**Huffman Coding**

**Difficulty: MEDIUM**

**Avg. time to solve**

**25 min**

**Success Rate**

**75%**

**Problem Statement**

**You are given an array 'ARR' of Integers having 'N' elements. The array contains an encoded message. For each index 'i', 'ARR[i]' denotes the frequency of the 'i'th' character in the message. The characters are of an alien language having 'N' alphabets. Given the frequency of each of the 'N' alphabets in the message, your task is to find out the Huffman codes for each of the 'N' alphabets in the message.**

**The Huffman Code for a message is the set of codes such that :**

1) All codes are binary strings.

2) Each code should be able to determine its corresponding character uniquely.

3) The total numbers of bits used to represent the message are minimized.

**Note:**

If there are multiple set of valid Huffman codes for a message. You can print any of them.

**For example:**

Consider the array ARR = [ 1, 4, 2 ] having 3 elements.

The array containing Huffman Codes for the above array will be [ '10', '0', '11' ]. Other Valid Huffman Codes are [ '01', '1', '00' ], [ '00', '1', '01' ] etc. Codes like [ '1', '0', '01' ], [ '1', '10' , '0' ] are some of the invalid Huffman Codes.

**Input Format:**

The first line of the input contains an integer, 'T,’ denoting the number of test cases.

The first line of each test case contains an integer 'N', denoting the number of elements in the array 'ARR'.

The second line of each test case contains 'N' space-separated integers denoting the array elements.

**Output Format:**

The checker will print 1 if the returned Huffman codes are correct and follow all the rules, otherwise, it will print 0.

Print the output of each test case in a new line.

**Note :**

You do not need to print anything. It has already been taken care of. Just implement the given function.

**Constraints:**

1 <= T <= 10

1 <= N <= 10^4

1 <= ARR[i] <= 10^4

Where 'T' denotes the number of test cases, 'N' denotes the elements in the array 'ARR', and 'ARR[i]' denotes the 'i'th' element of the array 'ARR'.

Time Limit: 1 sec

**Sample Input 1:**

2

3

5 7 6

2

4 3

**Sample Output 1:**

1

1

**Explanation For Sample Input 1:**

For the first test case :

The array representing the Huffman codes will be [ '11', '0', '10' ] . Note that there are multiple other possible answers like [ '00', '1' ,'01' ], [ '01', '1', '00' ] etc. All of them are valid, so we can return any of them.

For the second test case :

The array representing the Huffman codes will be [ '0', '1' ]. The array [ '1', '0' ] also represents a valid set of Huffman Code.

**Sample Input 2:**

2

3

1 2 5

4

5 6 3 1

**Sample Output 2:**

1

1

**Explanation For Sample Input 2:**

For the first test case :

The array representing the Huffman codes will be [ '11', '10', '0' ] .

For the second test case :

The array representing the Huffman codes will be [ '10', '0', '110', '111' ].

class Node {

    public:

        int val, index;

        Node\* left=NULL, \*right=NULL;

        Node(int val, int index) {

            this->val=val;

            this->index=index;

        }

};

struct CustomCompare {

    bool  operator() (Node\* n1, Node\* n2) {

        return n1->val>n2->val;

    }

};

void Inorder(Node\* t, string path, vector<string> &ans) {

    if (!t) return;

    if (!t->left and !t->right) {

        ans[t->index]=path;

        return;

    }

    Inorder(t->left, path+'0', ans);

    Inorder(t->right, path+'1', ans);

}

vector<string> huffmanCode(vector<int> &arr) {

    //    Write your code here

    int n=arr.size();

    priority\_queue<Node\*, vector<Node\*>, CustomCompare> pq;

    for (int i=0; i<n; i++) {

        Node\* node=new Node(arr[i], i);

        pq.push(node);

    }

    Node\* root=NULL;

    while (pq.size()>1) {

        Node\* n1=pq.top();

        pq.pop();

        Node\* n2=pq.top();

        pq.pop();

        Node\* sum=new Node((n1->val)+(n2->val), -1);

        sum->left=n2;

        sum->right=n1;

        pq.push(sum);

        root=sum;

    }

    vector<string> HuffmanCode(n, "0");

    Inorder(root, "", HuffmanCode);

    return HuffmanCode;

}