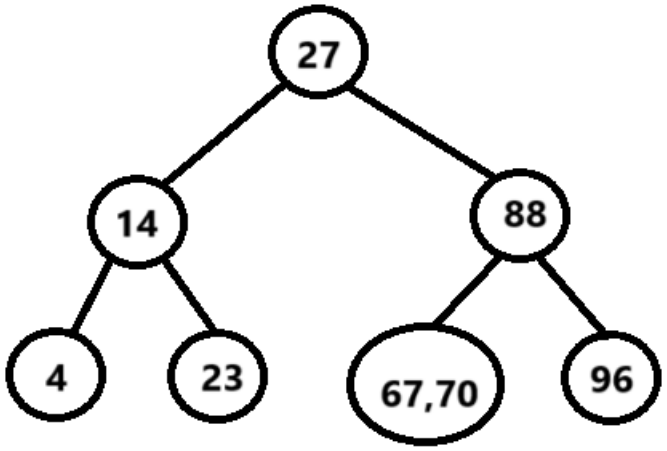
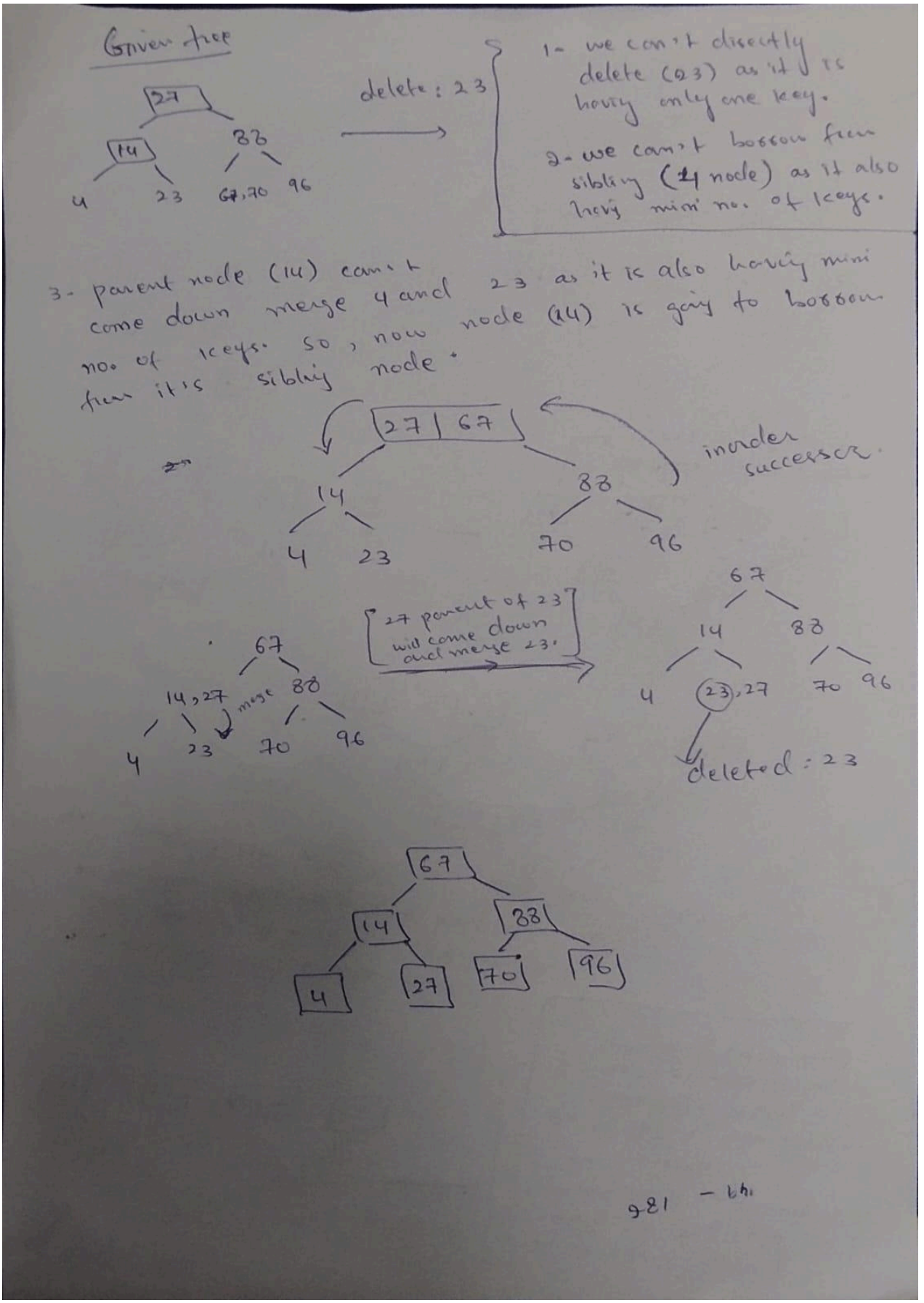
Name and ID: **Solution** Section: BAI-3A Marks: 15

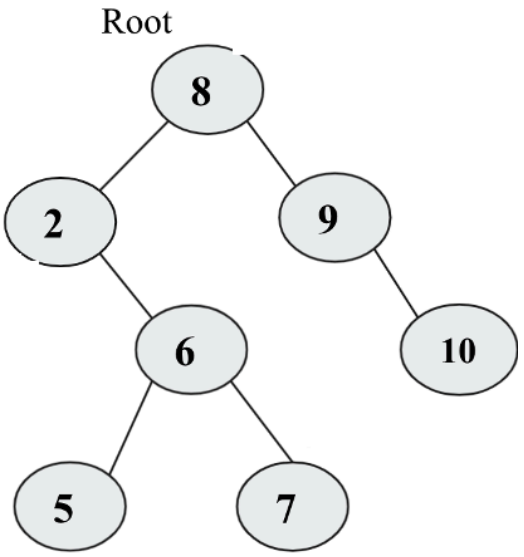
Question#01 [3+5 Marks]

a. Given the 2-3 tree below, delete 23. Show each step of the process clearly and the final resulting tree by drawing each step clearly.

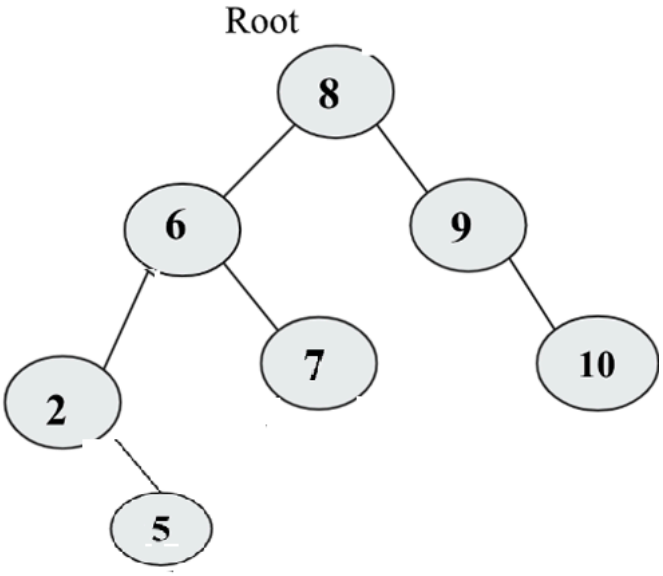




b. Find which node is imbalance in the following tree and which AVL rotation is used to balance the node. Show rotation dry run and write C++ function for that rotation case.



Rotation case: RR imbalance (left rotate)



left rotation

```
Node* RR_rotation(Node* node) {
        Node* child = node->right_node;
        node->right_node = child->left_node;
        child->left_node = node;

        node->height = max(get_height(node->left_node), get_height(node->right_node)) + 1;
        child->height = max(get_height(child->left_node), get_height(child->right_node)) + 1;

        return child;
    }
```

Question#02 [7 Marks]

Implement the Heap Sort algorithm in C++. You are given an array of integers. Your task is to sort the array in ascending order using the Heap Sort algorithm.

```
#include <bits/stdc++.h>
```

```
using namespace std;

// To heapify a subtree rooted with node i
// which is an index in arr[].
void heapify(vector<int>& arr, int n, int i) {
    // Initialize largest as root (for ascending order)
    int largest = i;

    // left index = 2*i + 1
    int l = 2 * i + 1;

    // right index = 2*i + 2
    int r = 2 * i + 2;

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

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

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

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

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

    // Build heap (rearrange vector as a max-heap)
    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
        swap(arr[0], arr[i]);

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

// A utility function to print vector of size n
void printArray(vector<int>& arr) {
    for (int i = 0; i < arr.size(); ++i)
        cout << arr[i] << " ";
    cout << "\n";
}

// Driver's code
int main() {
    vector<int> arr = { 9, 4, 3, 8, 10, 2, 5 };

    // Function call
    heapSort(arr);

    cout << "Sorted array in ascending order is:\n";
    printArray(arr);
```

```
    return 0;
}
```