Problem Solving: Combinatorics

Problem-01: Only Pluses

(Problem Source: https://codeforces.com/problemset/problem/1992/A)

Kmes has written three integers a, b and c in order to remember that he has to give Noobish_Monk a×b×c bananas. Noobish_Monk has found these integers and decided to do the following **at most** 5 **times**:

- pick one of these integers;
- increase it by 1.

For example, if a=2, b=3 and c=4, then one can increase a three times by one and increase b two times. After that a=5, b=5, c=4. Then the total number of bananas will be 5×5×4=100.

What is the maximum value of a×b×c Noobish_Monk can achieve with these operations?

Problem-01 (Cont.)

Input

Each test contains multiple test cases. The first line of input contains a single integer t ($1 \le t \le 1000$) — the number of test cases. The description of the test cases follows.

The first and only line of each test case contains three integers a, b and c ($1 \le a, b, c \le 10$) — Kmes's integers.

Output

For each test case, output a single integer — the maximum amount of bananas Noobish_Monk can get.

Example

Input

2	**Output**
2 3 4	100
10 1 10	600

The Solution

Input:
- An integer t → number of test cases
- For each test case: three integers a, b, c

Output:
- For each test case, the product a * b * c after incrementing the smallest number 5 times

Steps
1. **Read** the number of test cases t.
2. **Repeat** the following for each test case:
    1. **Read** three integers a, b, c.
    2. **Repeat 5 times**:
        - If a is less than or equal to both b and c, increment a by 1.
        - Else if b is less than or equal to both a and c, increment b by 1.
        - Else increment c by 1.
    3. **Compute** the product a * b * c.
    4. **Print** the result.

```
#include<iostream>
using namespace std;
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        int a,b,c;
        cin>>a>>b>>c;
        for(int i=0;i<5;i++)
        {
            if(a<=b && a<=c)
                a++;
            else if(b<=c && b<=a)
                b++;
            else
                c++;
        }
        cout<<a*b*c;
    }
}
```

What will be its Complexity?

Problem-02: trailing zeroes in the factorial of n

Given an integer n, determine the number of trailing zeroes in the factorial of n (n!).

A trailing zero is defined as a zero that appears at the end of a number, after the last non-zero digit. For example, 1200 has two trailing zeroes.

You must design an efficient algorithm that avoids direct computation of n!, since factorial values grow extremely large for even moderate values of n. Instead, focus on analyzing the factors that contribute to trailing zeroes.

Problem-02 (Cont.)

Input

A single integer n ($1 \leq n \leq 10^9$)

Output

An integer representing the number of trailing zeroes in n!

Example

Input	**Output**
10	2

Explanation: 10!=3,628,800, which has **2 trailing zeroes**.

The Solution

A factorial, denoted as n!, is the product of all positive integers from 1 to n. Factorials grow very quickly, and for larger values of n, the result often ends with several trailing zeroes (zeros at the end of the number).

For example:
5!=120 → has 1 trailing zero
10!=3,628,800 → has 2 trailing zeroes

Why do trailing zeroes appear?

- A trailing zero is produced when a number is divisible by 10.
- 10=2×5.
- In factorials, there are always more factors of 2 than 5.
- So, the number of trailing zeroes depends only on the number of times 5 divides the numbers from 1 to n.

Formula:

Trailing Zeroes in $n! = \lfloor n/5 \rfloor + \lfloor n/25 \rfloor + \lfloor n/125 \rfloor + \ldots$
We keep dividing n by 5 until it becomes 0.

Algorithm: Trailing Zeroes in Factorial

Input: An integer n
Output: Number of trailing zeroes in n!

1. **Initialize** count = 0.
2. **Repeat until n becomes 0**:
   - Divide n by 5 (integer division).
   - Add the quotient to count.
   - Update n = n / 5.

3. **Return** count.

Complexity?

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

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;

    int count = 0;

    while (n > 0) {

        n = n / 5;
         count = count+n;
}

    cout << "Trailing zeroes in factorial = " << count << endl;

    return 0;
}
```

Problem-03: count digits in a factorial

Given an integer n, find the number of digits that appear in its factorial, where factorial is defined as, factorial(n) = 1*2*3*4........*n and factorial(0) = 1

Example:

Input: *5*
Output: *3*
Explanation: *5! = 120, that has, 3 digits*

Input: *10*
Output: *7*
Explanation: *10! = 3628800, that has, 7 digits*

The Solution

Using logarithmic property:

We know, log(a*b) = log(a) + log(b)

Therefore:
log(n!) = log(1*2*3....... * n) = log(1) + log(2) + ... +log(n)

Now, observe that the floor value of log base 10 increased by 1, of any number, gives the number of digits present in that number.

Hence, output would be : floor(log(n!)) + 1.

Algorithm: Counting Digits in a Factorial

Input: An integer n
Output: Number of decimal digits in n!

1. **Handle base cases:**
   If n = 0 or n = 1, return 1 (since 0! = 1! = 1).

2. **Initialize sum:**
   Let sum = 0.0 (a floating-point variable to store the sum of logarithms).

3. **Compute log10 of factorial:**
   1. For each integer i from 2 to n:
      1. Add log10(i) to sum.

4. Calculate number of digits: digits = floor(sum) + 1
5. Return digits

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
   int n;
   cin >> n;

   if (n == 0 || n == 1)
   {
      cout << 1 << endl;
       return 0;
   }

   double sum = 0;
   for (int i = 2; i <= n; i++)
  {
      sum = sum + log10(i);
  }
   int digits = floor(sum) + 1;
   cout << digits << endl;
   return 0;
}
```

Complexity?