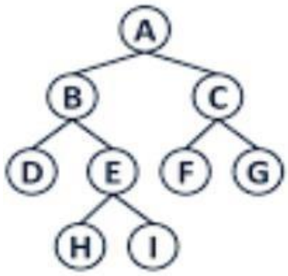
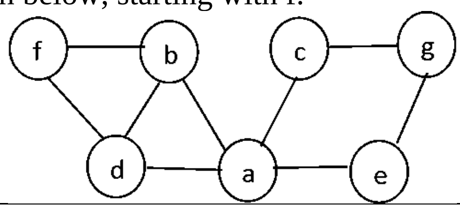
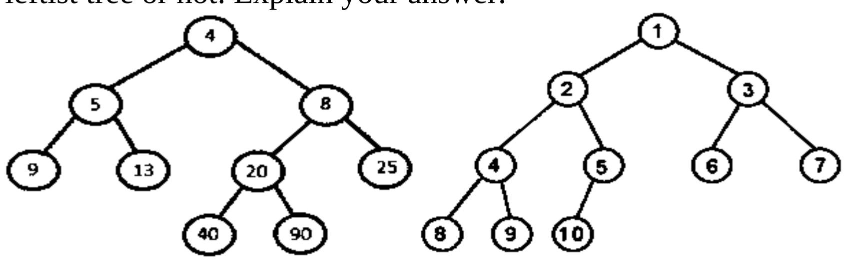
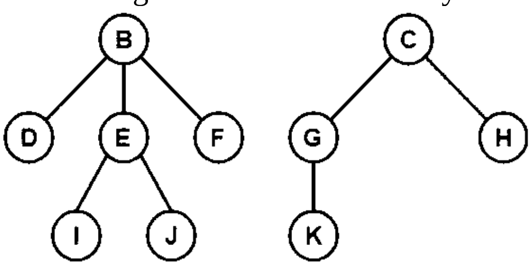
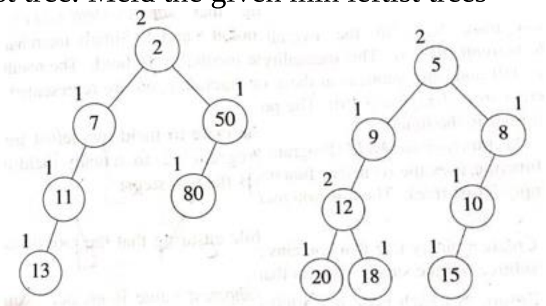
USN | | | | | | | | | | |

Internal Assessment Test 2 – Dec 2024

Sub:	Data Structures and Applications				Sub Code:	BCS304	Branch:	CSE
Date:	14-12-2024	Duration:	90 mins	Max Marks:	50	Sem / Sec:	III (A, B & C)	OBE

	<u>Answer any FIVE FULL Questions</u>	**MARKS**	**CO**	**RBT**
1.a)	Write recursive C functions for inorder, preorder and postorder traversals of a binary tree.	6	CO4	L1
1.b)	Find all the traversals for the given tree. 	4	CO4	L3
2. a)	Define the Threaded binary tree. Explain the construction of threaded binary trees, with suitable examples.	6	CO4	L2
2. b)	Define Binary Search tree. Construct a binary search tree (BST) for the following elements: 100, 85, 45, 55, 120, 20, 70, 90, 115, 65, 130, 145.	4	CO4	L3
3. a)	Write the C- function to traverse a graph using Depth First Search (DFS). Apply DFS for the graph given below, starting with f. 	6	CO4	L3
3. b)	Construct a binary tree from the Post-order and In-order sequence given below In-order: GDHBAEICF Post-order: GHDBIEFCA	4	CO4	L3
4.a)	Define the leftist tree. Give its declaration in C. Check whether the given binary tree is a leftist tree or not. Explain your answer. 	6	CO4	L3
4.b)	Define Forest. Transform the given forest into a Binary tree 	4	CO4	L2

5.a)	What is dynamic hashing? Explain the following techniques with examples: i. Dynamic hashing using directories ii. Directory less dynamic hashing	6	CO5	L1
5.b)	Construct the hash table to insert the keys: 7, 24, 18, 52, 36, 54, 11, 23 in a chained hash table of 9 memory locations. Use h(k) = k mod m.	4	CO5	L2
6. a)	Define min Leftist tree. Meld the given min leftist trees  	6	CO4	L3
6. b)	b) Explain the optimal binary search tree with a suitable example.	4	CO5	L1

CI **CCI** **HOD**

Internal Assessment Test 2 – Nov 2024

Solution

Sub	Data Structures and Applications					Sub code	BCS304
Date	07/11/24	Duration	**90 mins**	Max Marks	**50**	Sem /Sec	III A, B&C

1 a)Write recursive C functions for inorder, preorder and postorder traversals of a binary tree.

Solution:

Preorder Traversal:

```
void preorder(node *temp)
{
if(temp!=NULL)
{
        printf("\t%d",temp->info);
        preorder(temp->left);
        preorder(temp->right);
}
}
```

Postorder Traversal:

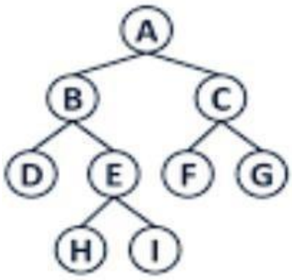
```
void postorder(node *temp)
{
if(temp!=NULL)
{
        postorder(temp->left);
        postorder(temp->right);
        printf("\t%d",temp->info);
}
}
```

Inorder Traversal:

```
void inorder(node *temp)
{
if(temp!=NULL)
{
        inorder(temp->left);
        printf("\t%d",temp->info);
        inorder(temp->right);
```

}
}

b)Find all the traversals for the given tree.



Solution:

1.b)

Inorder:- DBHEIAFCG

Preorder:- ABDEHICFG

Postorder:- DHIEBFGCA

2 a)Define the Threaded binary tree. Explain the construction of threaded binary trees, with suitable examples.

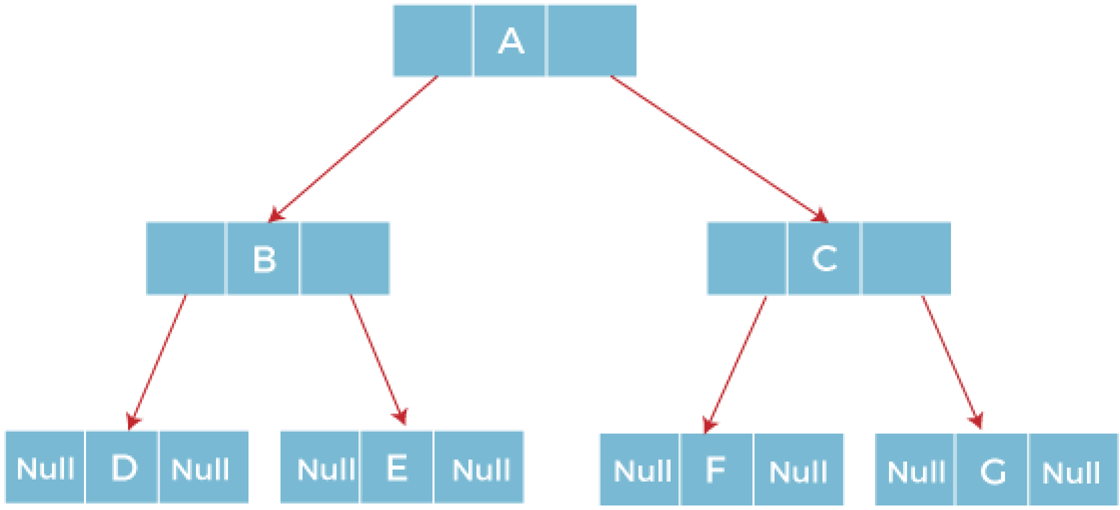
Solution:

Threaded binary tree:

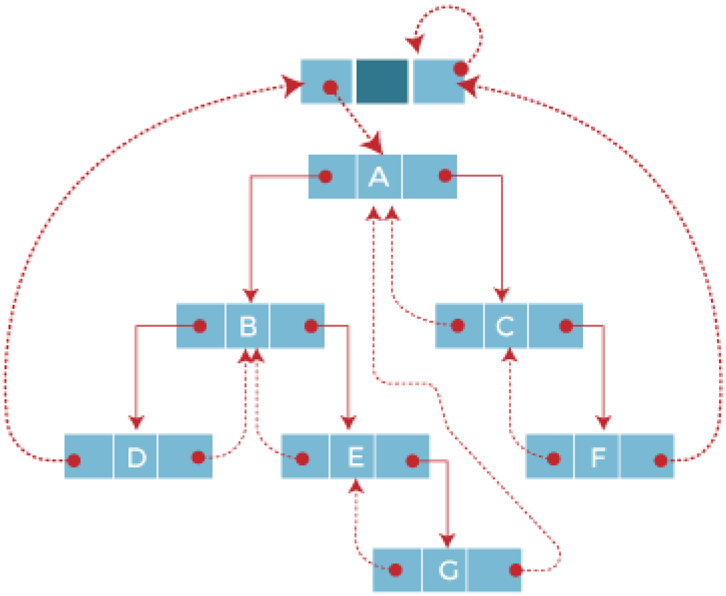
A Threaded Binary Tree is a type of binary tree where null pointers in leaf nodes are replaced with special pointers called threads. These threads provide an efficient way to traverse the tree without using a stack or recursion.

Rules to follow:

1. **If left child NULL it will point its inorder predecessor**
2. **If right child NULL it will point its inorder successor**



Threaded binary tree



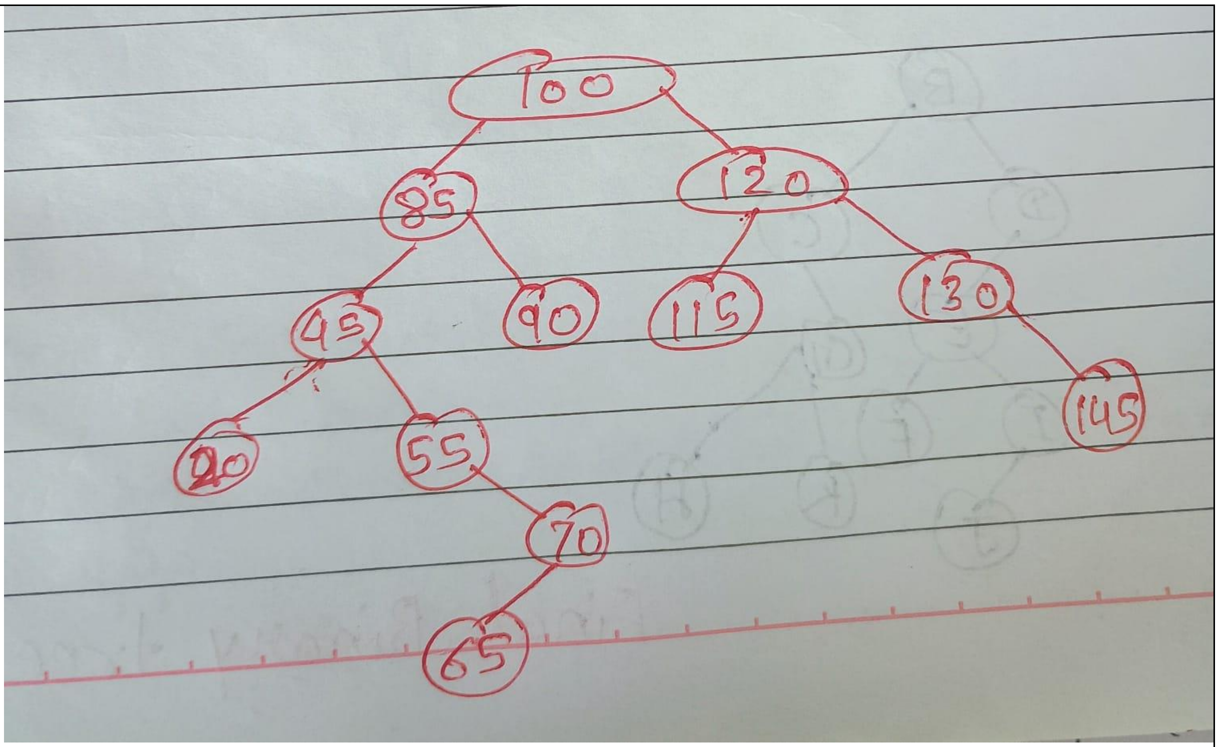
Two-way threaded - tree with header node

b)Define Binary Search tree. Construct a binary search tree (BST) for the following elements: 100, 85, 45, 55, 120, 20, 70, 90, 115, 65, 130, 145.

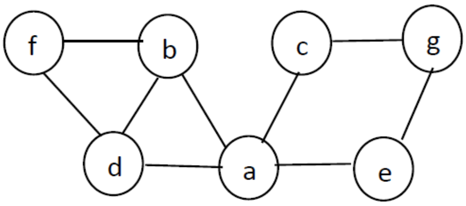
Solution:

A **Binary Search Tree (BST)** is a binary tree where:

1. Each node has a key, a left child, and a right child.
2. The **left subtree** contains keys **less than** the node's key.
3. The **right subtree** contains keys **greater than** the node's key.



3 a)Write the C- function to traverse a graph using Depth First Search (DFS). Apply DFS for the graph given below, starting with f.



Solution:

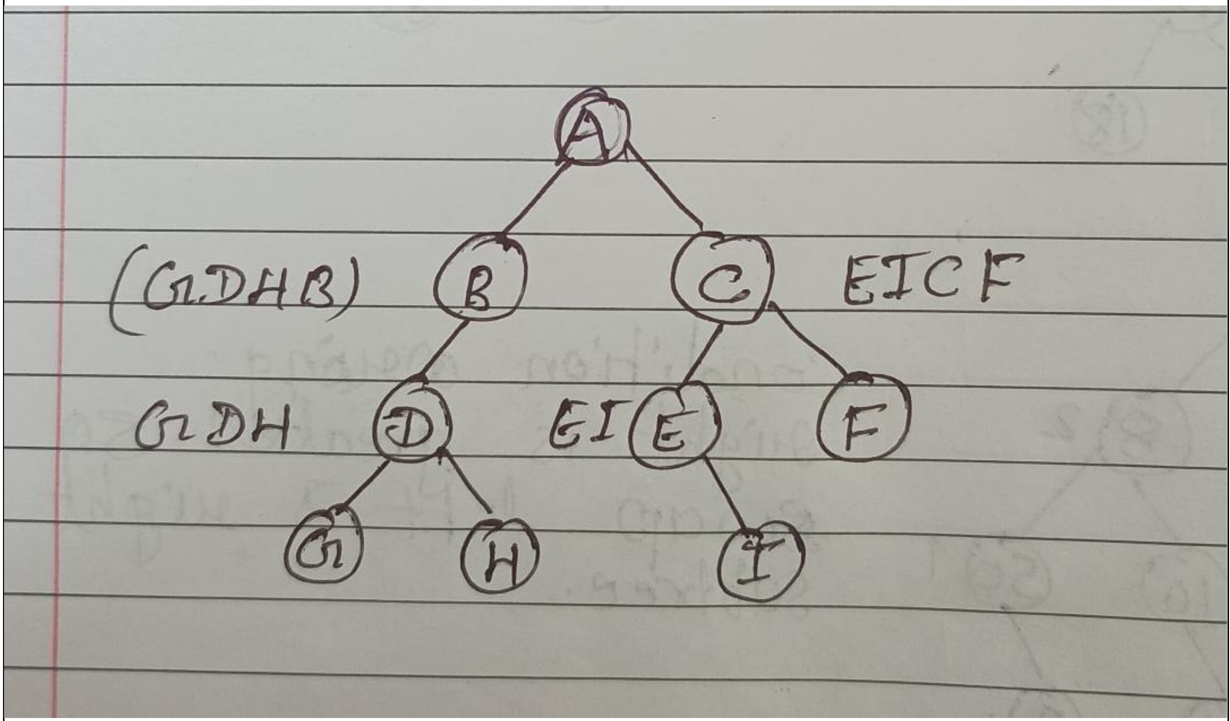
```
void dfs(int start)
{
   int visited[MAX]={0};
   int stack[MAX];
   int top=-1,i;
   printf("%d->",start);
   visited[start]=1;
   stack[++top]=start;
   for(i=0;i<MAX;i++)
   printf("%d",visited[i]);
   while(top!=-1)
   {
      start=stack[top];
      for(i=0;i<MAX;i++)
      {
         if(adj[start][i] && visited[i]==0)
         {
            stack[++top]=i;
            for(j=top;j>=0;j--)
            printf("\nStack:%d",stack[j]);
            printf("\nDFS:%d->",i);
            visited[i]=1;
            break;
         }
      }
```

```
      if(i==MAX)
      top--;
   }
}
```

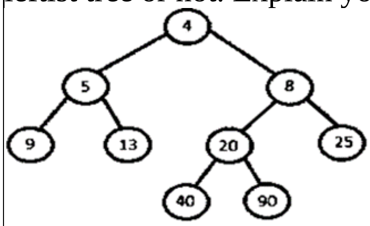
DFS traversal : f b d a c g e

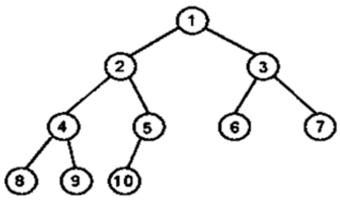
b)Construct a binary tree from the Post-order and In-order sequence given below
In-order: GDHBAEICF Post-order: GHDBIEFCA

Solution:



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





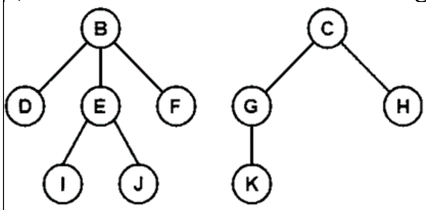
Solution:

A **leftist tree** (or **leftist heap**) is a special type of binary tree used for efficient priority queue operations, particularly merging two heaps. It maintains the **min-heap property** (the key at any node is smaller than or equal to the keys of its children) and an additional structural property to ensure balance.

```c
typedef struct LeftistNode {
    int key;                    // Key value (priority)
    int npl;                    // Null path length
    struct LeftistNode* left;   // Pointer to the left child
    struct LeftistNode* right;  // Pointer to the right child
} LeftistNode;

// Function to create a new node
LeftistNode* createNode(int key) {
    LeftistNode* node = (LeftistNode*)malloc(sizeof(LeftistNode));
    node->key = key;
    node->npl = 0; // New nodes have an NPL of 0
    node->left = NULL;
    node->right = NULL;
    return node;
}
```

b) Define Forest. Transform the given forest into a Binary tree

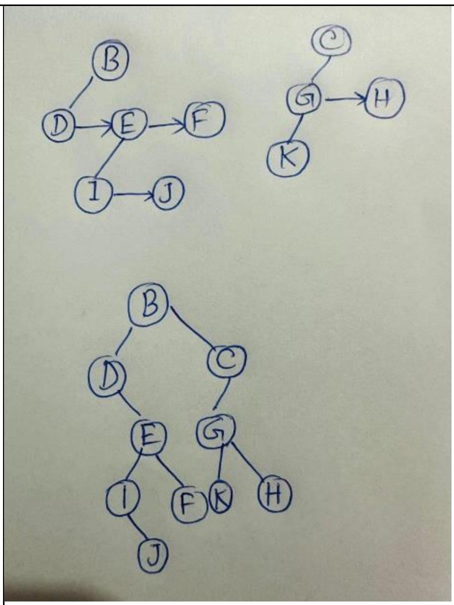


Solution:

A **forest** in data structures is a collection of **disjoint trees**, meaning there are multiple trees, and no tree shares any nodes with another. In simpler terms, it's a group of non-connected tree structures.

Key Characteristics of a Forest

1. A forest can contain one or more trees.
2. The trees in a forest are independent of each other.
3. The concept is used in hierarchical representations where multiple root nodes exist.



5	a) What is dynamic hashing? Explain the following techniques with examples:. Dynamic hashing using directories Directory less dynamic hashing

Solution: Dynamic hashing is a hashing technique that allows the hash table to grow and shrink dynamically as the data changes. It efficiently handles scenarios where the data set size is unpredictable, preventing excessive memory usage or frequent rehashing.

Dynamic Hashing Using Directories

In this technique, a directory (array of pointers) is used to manage access to the buckets. The directory size can grow or shrink as necessary. A hash function determines the index in the directory, and the directory points to corresponding buckets. The technique is commonly implemented using **extendible hashing**.

Process

1. **Hash Function:** A bit-string hash function is used (e.g., taking the first d bits).
2. **Directory:** Points to buckets, where the buckets store records.
3. **Splitting Buckets:** When a bucket overflows, only that bucket splits, and the directory adjusts accordingly.
4. **Doubling Directory:** If all buckets at a given level are full, the directory size doubles to accommodate new data.

Advantages

- Efficient memory use.
- Handles overflows with minimal rehashing

Directory-Less Dynamic Hashing

This technique eliminates the directory and directly manages data in buckets using techniques like **linear hashing**.

Process

1. **Buckets:** Organized sequentially.
2. **Hash Function:** A series of hash functions, h0,h1,h2,…, is applied as the table grows.
3. **Bucket Splitting:** When a bucket overflows, the next bucket in the sequence splits, redistributing records based on the next-level hash function.
4. **Pointerless:** No directory; pointers are internal to buckets.

Advantages

- No additional memory overhead for directories.
- Simpler structure compared to directory-based methods.

b) Construct the hash table to insert the keys: 7, 24, 18, 52, 36, 54, 11, 23 in a chained hash table of 9 memory locations. Use h(k) = k mod m.

Solution:

h(7)=7mod9=7

h(24)=24mod 9=6

h(18)=18mod 9=0

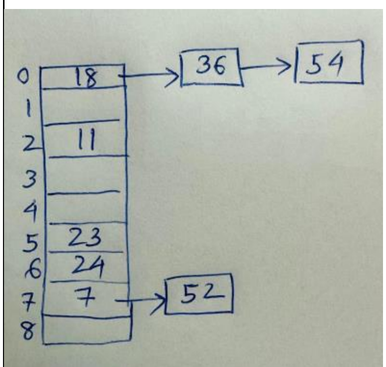
h(52)=52mod 9=7

h(36)=36mod 9=0

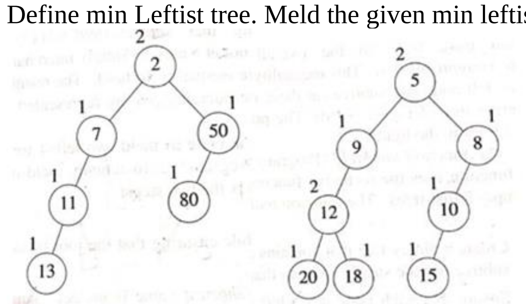
h(54)=54mod 9=0

h(11)=11mod 9=2

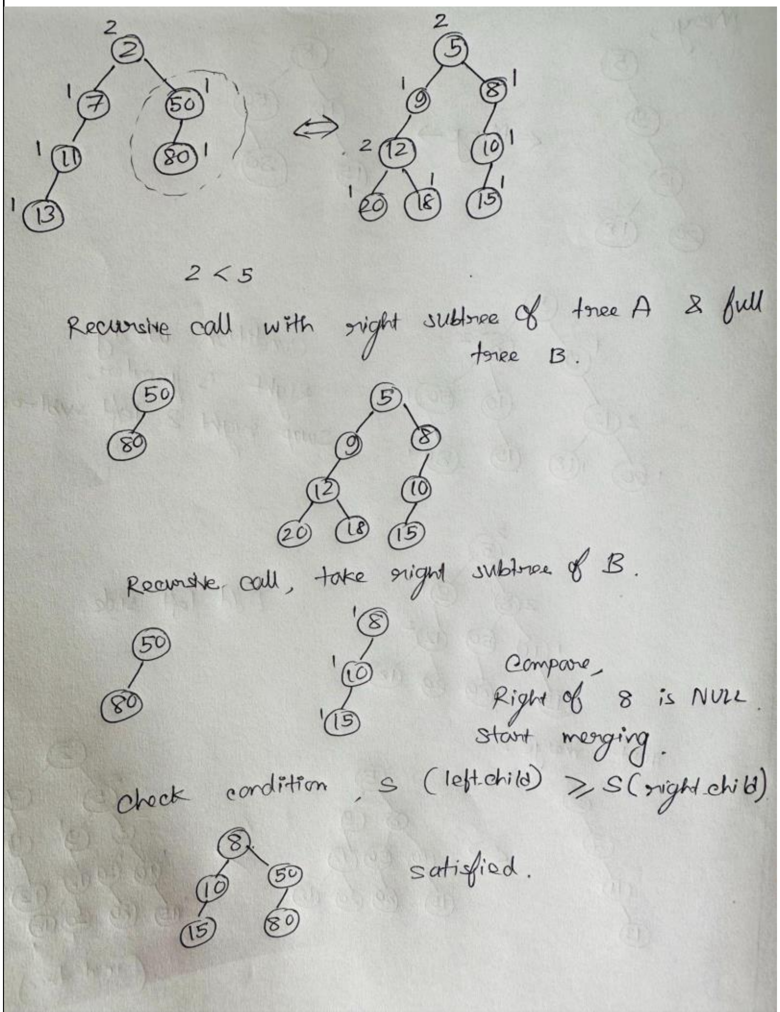
h(23)=23mod 9=5

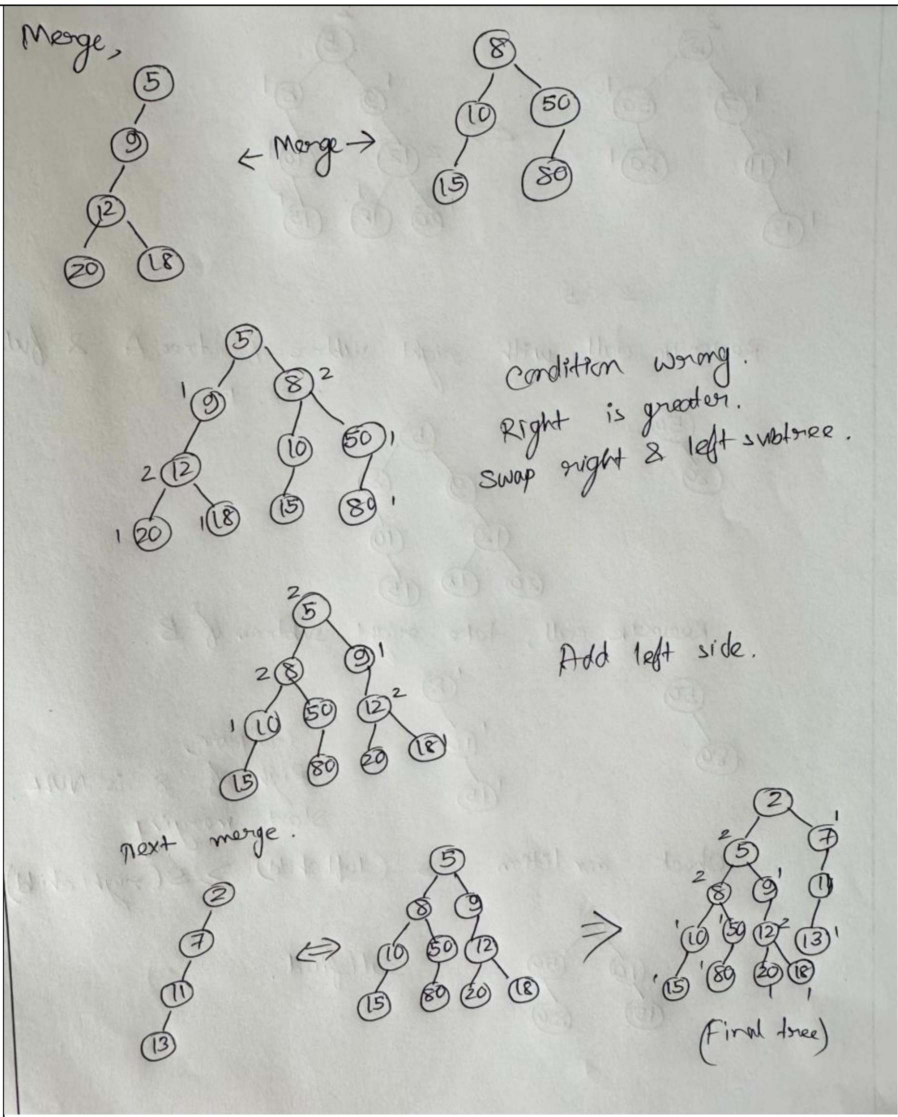


6 a) Define min Leftist tree. Meld the given min leftist trees



Solution: A **Min Leftist Tree** is a type of binary tree designed for efficient priority queue operations, adhering to two key properties: the **Min-Heap Property** and the **Leftist Property**. The Min-Heap Property ensures that the key value at any node is smaller than or equal to the keys of its children, maintaining a hierarchical ordering. The Leftist Property requires that the **Null Path Length (NPL)** of the left child is always greater than or equal to the NPL of the right child, promoting a structure with a shorter right spine for efficient merging. The NPL of a node is defined as the length of the shortest path from the node to a null child, with a null node having an NPL of −1−1−1. These properties collectively ensure that Min Leftist Trees are balanced and optimized for operations like melding, insertion, and deletion.





b) Explain the optimal binary search tree with a suitable example.

Solution: An Optimal Binary Search Tree (OBST), also known as a Weighted Binary Search Tree, is a binary search tree that minimizes the expected search cost. In a binary search tree, the search cost is the number of comparisons required to search for a given key. In an OBST, each node is assigned a weight that represents the probability of the key being searched for. The sum of all the weights in the tree is 1.0. The expected search cost of a node is the sum of the product of its depth and weight, and the expected search cost of its children.

```
Input:  keys[] = {10, 12, 20}, freq[] = {34, 8, 50}
There can be following possible BSTs
    10                12                 20         10              20
      \             /    \              /             \            /
      12          10     20           12               20         10
        \                            /                 /           \
         20                        10               12              12
     I               II             III             IV              V
Among all possible BSTs, cost of the fifth BST is minimum.
Cost of the fifth BST is 1*50 + 2*34 + 3*8 = 142
```