Binary_seach_with_answer.cpp

```cpp
 /**
*    author:  devesh95
*
**/
#include <bits/stdc++.h>
using namespace std;

#define int             long long int
#define double          long double
#define F               first
#define S               second
#define pb              push_back
#define lb              lower_bound
#define ub              upper_bound
#define si              set <int>
#define vi              vector <int>
#define vvi             vector <vi>
#define pii             pair <int, int>
#define vpi             vector <pii>
#define mii             map <int, int>
#define sz(v)           ((int) v.size())
#define form(i, a, b)   for (int i=a; i<(b); i++)
#define forn(i, a)      for (int i=0; i<(a); i++)

////////////////////////////////////////////////////////
// Binary Search with Answer - Quick Notes
//
// Purpose: Solve optimization problems by finding the smallest or
//          largest value satisfying a condition.
//
// ------------------------------------------------------
// Problem Statement:
// Given n rectangles of size a x b, find the side length of
// the smallest square that can contain all rectangles.
//
// ------------------------------------------------------
// Steps:
// 1. Define the `Good` function:
//    - `Good(x)` checks if all rectangles can fit in a square of side `x`.
//
// 2. Binary Search for the answer:
//    - Initialize `l` as a bad value (e.g., 0).
//    - Initialize `r` as a good value, doubling until `Good(r)` is true.
//    - While `l + 1 < r`, calculate `m` as the midpoint of `l` and `r`.
//      a. If `Good(m)` is true, set `r = m`.
//      b. Otherwise, set `l = m`.
//
// 3. Result:
//    - `r` contains the smallest good value satisfying the condition.
//
// ------------------------------------------------------
```

```cpp
// Applications:
// - Optimization problems with a monotonic condition.
// - Finding minimum/maximum values in geometric or numeric setups.
//
// Tips:i) l is bad and r is good always.
//      ii) Don't forget to set precision when working with floating-point.
////////////////////////////////////////////////////////

// Function to solve the problem of finding the smallest square side length
void solve_smallest_square() {
    int a, b, n;
    cin >> a >> b >> n;

    // Define the Good function
    auto Good = [&](int m)->bool {
        if ((m / a) * (m / b) >= n) return true;
        return false;
    };

    int l = 0, r = 1; // l is bad and r is good always
    while (!Good(r)) r *= 2;

    while (l + 1 < r) {
        int m = l + (r - l) / 2;
        if (Good(m)) r = m;
        else l = m;
    }

    cout << r << '\n';
}

// Function to solve the problem of finding the maximum rope length
// Problem Statement:
// There are n ropes, you need to cut k pieces of the same length from them.
// Find the maximum length of pieces you can get.
void solve_rope_problem() {
    int n, k;
    cin >> n >> k;
    vi v(n);
    forn(i, n) cin >> v[i];

    // Define the Good function
    auto Good = [&](double m)->bool {
        int cnt = 0;
        forn(i, n) {
            cnt += (double)v[i] / m;
        }
        if (cnt >= k) return true;
        return false;
    };

    double l = 0, r = 0.0000001;
    while (Good(r)) r *= 2.0;

```

```cpp
    while (r - l > 0.0000001) {
        double m = l + (r - l) / 2.0;
        if (Good(m)) l = m;
        else r = m;
    }
/*  Another safe approach :
    double l = 0, r = 1e8;
    forn(i, 100) {
        double m = (l + r) / 2;
        if (Good(m))l = m;
        else r = m;
    }                       */
    cout << setprecision(20) << l << '\n';
}

int32_t main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
    clock_t z = clock();
    int t = 1;
    //cin >> t;
    while (t--) {
        solve_smallest_square();
        solve_rope_problem();
    }
    cerr << "Run Time : " << ((double)(clock() - z) / CLOCKS_PER_SEC);
    return 0;
}

```