Question 1:

Construction a Binary tree program that inserts nodes in level order (breadth-first insertion) and then performs a preorder traversal of the tree. Your task is to identify the output of the program given a specific sequence of node values as input.

Input Format

- The first line contains a single integer N, the number of nodes to be inserted into the binary tree.
- The next N lines contain N integers, each representing the value of a node to be inserted into the binary tree. The nodes are inserted in the order they are provided.

Output Format

- Output a single line containing the preorder traversal of the binary tree, with node values separated by space.

Constraints

- Implement class Node.
- Implement class Binary Tree.

Title for Question 1: Binary Tree

Solution:

```
#include <iostream>
#include <queue>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

class BinaryTree {
private:
    Node* root;

public:
    BinaryTree() : root(nullptr) {}

    // Function to insert a node into the binary tree
    void insert(int val) {
        if (root == nullptr) {
```

```
            root = new Node(val);
            return;
        }

        queue<Node*> q;
        q.push(root);

        while (!q.empty()) {
            Node* temp = q.front();
            q.pop();

            if (temp->left != nullptr) {
                q.push(temp->left);
            } else {
                temp->left = new Node(val);
                return;
            }

            if (temp->right != nullptr) {
                q.push(temp->right);
            } else {
                temp->right = new Node(val);
                return;
            }
        }
    }

    // Function to print the binary tree in preorder traversal
    void preorderTraversal(Node* node) {
        if (node != nullptr) {
            cout << node->data << " ";
            preorderTraversal(node->left);
            preorderTraversal(node->right);
        }
    }

    // Wrapper function to perform preorder traversal
    void preorder() {
        preorderTraversal(root);
    }
};

int main() {
    BinaryTree tree;
    int numNodes;
    cin >> numNodes;
    for (int i = 0; i < numNodes; ++i) {
        int val;
        cin >> val;
        tree.insert(val);
    }
    tree.preorder();
    cout << endl;

    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 1 2 3 4 5	1 2 4 5 3
2	7 1 2 3 4 5 6 7	1 2 4 5 3 6 7
3	3 7 8 9	7 8 9
4	6 6 5 4 3 2 1	6 5 3 2 4 1
5	4 100 50 150 200	100 50 200 150
6	9 50 25 75 12 37 62 87 6 18	50 25 12 6 18 37 75 62 87

White List: class Node, BinaryTree

Black List:

Question 2:

Design a program that checks if a binary tree is complete. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible. You need to use the program to determine if the constructed binary tree, based on a series of inputs that specify node insertions, is complete.

Input Format

- The first line contains an integer, root_value, the value of the root node of the binary tree.
- Following lines will consist of a sequence of operations until a 0 is encountered. Each operation begins with an integer choice:If choice is 1, it indicates the addition of a left child.
- If choice is 2, it indicates the addition of a right child.
- A 0 indicates the end of the input.
- For each non-zero choice, two integers follow:
- The first integer, parent_value, specifies the value of the parent node to which a child will be added.
- The second integer, child_value, specifies the value of the child node to be added to the tree.

Output Format

- Print either "The binary tree is complete." or "The binary tree is not complete.", depending on whether the constructed binary tree meets the criteria for a complete binary tree.

Title for Question 2: Complete Binary Tree

Solution:

```
#include <iostream>
#include <queue>
using namespace std;

struct TreeNode {
    int data;
    TreeNode* left;
```

```
    TreeNode* right;
};

TreeNode* createNode(int data) {
    TreeNode* newNode = new TreeNode();
    newNode->data = data;
    newNode->left = nullptr;
    newNode->right = nullptr;
    return newNode;
}

bool isComplete(TreeNode* root) {
    if (root == nullptr)
        return true;
    queue<TreeNode*> q;
    bool isNullFound = false;
    q.push(root);
    while (!q.empty()) {
        TreeNode* temp = q.front();
        q.pop();
        if (temp == nullptr) {
            isNullFound = true;
            continue;
        }
        if (isNullFound)
            return false;
        q.push(temp->left);
        q.push(temp->right);
    }
    return true;
}

int main() {
    TreeNode* root = nullptr;
    int data, choice;
    // cout << "Enter the root value: ";
    cin >> data;
    root = createNode(data);
    // cout << "Enter 1 to add a left child, 2 to add a right child, and
    cin >> choice;
    while (choice) {
        TreeNode* temp = root;
        // cout << "Enter the parent value: ";
        cin >> data;
        while (temp != nullptr) {
            if (temp->data == data) {
                break;
            } else if (data < temp->data) {
                temp = temp->left;
            } else {
                temp = temp->right;
            }
        }
        if (temp != nullptr) {
            // cout << "Enter the child value: ";
            cin >> data;
            if (choice == 1)
                temp->left = createNode(data);
            else
                temp->right = createNode(data);
```

```
        }
        // cout << "Enter 1 to add a left child, 2 to add a right child,
        cin >> choice;
    }
    if (isComplete(root))
        cout << "The binary tree is complete.";
    else
        cout << "The binary tree is not complete.";
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 1 5 10 2 5 15 0	The binary tree is complete.
2	10 1 10 5 1 5 2 0	The binary tree is not complete.
3	1 0	The binary tree is complete.
4	1 2 1 2 0	The binary tree is not complete.
5	1 1 1 2 2 1 3 2 3 4 0	The binary tree is not complete.
6	1 1 1 2 2 1 3 0	The binary tree is complete.

White List:

Black List:

Question 3:

A program that implements a Max Heap cl. The Max Heap property ensures that for every node other than the root, the value of the node is less than or equal to the value of its parent. Your task is to use this program to perform a series of insertions into the heap and then perform operations to query and remove the maximum element in the heap.

Input Format

- The first line contains an integer N, representing the number of elements to be inserted into the Max Heap.
- The next N lines each contain an integer value, representing the elements to be inserted into the heap, one per line.

Output Format

- The first line should output the string "Max element in the heap: " followed by the maximum element in the heap after all insertions are completed.
- The second line should output the string "Removing max element...".
- The third line should output the string "Max element in the heap after removal: " followed by the new maximum element in the heap after the top element is removed.

Title for Question 3: Heap

Solution:

```
#include <iostream>
#include <vector>
using namespace std;

class MaxHeap {
private:
    vector<int> heap;

    void heapifyUp(int index) {
        int parentIndex = (index - 1) / 2;
        while (index > 0 && heap[index] > heap[parentIndex]) {
            swap(heap[index], heap[parentIndex]);
            index = parentIndex;
            parentIndex = (index - 1) / 2;
        }
    }

    void heapifyDown(int index) {
        int leftChild = 2 * index + 1;
        int rightChild = 2 * index + 2;
        int largest = index;

        if (leftChild < heap.size() && heap[leftChild] > heap[largest])
            largest = leftChild;

        if (rightChild < heap.size() && heap[rightChild] > heap[largest])
            largest = rightChild;

        if (largest != index) {
            swap(heap[index], heap[largest]);
            heapifyDown(largest);
        }
    }

public:
    void insert(int value) {
        heap.push_back(value);
        heapifyUp(heap.size() - 1);
    }

    int getMax() {
        if (heap.empty()) {
            cerr << "Heap is empty.\n";
            return -1; // Or any appropriate error handling
        }
        return heap[0];
    }

    void removeMax() {
        if (heap.empty()) {
            cerr << "Heap is empty.\n";
            return; // Or any appropriate error handling
        }
        heap[0] = heap.back();
        heap.pop_back();
        heapifyDown(0);
```

```
    }
};

int main() {
    MaxHeap heap;
    int n, value;
    cin >> n;
    for (int i = 0; i < n; ++i) {
        cin >> value;
        heap.insert(value);
    }
    cout << "Max element in the heap: " << heap.getMax() << endl;
    cout << "Removing max element...\n";
    heap.removeMax();
    cout << "Max element in the heap after removal: " << heap.getMax() <<
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	3 1 2 3	Max element in the heap: 3 Removing max element... Max element in the heap after removal: 2
2	4 10 20 5 15	Max element in the heap: 20 Removing max element... Max element in the heap after removal: 15
3	5 40 35 15 25 10	Max element in the heap: 40 Removing max element... Max element in the heap after removal: 35
4	6 8 3 9 10 2 7	Max element in the heap: 10 Removing max element... Max element in the heap after removal: 9
5	2 100 200	Max element in the heap: 200 Removing max element... Max element in the heap after removal: 100
6	7 55 45 60 40 50 65 35	Max element in the heap: 65 Removing max element... Max element in the heap after removal: 60

White List: class MaxHeap

Black List:

Question 4:

Create a program that implements the Heap Sort algorithm to sort an array of integers in ascending order. Your program should first build a mi heap from the input array and then perform the sorting operation by repeatedly removing the largest element from the heap and adjusting the heap accordingly.

Input Format:

- The first line contains a single integer n, representing the number of elements in the array.
- The second line contains n space-separated integers, representing the elements of the array.

Output Format:

- The first line should print "Original array: " followed by the original elements of the array, separated by spaces.
- The second line should print "Sorted array: " followed by the sorted elements of the array, separated by spaces.

Title for Question 4: Heap Sort

Solution:

```
#include <iostream>
#include <vector>

// Function to heapify a subtree rooted at index i
void heapify(std::vector<int>& arr, int n, int i) {
    int largest = i;    // Initialize largest as root
    int left = 2 * i + 1;  // left = 2*i + 1
    int right = 2 * i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (left < n && arr[left] > arr[largest])
        largest = left;

    // If right child is larger than largest so far
    if (right < n && arr[right] > arr[largest])
        largest = right;

    // If largest is not root
    if (largest != i) {
        std::swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// Main function to do heap sort
void heapSort(std::vector<int>& arr) {
    int n = arr.size();

    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i = n - 1; i > 0; i--) {
        // Move current root to end
        std::swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
```

```
    }
}

// Utility function to print an array
void printArray(const std::vector<int>& arr) {
    for (int i : arr)
        std::cout << i << " ";
    std::cout << std::endl;
}

int main() {
    int n;
    std::cin >> n;
    std::vector<int> arr(n);
    for (int i = 0; i < n; ++i)
        std::cin >> arr[i];
    std::cout << "Original array: ";
    printArray(arr);
    heapSort(arr);
    std::cout << "Sorted array: ";
    printArray(arr);
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 4 2 7 1 9	Original array: 4 2 7 1 9 Sorted array: 1 2 4 7 9
2	6 10 5 8 2 6 3	Original array: 10 5 8 2 6 3 Sorted array: 2 3 5 6 8 10
3	3 12 6 4	Original array: 12 6 4 Sorted array: 4 6 12
4	8 15 13 11 10 9 7 5 3	Original array: 15 13 11 10 9 7 5 3 Sorted array: 3 5 7 9 10 11 13 15
5	4 1 2 3 4	Original array: 1 2 3 4 Sorted array: 1 2 3 4
6	2 10 1	Original array: 10 1 Sorted array: 1 10

White List:

Black List:

Question 5:

Implement a priority queue using a binary heap. The priority queue should support the insertion of elements and removal of the smallest element. Your program should read a series of integers from input, add them to the priority queue, and then remove them in sorted order (from smallest to largest).

Input Format:

- The first line contains a single integer n, representing the number of elements to be added to the priority queue.
- The second line contains n space-separated integers representing the elements to be added to the priority queue.

Output Format:

- A single line starting with "Elements in priority queue in sorted order: " followed by the n elements in ascending order, separated by spaces.

Title for Question 5: Priority Queue in Heap

Solution:

```
#include <iostream>
#include <vector>

using namespace std;

class PriorityQueue {
private:
    vector<int> heap;

    void heapifyUp(int index) {
        while (index > 0) {
            int parentIndex = (index - 1) / 2;
            if (heap[parentIndex] > heap[index]) {
                swap(heap[parentIndex], heap[index]);
                index = parentIndex;
            } else {
                break;
            }
        }
    }

    void heapifyDown(int index) {
        int leftChild, rightChild, minIndex;
        while (true) {
            leftChild = 2 * index + 1;
            rightChild = 2 * index + 2;
            minIndex = index;

            if (leftChild < heap.size() && heap[leftChild] < heap[minInde
                minIndex = leftChild;
            }

            if (rightChild < heap.size() && heap[rightChild] < heap[minIn
                minIndex = rightChild;
            }

            if (minIndex != index) {
                swap(heap[minIndex], heap[index]);
                index = minIndex;
            } else {
                break;
            }
        }
    }
```

```
public:
    void push(int value) {
        heap.push_back(value);
        heapifyUp(heap.size() - 1);
    }

    int pop() {
        if (heap.empty()) {
            cerr << "Priority queue is empty!" << endl;
            return -1; // You can throw an exception here instead
        }

        int minValue = heap[0];
        heap[0] = heap.back();
        heap.pop_back();
        heapifyDown(0);
        return minValue;
    }

    bool isEmpty() {
        return heap.empty();
    }
};

int main() {
    PriorityQueue pq;
    int num_elements;
    cin >> num_elements;
    for (int i = 0; i < num_elements; ++i) {
        int element;
        cin >> element;
        pq.push(element);
    }
    cout << "Elements in priority queue in sorted order: ";
    while (!pq.isEmpty()) {
        cout << pq.pop() << " ";
    }
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 3 7 1 4 2	Elements in priority queue in sorted order: 1 2 3 4 7
2	4 9 5 8 3	Elements in priority queue in sorted order: 3 5 8 9
3	3 12 6 10	Elements in priority queue in sorted order: 6 10 12
4	7 2 4 1 6 9 5 3	Elements in priority queue in sorted order: 1 2 3 4 5 6 9
5	6 11 15 7 18 13 9	Elements in priority queue in sorted order: 7 9 11 13 15 18
6	2 20 17	Elements in priority queue in sorted order: 17 20

White List:

Black List:

Question 6:

Implement the Heap Sort algorithm to sort a given array of integers in non-decreasing order. The program should first convert the input array into a m heap. Then, it should repeatedly swap the first element of the heap (the maximum element) with the last element of the current heap, reduce the size of the heap by one, and finally, heapify the root of the tree. Continue this process until the heap size is reduced to one.

Input Format:

- The first line contains a single integer n, the number of elements in the array.
- The second line contains n space-separated integers arr[i], representing the elements of the array.

Output Format:

- The first line should print "Original array: " followed by the original elements of the array, separated by spaces.
- The second line should print "Sorted array: " followed by the sorted elements of the array, separated by spaces.

Title for Question 6: Implement a Heap Sort

Solution:

```
#include <iostream>
#include <vector>

// Function to heapify a subtree rooted at index i
void heapify(std::vector<int>& arr, int n, int i) {
    int largest = i;   // Initialize largest as root
    int left = 2 * i + 1; // left = 2*i + 1
    int right = 2 * i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (left < n && arr[left] > arr[largest])
        largest = left;
```

```
    // If right child is larger than largest so far
    if (right < n && arr[right] > arr[largest])
        largest = right;

    // If largest is not root
    if (largest != i) {
        std::swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// Main function to do heap sort
void heapSort(std::vector<int>& arr) {
    int n = arr.size();

    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i = n - 1; i > 0; i--) {
        // Move current root to end
        std::swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
```

```
}

// Utility function to print an array
void printArray(const std::vector<int>& arr) {
    for (int i : arr)
        std::cout << i << " ";
    std::cout << std::endl;
}

int main() {
    int n;
    std::cin >> n;
    std::vector<int> arr(n);
    for (int i = 0; i < n; ++i)
        std::cin >> arr[i];
    std::cout << "Original array: ";
    printArray(arr);
    heapSort(arr);
    std::cout << "Sorted array: ";
    printArray(arr);
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	3 12 6 4	Original array: 12 6 4 Sorted array: 4 6 12
2	8 15 13 11 10 9 7 5 3	Original array: 15 13 11 10 9 7 5 3 Sorted array: 3 5 7 9 10 11 13 15
3	4 1 2 3 4	Original array: 1 2 3 4 Sorted array: 1 2 3 4
4	5 4 2 7 1 9	Original array: 4 2 7 1 9 Sorted array: 1 2 4 7 9
5	6 10 5 8 2 6 3	Original array: 10 5 8 2 6 3 Sorted array: 2 3 5 6 8 10
6	2 10 1	Original array: 10 1 Sorted array: 1 10

White List:

Black List: