Pass 1.** Merge each pair of elements to obtain  
the following list of sorted pairs:

[10, 75], [20, 70], [80, 90], [40, 100], [30, 50]

**Pass 2.** Merge each pair of pairs to obtain the  
list of sorted elements:

[10, 20, 70, 75], [40, 80, 90, 100] [30, 50]

**Pass 3.** Again merge each pair of pairs:

[10, 20, 40, 70, 75, 80, 90, 100] [30, 50]

**Pass 4.** Again merge each pair of pairs:

[10, 20, 30, 40, 50, 70, 75, 80, 90, 100]

(c) Explain the following:

- (i) Heap Sort
- (ii) Radix Sort.

**Ans. Heap Sort:** The heap data structure is an  
array object which can be easily visualized as a complete

binary tree. There is a one to one correspondence between elements of the array and nodes of the tree. The tree is completely filled on all levels except possibly the lowest, which is filled from the left upto a point. All nodes of heap also satisfy the relation that the key value at each node is at least as large as the value at its children.

**Step I:** The user inputs the size of the heap (within a specified limit). The program generates a corresponding binary tree with nodes having randomly generated key values.

**Step II: Build Heap Operation:** Let  $n$  be the number of nodes in the tree and  $i$  be the key of a tree. For this, the program uses operation Heapify. When Heapify is called both the left and right subtree of the  $i$  are Heaps. The function of Heapify is to let  $i$  settle down to a position (by swapping itself with the larger of its children, whenever the heap property is not satisfied) till the heap property is satisfied in the tree which was rooted at  $(i)$ .

**Step III: Remove maximum element:** The program removes the largest element of the heap (the root) by swapping it with the last element.

**Step IV:** The program executes Heapify (new root) so that the resulting tree satisfies the heap property.

**Step V:** Go to step III till heap is empty.

**(ii) Radix Sort:** It is also known as bucket sort. This method can be used to sort the names alphabetically or the numbers in ascending or descending order taking base or radix 26 in case of alphabets and 10 in case of decimal numbers.

To sort the decimal numbers, we need 10 buckets and these buckets are numbered 0 to 9. The decimal numbers are sorted first according to unit digit and then tens digit and then 100th digit and so on.

#### Algorithm

1. [Initialization]  
    large = largest element in the array  
    num = total number of digits in the largest element  
    digit = num  
    pass = 1
2. Repeat steps 3 to 8 while pass  $\leq$  num
3. Initialize bucket and set  $i = 0$
4. Repeat steps 5 to 7 while  $i \leq n - 1$
5. Set  $k = \text{pass} - 1$  [position of number  $A[i]$ ]
6. Put the number  $A[i]$  into bucket  $k$
7. Set  $i = i + 1$
8. Set pass = pass + 1
9. Print the numbers from the bucket
10. Exit