Magic Sum of Divisors

JAVA Solution: (https://leetcode.com/problems/four-divisors/discuss/659702/JAVA-or-Clean-Code-Solution)

```java
class Solution {

    int number = 1, sum = 1;

    private void divisor(int n) {

        for(int i = 2; i * i <= n; i++) {
            if(n % i == 0) {
                sum += i;
                if(n / i == i) ++number;
                else {
                    sum += n / i;
                    number += 2;
                }
            }
            if(number > 4) break;
        }
    }

    public int sumFourDivisors(int[] nums) {

        int finalSum = 0;
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        for(int i = 0; i < nums.length; i++) {
            number = 2;
            sum = nums[i] + 1;
            if(!map.containsKey(nums[i])) divisor(nums[i]);


            if(map.containsKey(nums[i]) || number == 4) {

                if(!map.containsKey(nums[i])) map.put(nums[i], sum);

                finalSum += map.get(nums[i]);

            }

        }


        return finalSum;

    }

}
```

CPP Solution:

Traverse all numbers in the array one by one and check if any number has exactly 4 divisors . If any number exists add the sum of factors of those numbers into the final result. Do this for all the numbers.

<u>CODE:</u>

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

using namespace std;

int getDivisorSum(int num){

    int sum = 0;
    int count = 0;
    int sq = sqrt(num);
    for (int i = 1; i <= sq; i++){
        if (num % i == 0){
            sum += i;
            count++;
            if (i*i != num){
                sum += (num/i);
                count++;
            }
        }
        if (count > 4){
            break;
        }
    }
    if (count == 4){
        return sum;
    } else {
        return 0;
    }
}

int sumFourDivisors(vector<int>& nums) {

    int sum = 0;
    for (auto& num : nums){
        sum += getDivisorSum(num);
    }
    return sum;
}

int main() {

    int n;
    cin>>n;
    vector<int> v(n);
    for(int i=0;i<n;i++){
        cin>>v[i];
    }
    cout<<sumFourDivisors(v);
}
```

Time Complexity : O(N sqrt(m)).

Fractional Problem

JAVA Solution: (https://leetcode.com/problems/simplified-fractions/discuss/659712/JAVA-or-Clean-Code-or-GCD-Method)

```java
class Solution {
        private int gcd(int a, int b) {
                if(a == 0) return b;
                return gcd(b % a, a);
        }

        public List<String> simplifiedFractions(int n) {
                List<String> ans = new LinkedList<>();
                for(int i = 1; i < n; i++)
                        for(int j = i + 1; j <= n; j++)
                                if(gcd(i, j) == 1) ans.add(i + "/" + j);
                return ans;
        }
}
```

CPP Solution:

Using gcd we can approach the problem

```
class Solution {

    private int gcd(int a, int b) {
        if(a == 0) return b;
        return gcd(b % a, a);
    }

    public List<String> simplifiedFractions(int n) {

        List<String> ans = new LinkedList<>();
        for(int i = 1; i < n; i++)
            for(int j = i + 1; j <= n; j++)
                if(gcd(i, j) == 1) ans.add(i + "/" + j);
        return ans;
    }
}
```

Counting Ending 0’s

JAVA Solution: (https://leetcode.com/problems/factorial-trailing-zeroes/discuss/659737/JAVA-or-Clean-Code-Solution-or-Easy-To-Understand)

```java
class Solution {
    public int trailingZeroes(int n) {
        int tailingZeroes = 0;
        while(n >= 5) {
            n /= 5;
            tailingZeroes += n;
        }
        return tailingZeroes;
    }
}
```

CPP Solution:

```cpp
class Solution {
public:
    int trailingZeroes(int n) {
        long long cnt=0,i=5;
        while(i<=n) {cnt+=n/i; i*=5;}
        return cnt;
    }
};
```

Fun With Divisors

Given a positive integer value **N**. The task is to find how many numbers **less than or equal to N** have numbers of divisors exactly equal to **3**.

Input:
The first line contains integer T, denoting number of test cases. Then T test cases follow. The only line of each test case contains an integer N.

Output:
For each testcase, in a new line, print the answer of each test case.

Your Task:
This is a function problem. You only need to complete the function **exactly3Divisors()** that takes **N** as parameter and **returns** count of numbers **less than or equal to N** with exactly **3 divisors**.

Constraints :
1 <= T <= 100
1 <= N <= 10^9

Example:
Input :
3
6
10
30
Output :
1
2
3

Explanation:
Testcase 1: There is only one number 4 which has exactly three divisors 1, 2 and 4.
Testcase 2: 4 and 9 are the only two numbers less than or equal to 10 that have exactly three divisors.
Testcase 3: 4, 9, 25 are the only numbers less than or equal to 30 that have exactly three divisors.

JAVA Solution:

```
class Divisors
{

    public int exactly3Divisors(int N)
    {
        boolean[] prime = new boolean[N + 1];
        prime[0] = true;
        prime[1] = true;
        for(int i = 2; i * i <= N; i++)
                  if(!prime[i])
                           for(int j = i + i; j <= N; j += i)
                                    prime[j] = true;

        int count = 0;
        for(int i = 2; i * i <= N; i++) if(!prime[i]) ++count;
        return count;
    }
}
```

CPP Solution:

```
#include <bits/stdc++.h>
using namespace std;

// Generates all primes upto n and prints their squares
void numbersWith3Divisors(int n)
{
    bool prime[n+1];
    memset(prime, true, sizeof(prime));
    prime[0] = prime[1] = 0;

    for (int p=2; p*p<=n; p++)
    {
        // If prime[p] is not changed, then it is a prime
        if (prime[p] == true)
        {
           // Update all multiples of p
           for (int i=p*2; i<=n; i += p)
              prime[i] = false;
        }
    }

    // print squares of primes upto n.
    cout << "Numbers with 3 divisors :\n";
    for (int i=0;  i*i <= n ; i++)
        if (prime[i])
          cout << i*i << " ";
}

// driver program
int main()
{
    // sieve();
    int n = 96;
    numbersWith3Divisors(n);

    return 0;
}
```