**Q1 Given an array A of N integers. Find the number of good triplets (i, j, k), where i, j, k are all distinct indices such that 0 < i , j , k <= N. A good triplet (i, j, k) is a triplet such that the sum, S = A[i] + A[j] + A[k], is divisible by exactly one of A[i], A[j], or A[k].**

**Array values of a triplet (i,j,k) is (A[i], A[j], A[k]).**

**input: N=4 A=[1,2,2,1]**

**output: 12**

**Explanation : S=2+2+1=5 is divisible only by number 1 in the triplet and the triplet with array values 1, 1, 2 is not a good triplet as S = 4 is divisible by all three. So there are two triplets (1,2,2) and (2,2,1). Look at the i,j,k values which are indices. So, there are 12 possibilities of triplets of indices that can have array values as 2, 2, 1. They are: (1, 2, 3), (1, 3, 2), (2, 1, 3), (3, 1, 2), (2, 3, 1), (3, 2, 1), (2, 3, 4), (2, 4, 3), (3, 2, 4), (4, 2, 3), (3, 4, 2), (4, 3, 2). I tried this (but i want to decrease the time complexity):**

**ANS:-** def good\_triplets (arr, n):

c=0

for i in range(0,n-2):

for j in range(i+1,n-1):

for k in range(j+1,n):

sum=arr[i]+arr[j]+arr[k]

if(sum%arr[i]==0 and sum%arr[j]!=0 and sum%arr[k]!=0):

c=c+1

elif(sum%arr[j]==0 and sum%arr[k]!=0 and sum%arr[i]!=0):

c=c+1

elif(sum%arr[k]==0 and sum%arr[i]!=0 and sum%arr[j]!=0):

c=c+1

return(c\*6)

n = int(input())

arr = []

for i in range(n) :

x = int(input())

arr.append(x)

out\_ = good\_triplets(arr, n)

print (out\_)

**Q2Problem Statement** You are given a tree with N nodes rooted at 1. Each of the N nodes have some special number Se related to it. Each Node also has certain Power. Power of each node of the tree is defined as the number of heavy nodes in the subtree of the node (including that node). A heavy node is a node whose sum of divisors of its special number is a multiple of 3. You are given Q queries  
There are two types of queries : Type 1: Update special number of node. Type 2: Tell the power of a certain node.

**Input Format**

First line: Two space separated integers N and Q denoting number of nodes in the tree and number of queries respectively.Each of the next N-1 lines: Two space separated integers U and V denoting there is an edge between them. Next line: N space separated integers, i of which denotes the special number related to node i.First integer in next Q lines is T (the type of query). if T is 1, it is followed by 2 integers X and Y denoting special number S of node X is to be updated to Y if T is 2, it is followed by single integer X

**Output Format**

For each query of type 2, output the power of the given node X. Answer for each query should come in a new line

**Explaination of my problem statement** Basically, we have a tree with numbered nodes (Si). Some of these nodes presumably have children (since it's a tree).

What the question is asking for is to find the "Power" of a node, where power is defined as the number of "heavy nodes" in it's children.

A "heavy node" is defined as a node which has the sum of its divisors for Si as a multiple of 3.

There is a bunch of information we need to calculate here:  
\* For a given node, we need to get all it's children nodes  
\* For each of child node, we need to find the divisors for that node's number  
\* We need to sum the divisors and determine if it's a multiple of 3

**given sample input and outputs like**

sample input

5 5

1 2

1 3

3 4

3 5

16 8 17 3 18

2 1

2 3

1 3 7

2 1

2 3

sample output

3

2

2

1

ANS function BinarySearchTree() {

this.root = null;

}

BinarySearchTree.prototype.insertNode = function (val) {

var node = {

data : val,

left : null,

right : null

};

var currentNode;

if (!this.root) {

this.root = node;

} else {

currentNode = this.root;

while (currentNode) {

if (val < currentNode.data) {

if (!currentNode.left) {

currentNode.left = node;

break;

} else {

currentNode = currentNode.left;

}

} else if (val > currentNode.data) {

if (!currentNode.right) {

currentNode.right = node;

break;

} else {

currentNode = currentNode.right;

}

} else {

break;

}

}

}

};

var BST = new BinarySearchTree();

BST.insertNode(16);

BST.insertNode(8);

BST.insertNode(17);

BST.insertNode(3);

BST.insertNode(18);

console.log(BST);