B.Tech. (CSE/IT) – 3rd Sem
DATA STRUCTURES Paper Key
ACCS - 16301

Section - A

Q1)

i) Imagine a 3D array A(3:6, 2:8, 4: 8). Calculate total no. of elements in this array.

Answer : 140 elements

ii) Convert A* (B + D * G) – E * F into its postfix expression.

Answer : A B D G * + * EF * -

iii) Write code for syntax of creating a node in a Linked list.

Answer :

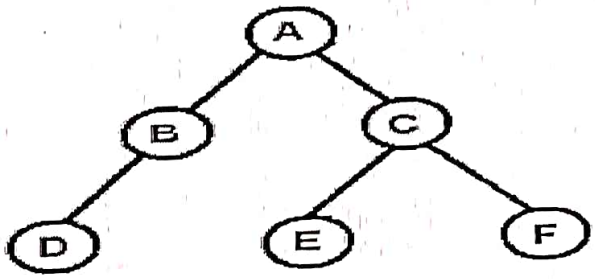
```
struct *node
{
int data;
node *next
}
```

iv) Which sorting algorithm uses Divide & Conquer methodology ? Explain how.

Answer :

- Quick Sort
- Merge Sort

v) Write In-order traversal expression for tree

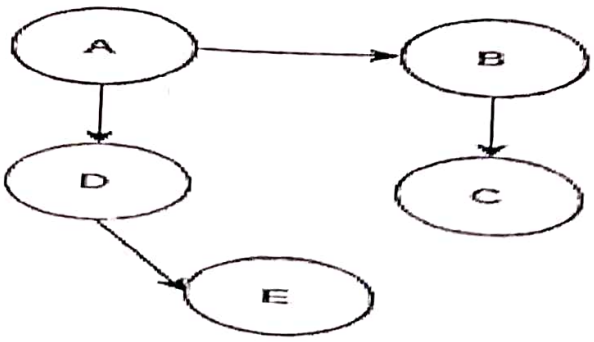


Answer : D B A E C F

vi) Explain term Big O Notation with taking bubble sort as example.

Answer : $O(n^2)$

vii) Explain Indegree and Outdegree for the graph given below



Answer : A B C D E

INdegree : 0 1 1 1 1

OUTdegree : 2 1 1 1 1

viii) Explain any 5 applications of Stack data Structures.

Answer :

1. Quicksort
2. Infix to postfix conversion
3. Postfix evaluation
4. Tree Traversal
5. Graph Traversal

ix) Find the Big O Notation for the below given code

```
for(i=1 ; i<=n ; i++)
{
for(j=1 ; j<=n ; j++)
{
cout<<i<<j;
} }
```

Answer : $O(n^2)$

x) Explain complexity of Merge sort in best, average case.

Answer: In sorting n objects, merge sort has an average and worst-case performance of O(n log n). ... In terms of moves, merge sort's worst case complexity is O(n log n)—the same complexity as quicksort's best case, and merge sort's best case takes about half as many iterations as the worst case

Section – B

(4 × 5 = 20)

Q2) Sort 33,44,22,77,66,11,55 using insertion sort. Also Write algorithm for it.

Answer :

```
function insertionSort(array A)
  for i from 1 to length[A]-1 do
    value := A[i]
```

```
    j := i-1
    while j >= 0 and A[j] > value do
      A[j+1] := A[j]
      j := j-1
    done
    A[j+1] = value
  done
```

Q3) Consider a 3D array A(2:8,3:6,4:8). Base(A)=200, w=4, Calculate Loc(A[6,4,6]).

Answer :

Equations :

Let C be an n-dimensional array.

The length L(*i*) of dimension *i* of C can be calculated by,

L(i)=upper bound - lower bound +1

For a given subscript K(*i*) , the effective index E(*i*) of L(*i*) can be calculated from,

E(i)=K(i) - lower bound

Then address C[K(1),K(2),...,K(n) of element of C is,

Base (C) +w[(((.....(E(n)L(n-1)+E(n-1))L(n-2))+....+E(3))L(2)+E(2))L(1)+E(1)] , when C is stored in column major, and

Base (C)+w[(...((E(1)L(2)+E(2))L(3)+E(3))L(4)+...+E(n-1))L(n)+E(n)] , when C is stored in row major order.

Base (C) denotes the address of 1st element of C and w denotes the number of words per memory location.

Q4) Explain various hash methods used in hashing with example for each.

Answer :

Types of hash function

There are various types of hash function which are used to place the data in a hash table,

1. Division method

In this the hash function is dependent upon the remainder of a division. For example:-if the record 52,68,99,84 is to be placed in a hash table and let us take the table size is 10.

Then:

h(key)=record% table size.

2=52%10
8=68%10
9=99%10
4=84%10

2. Mid square method

In this method firstly key is squared and then mid part of the result is taken as the index. For example: c... if we want to place a record of 3101 and the size of table is 1000. So 3101*3101=9616201 i.e. h (3101) = ... (middle 3 digit)

3. Digit folding method

In this method the key is divided into separate parts and by using some simple operations these parts are ... to produce a hash key. For example: consider a record of 12465512 then it will be divided into parts i.e. ... 12. After dividing the parts combine these parts by adding it.

H(key)=124+655+12
=791

Q5) Explain algorithm for Quick sort for sorting 35, 12, 08, 10, 66, 32, 46.

Answer :

Algorithm :

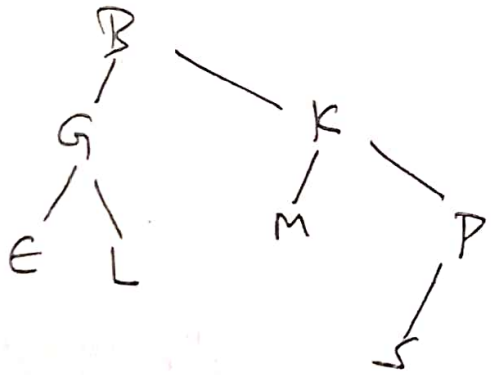
Step 1 – Choose the highest index value has pivot
Step 2 – Take two variables to point left and right of the list excluding pivot
Step 3 – left points to the low index
Step 4 – right points to the high
Step 5 – while value at left is less than pivot move right
Step 6 – while value at right is greater than pivot move left
Step 7 – if both step 5 and step 6 does not match swap left and right
Step 8 – if left ≥ right, the point where they met is new pivot

Q6) Make a binary tree for

Preorder : B G E L K M P S

Inorder : E G L B M K S P

Answer :



Section – C

Q7) Convert B + (D * G + H) – (W / P * Q) into its postfix using stack data structure. Also write the algorithm for the same.

Answer :

Output : BDG*H+-WP/Q*-

BDG*H+WPQ*/-+

Q8) Explain warshall's shortest path finding algorithm with example.

Answer:

Create a |V| x |V| matrix, M, that will describe the distances between vertices

For each cell (i, j) in M:

```
  if i == j:
    M[i][j] = 0
  if (i, j) is an edge in E:
    M[i][j] = weight(i, j)
  else:
    M[i][j] = infinity
for k from 1 to |V|:
  for i from 1 to |V|:
    for j from 1 to |V|:
      if M[i][j] > M[i][k] + M[k][j]:
        M[i][j] = M[i][k] + M[k][j]
```

IT= 1 fail.

Q9) Explain Breadth first search, depth first search algorithm with example for each.

Answer:

Depth First Search:

The general idea behind depth first traversal is that, starting from any random vertex, single path is traversed until a vertex is found whose all the neighbors are already been visited. The search then backtracks on the path until a vertex with unvisited adjacent vertices is found and then begin traversing a new path starting from that vertex, and so on. This process will continue until all the vertices of the graph are visited.

//Algorithm for DFS()

Step1. Initialize all the vertices to ready state (STATUS = 1)

Step2. Put the starting vertex into STACK and change its status to waiting (STATUS = 2)

Step 3: Repeat Step 4 and 5 until STACK is EMPTY

Step 4: POP the top vertex from STACK, Process the vertex, Change its status to processed state (STATUS = 3)

Step 5: PUSH all the neighbors in the ready state (STATUS = 1) to the STACK and change their status to waiting state (STATUS = 2)

Step 6: Exit.

Breadth First Search:

The general idea behind breadth first traversal is that, start at a random vertex. then visit all of its neighbors, the first vertex that we visit, say is at level '0' and the neighbors are at level '1'. After

visiting all the vertices at level '1' we then pick one of these vertexes at level '1' and visit all unvisited neighbors, we repeat this procedure for all other vertices at level '1'. Say neighbors level 1 are in level 2, now we will visit the neighbors of all the vertices at level 2, and this procedure will continue.

// algorithm for BFS()

Step1. Initialize all the vertices to ready state (STATUS = 1)

Step2. Put the starting vertex into QUEUE and change its status to waiting (STATUS = 2)

Step 3: Repeat Step 4 and 5 until QUEUE is EMPTY

Step 4: Remove the front vertex from QUEUE, Process the vertex, Change its status to process state (STATUS = 3)

Step 5: ADD all the neighbors in the ready state (STATUS = 1) to the RARE of the QUEUE and change their status to waiting state (STATUS = 2)

Step 6: Exit.