Sub:	**DATA STRUCTURES AND APPLICATIONS**					Sub Code:	**BCS304**
Date:		Duration:	**90 minutes**	Max Marks:	**50**	Sem/Sec: III A,B,C	

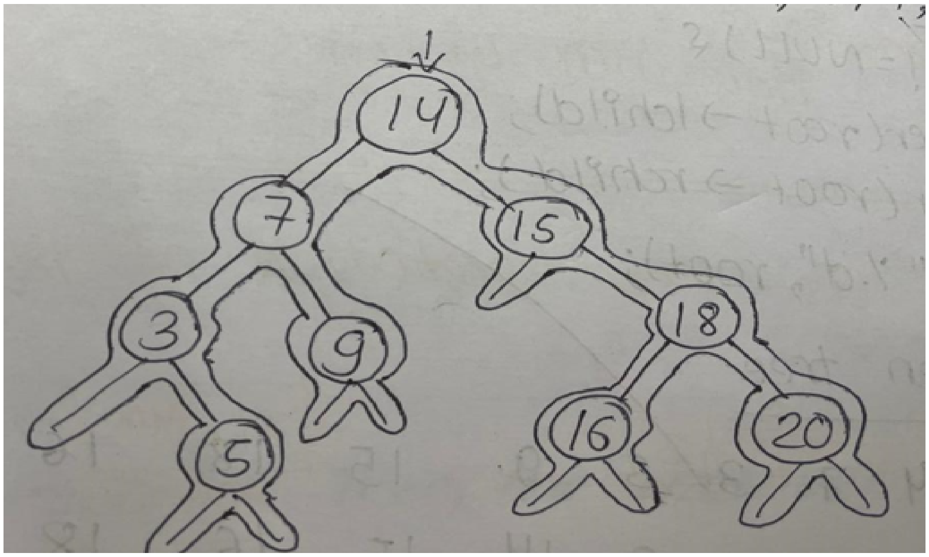
Scheme and Solutions

1 **a**

Construct a binary search tree for the given values 14, 15, 7, 9, 18, 3, 5, 16, 20. Write the C function for the inorder, preorder and postorder traversal and apply the same.

Answer:

Construction of BST-2M



C function (5 marks)

```
void inorderTraversal(struct Node* root) {

   // Empty Tree
   if (root == NULL)
       return;

   // Recur on the left subtree
   inorderTraversal(root->left);

   // Visit the current node
   printf("%d ", root->data);

   // Recur on the right subtree
   inorderTraversal(root->right);
}
void preorderTraversal(struct Node* root) {
   // Base case
   if (root == NULL)
      return;
   // Visit the current node
   printf("%d ", root->data);
   // Recur on the left subtree
   preorderTraversal(root->left);
   // Recur on the right subtree
   preorderTraversal(root->right);
}
// Function to perform postorder traversal
void postorderTraversal(struct Node* node) {
   // Base case
   if (node == NULL)
      return;
   // Recur on the left subtree
   postorderTraversal(node->left);
   // Recur on the right subtree
   postorderTraversal(node->right);
   // Visit the current node
   printf("%d ", node->data);
}
```

Traversals (3M)

Inorder: 3, 5, 7, 9, 14, 15, 16, 18, 20
Preorder: 14, 7, 5, 3, 9, 16, 15, 18, 20
Postorder: 3, 5, 9, 7, 15, 20, 18, 16, 14

2

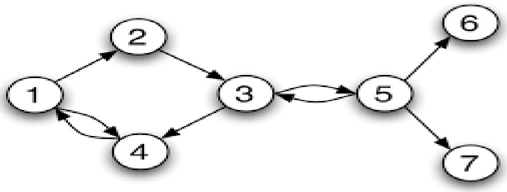
a Explain winner tree and looser tree with suitable examples.

Answer:

Winner Tree Explanation with example-**2.5M**

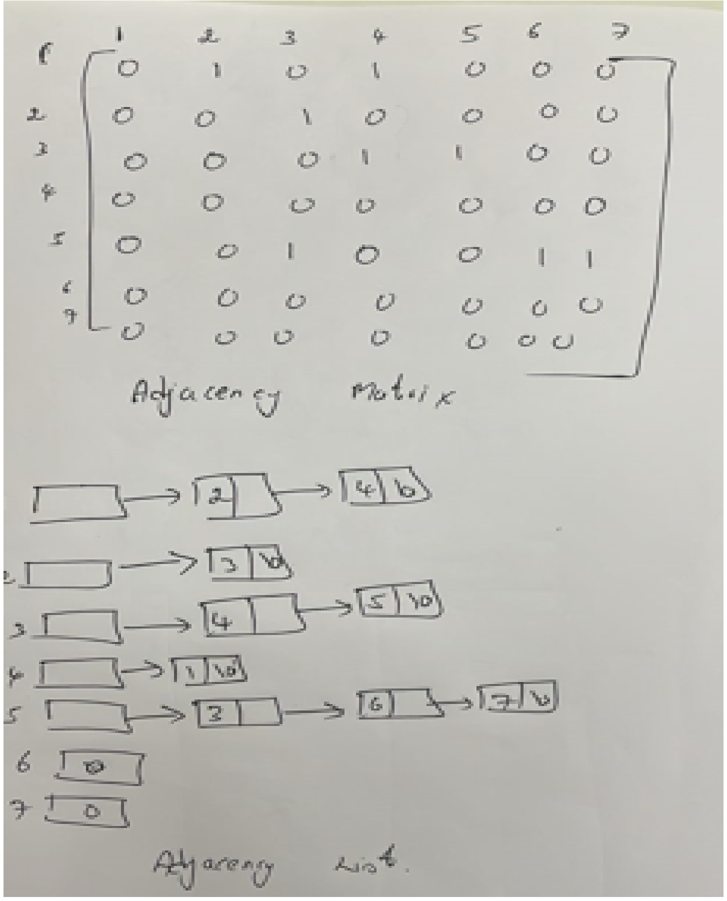
Looser Tree Explanation with example-**2.5M**

b Define a graph. Show the adjacency matrix and adjacency list for the following.



Answer:

Graph Defn: 1M
Representation as adjacency matrix 2M lists 2M



$$\begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Adjacency matrix

Adjacency List.

What is dynamic hashing? Explain the following techniques with examples:

i) Dynamic hashing using directories

ii) Directory less dynamic hashing.

Answer:

a

Dynamic hashing is a technique used in hash-based data structures to efficiently handle growing datasets. Unlike static hashing, where the number of buckets is fixed and may lead to excessive collisions or wasted space, dynamic hashing allows the number of buckets to grow or shrink dynamically based on the number of elements in the hash table. This ensures that the table remains efficient and scalable, without requiring a complete rehashing when the data set changes. **(1 M)**

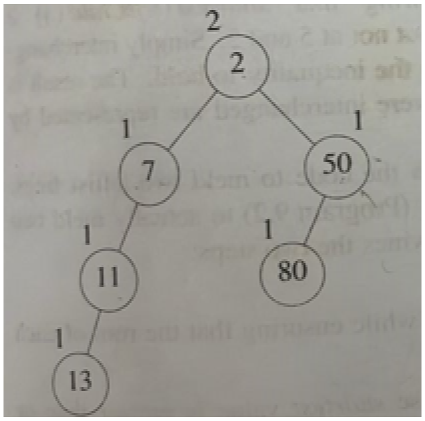
Dynamic hashing using directories with example (2M)

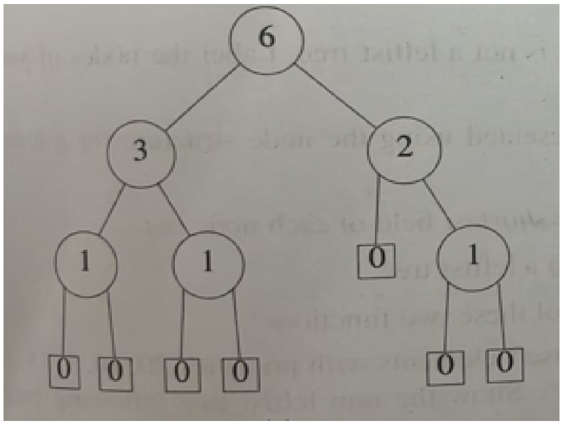
Directory less dynamic hashing. (2M)

b

Differentiate between height-biased and weight-biased leftist tree with examples.
Answer: (4 M + 1 M Example)

Aspect	Height-biased Leftist Tree	Weight-biased Leftist Tree
Balance Criterion	Based on the height of the left and right subtrees.	Based on the number of nodes in the left and right subtrees.
Null Path Length (NPL)	NPL of a node is the height of its right subtree.	NPL of a node is the number of nodes in its right subtree.
Subtree Priority	The left subtree is prioritized by height.	The left subtree is prioritized by size (number of nodes).
Use Case	More focused on maintaining the height balance.	More focused on maintaining a size balance between subtrees.
Tree Shape	Tends to be more height-balanced.	Tends to be more size-balanced.

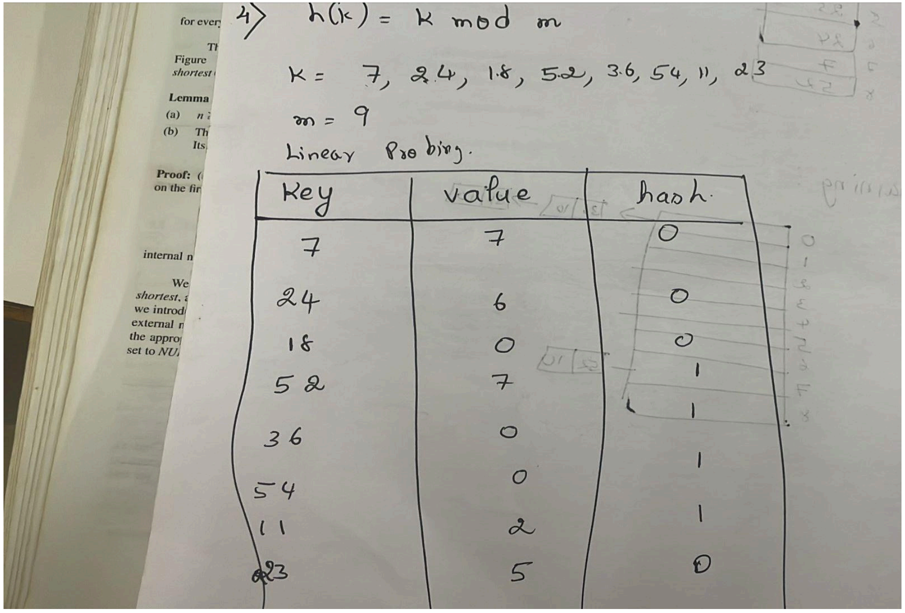




4 a

What is a collision? Explain the collision resolution techniques in detail considering 7, 24, 18, 52, 36, 54, 11, 23 with 9 memory locations. Use h(k) = k mod m.

Answer:
Collision and Collision resolution techniques (2M)
Solution (8 M)

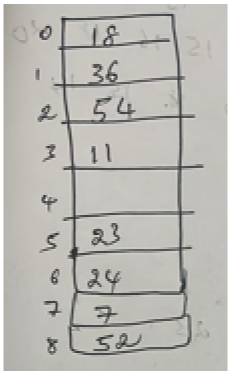
<table>
<tr><td></td><td></td><td>

4) h(k) = k mod m

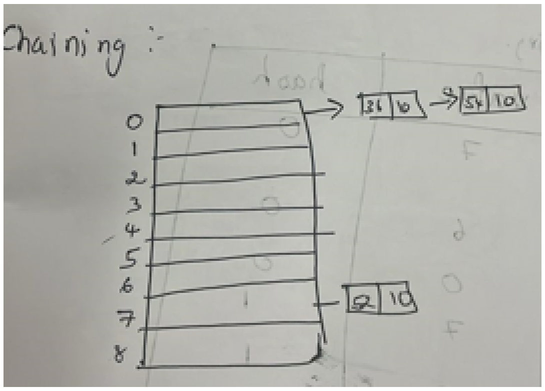
k = 7, 24, 18, 52, 36, 54, 11, 23

m = 9

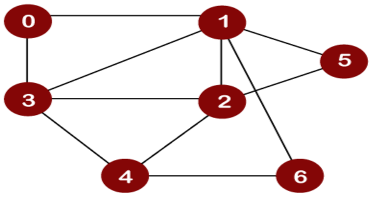
Linear Probing.

<table>
<tr><th>Key</th><th>value</th><th>hash</th></tr>
<tr><td>7</td><td>7</td><td>0</td></tr>
<tr><td>24</td><td>6</td><td>0</td></tr>
<tr><td>18</td><td>0</td><td>0</td></tr>
<tr><td>52</td><td>7</td><td>1</td></tr>
<tr><td>36</td><td>0</td><td>1</td></tr>
<tr><td>54</td><td>0</td><td>1</td></tr>
<tr><td>11</td><td>2</td><td>1</td></tr>
<tr><td>23</td><td>5</td><td>0</td></tr>
</table>


<table>
<tr><th></th><th></th></tr>
<tr><td>0</td><td>18</td></tr>
<tr><td>1</td><td>36</td></tr>
<tr><td>2</td><td>54</td></tr>
<tr><td>3</td><td>11</td></tr>
<tr><td>4</td><td></td></tr>
<tr><td>5</td><td>23</td></tr>
<tr><td>6</td><td>24</td></tr>
<tr><td>7</td><td>7</td></tr>
<tr><td>8</td><td>52</td></tr>
</table>


Chaining :-

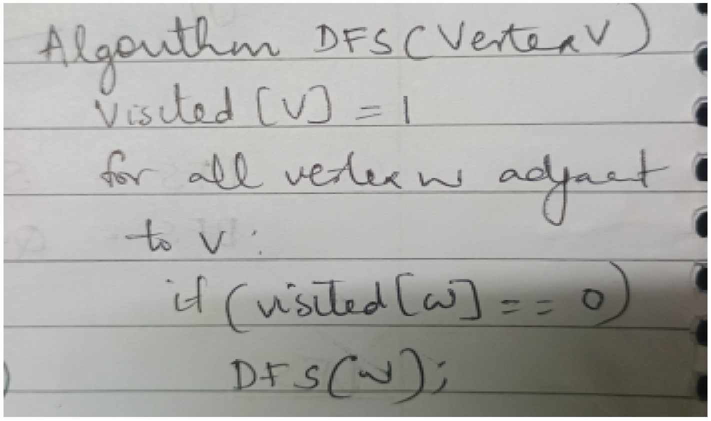
</td></tr>
<tr><td>5</td><td>a</td><td>
Explain BFS and DFS with its C-function. Apply BFS and DFS for the below graph considering 0 to be the root node.



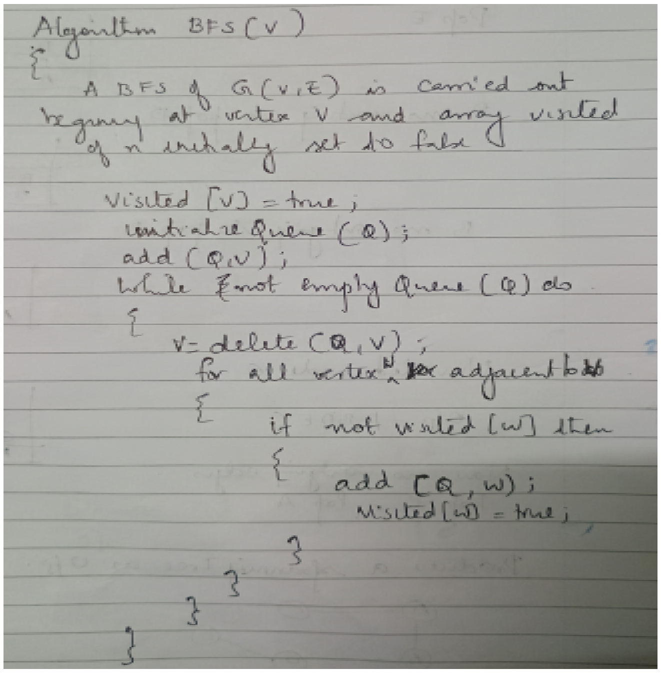
Answer:DFS (Depth First Search) BFS (Breadth First Search)

Algorithm for DFS or BFS 5M

Solving 5 M
</td></tr>
</table>



```
Algorithm DFS(Vertex V)
Visited [V] = 1
for all vertex w adjacent
 to V :
   if (visited[w] == 0)
      DFS(w);
```



```
Algorithm BFS(V)
{
   A BFS of G(V,E) is carried out
beginning at vertex V and array visited
of n initially set to false

   Visited [V] = true;
    initialize Queue (Q);
    add (Q,V);
   while not empty Queue (Q) do
    {
       V= delete (Q,V);
        for all vertex w adjacent to
        {
              if not visited [w] then
              {
                   add [Q, w);
                    visited[w] = true;
              }
        }
    }
}
```

BFS: 0 1 3 2 5 6 4

DFS: 0 1 5 2 3 4 6

6 a

What is the need for an optimal BST. Find the optimal BST for n=4,

Keys are 10,15,20, 25.

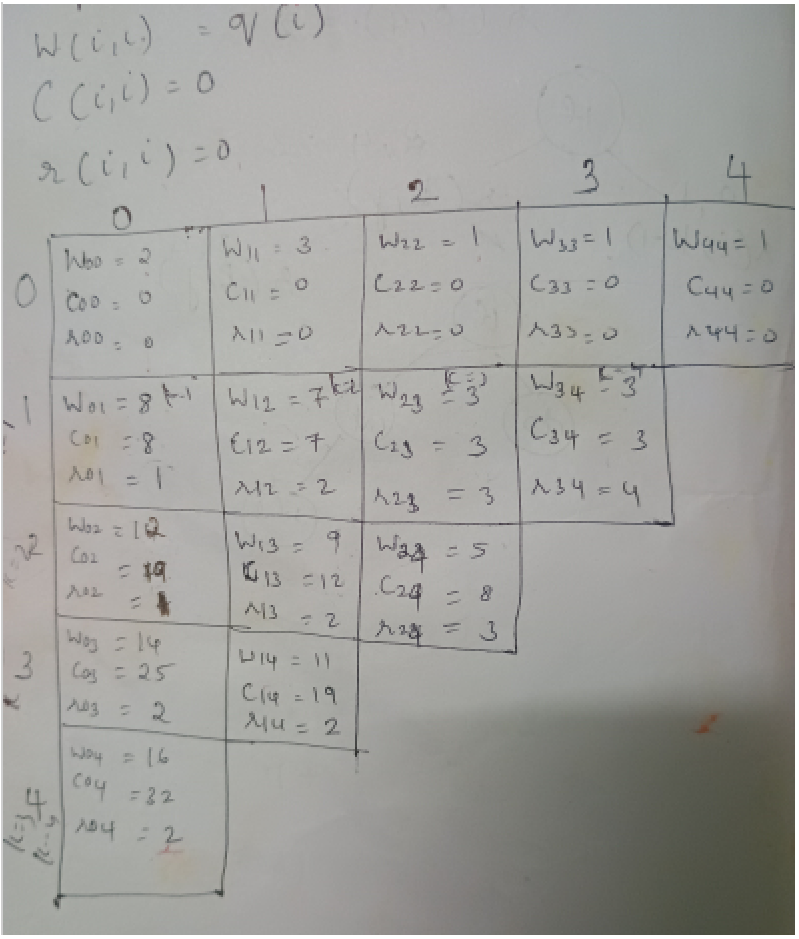
p1, p2, p3, p4 =3,3,1,1

q0, q1, q2, q3, q4 =2,3,1,1,1

Answer:

Need for BST-1M

Problem-4M



$W(i,i) = q(i)$

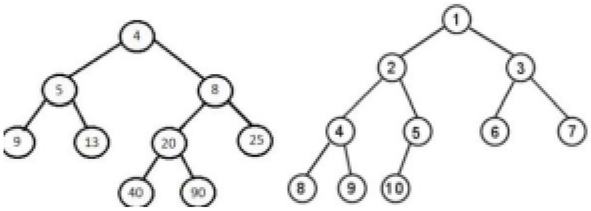
$C(i,i) = 0$

$r(i,i) = 0$

	0	1	2	3	4
0	$W_{00} = 2$ $C_{00} = 0$ $r_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $r_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $r_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $r_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $r_{44} = 0$
1	$W_{01} = 8$ $k=1$ $C_{01} = 8$ $r_{01} = 1$	$W_{12} = 7$ $k=2$ $C_{12} = 7$ $r_{12} = 2$	$W_{23} = 3$ $k=3$ $C_{23} = 3$ $r_{23} = 3$	$W_{34} = 3$ $k=4$ $C_{34} = 3$ $r_{34} = 4$	
2	$W_{02} = 12$ $C_{02} = 19$ $r_{02} =$ [illegible]	$W_{13} = 9$ $C_{13} = 12$ $r_{13} = 2$	$W_{24} = 5$ $C_{24} = 8$ $r_{24} = 3$		
3	$W_{03} = 14$ $C_{03} = 25$ $r_{03} = 2$	$W_{14} = 11$ $C_{14} = 19$ $r_{14} = 2$			
4	$W_{04} = 16$ $C_{04} = 32$ $r_{04} = 2$				

b

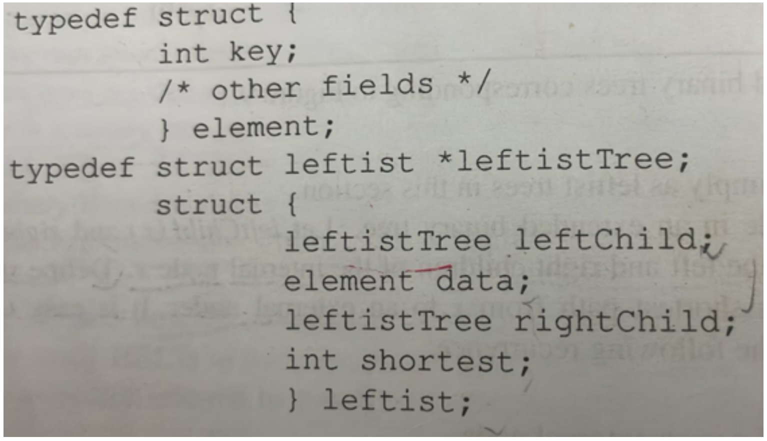
Define the leftist tree. Give its declaration in C. Check whether the given binary tree is a leftist tree or not. Explain your answer



Answer:

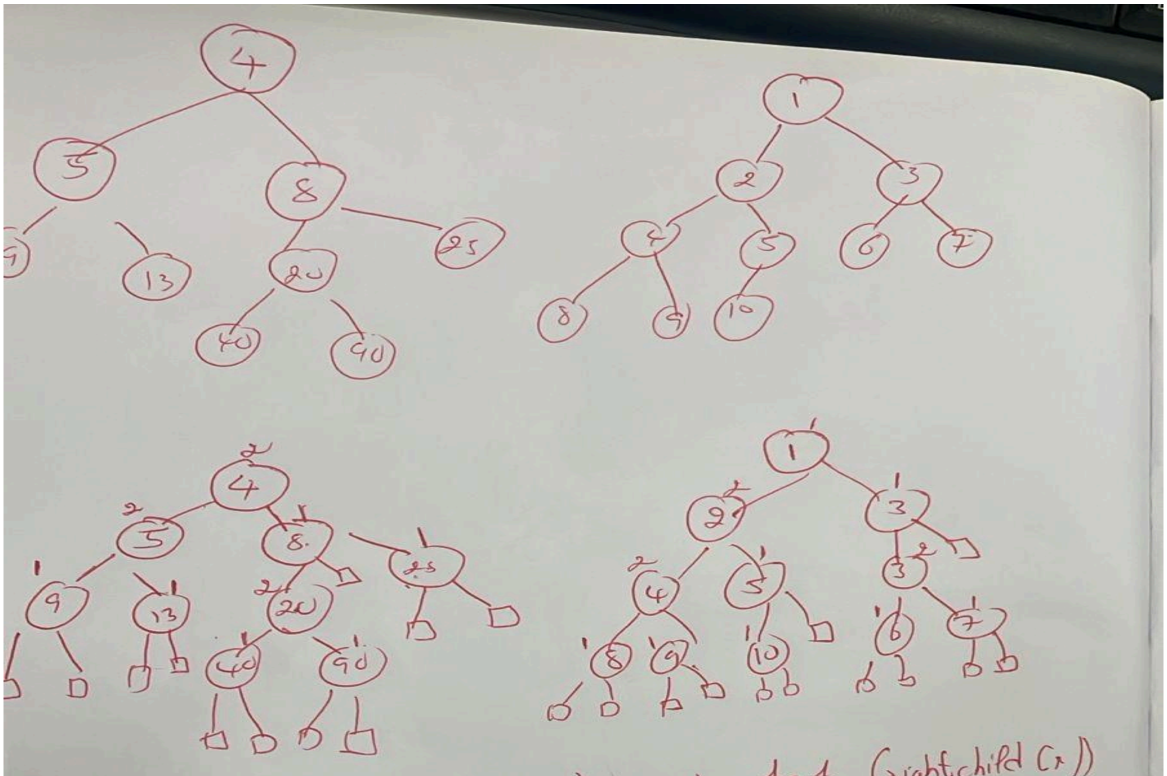
Leftist tree definition 1 M

C function 2 M



```
typedef struct {
        int key;
        /* other fields */
        } element;
typedef struct leftist *leftistTree;
        struct {
              leftistTree leftChild;
              element data;
              leftistTree rightChild;
              int shortest;
              } leftist;
```

Solution 2 M



shortest (leftchild(x)) ≥ shortest (rightchild (x))

~~2 < 1~~

∴ The given trees are leftlist trees.