

Problem Tutorial: "Leftmost Occurrences"

Here you can use binary search from the standard library of your language or write it manually. Here is the example of implementation:

```
int l = 0, r = n - 1;
while (r - l > 1) {
    int mid = (l + r) / 2;
    if (a[mid] < b)
        l = mid;
    else
        r = mid;
}
int result = -1;
for (int i = r; i >= l; i--)
    if (a[i] == b)
        result = i + 1;
```

```
auto i = lower_bound(a.begin(), a.end(), x);
if (i != a.end() || *i == x)
    return -1;
else
    return i - a.begin();
```

Problem Tutorial: "Rightmost Occurrences"

Here you can use binary search from the standard library of your language or write it manually. Here is the example of implementation:

```
int l = 0, r = n - 1;
while (r - l > 1) {
    int mid = (l + r) / 2;
    if (a[mid] <= b)
        l = mid;
    else
        r = mid;
}
int result = -1;
for (int i = l; i <= r; i++)
    if (a[i] == b)
        result = i + 1;
```

```
auto i = upper_bound(a.begin(), a.end(), x);
if (i == a.begin())
    return -1;
i--;
if (*i == x)
    return i - a.begin();
else
    return -1;
```

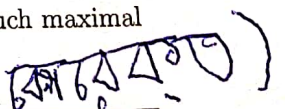
Problem Tutorial: "Difference with Closest"

Here you can use binary search from the standard library of your language or write it manually. You can try the smallest element greater or equal than a value (lower bound) and previous element. Choose one with the smallest distance (skip it if it doesn't exist). Here is the example code

```
auto j = lower_bound(a.begin(), a.end(), b);
int d = INT_MAX;
if (j != a.end())
    d = min(d, abs(*j - b));
if (j != a.begin()) {
    j--;
    d = min(d, abs(*j - b));
}
cout << d << endl;
```

Problem Tutorial: "Threads"

Let $f(x)$ = maximal number of pieces to cut if their length is x . Now the problem is: find such maximal x that $f(x) \geq k$. Easy to see, that $f(x)$ monotonically decreases.

My code is 

Use binary search to find such x where $f(x) = a_1/x + a_2/x + \dots + a_n/x$.

Problem Tutorial: "Root of Tricky Equation"

It is easy to see that $f(x) = 4x + 8\ln(x+1) + x \cdot \ln(x+1)$ is an increasing function (for $x > 0$). So use binary search to find the answer. The code is:

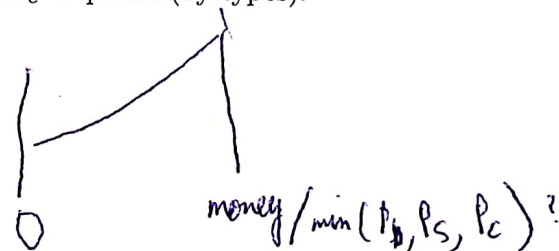
```
double l = 0;
double r = 1E20;
for (int i = 0; i < 200; i++) {
    double x = (l + r) / 2;
    if (4 * x + 8 * log(x + 1) + x * log(x + 1) < c)
        l = x;
    else
        r = x;
}
printf("%.5f\n", (l + r) / 2);
```

Problem Tutorial: "Hamburgers"

Let's use binary search approach. For given number of hamburgers (say, x) let's find the minimal number of money needed to cook them. Say, for one hamburger Polycarpus needs c_b bread pieces, c_s sausages pieces, c_c cheese pieces. So for x hamburgers he needs: $c_b \cdot x$, $c_s \cdot x$ and $c_c \cdot x$ pieces (by types).

Since he already has n_b , n_s and n_c pieces, so he needs to buy:

- bread: $\max(0, c_b \cdot x - n_b)$,
- sausages: $\max(0, c_s \cdot x - n_s)$,
- cheese: $\max(0, c_c \cdot x - n_c)$.



So the formula to calculate money to cook x hamburgers is:

$$f(x) = \max(0, c_b \cdot x - n_b) \overset{\text{remove negative values}}{p_b} + \max(0, c_s \cdot x - n_s) p_s + \max(0, c_c \cdot x - n_c) p_c$$

Obviously, the function $f(x)$ is monotonic (increasing). So it is possible to use binary search approach to find largest x such that $f(x) \leq r$.

Problem Tutorial: "K-th Element (Matrix Edition)"

Just binary search the answer. Let's sort a and b . Such sortings do not change the answer since it is just permutations of rows and columns. If we have function $cntless(x)$ returning number of cells in the matrix strictly less than x , then the answer is $x - 1$, where x is such minimal value that $cntless(x) \geq k$. So use binary search to find such x .

You can also use a binary to implement $cntless(x)$. For each row i you need to find the number of such j that $a_i + b_j < x$. It is exactly index in b of the first value greater or equal than $x - a_i$.

The complete code is:

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

using namespace std;

#define forn(i, n) for (int i = 0; i < int(n); i++)
```



```
int n, m;
vector<long long> a, b;

int cntless(long long x) {
    int result = 0;
    forn(i, n)
        result += lower_bound(b.begin(), b.end(), x - a[i]) - b.begin();
    return result;
}

int main() {
    int k;
    cin >> n >> m >> k;
    a = vector<long long>(n);
    forn(i, n)
        cin >> a[i];
    b = vector<long long>(m);
    forn(j, m)
        cin >> b[j];
    sort(a.begin(), a.end());
    sort(b.begin(), b.end());
    long long l = -2000000001;
    long long r = +2000000001;
    while (r - l > 1) {
        long long mid = (l + r) / 2;
        if (cntless(mid) < k)
            l = mid;
        else
            r = mid;
    }
    long long pos(0LL);
    for (long long i = r; i >= l; i--)
        if (cntless(i) >= k)
            pos = i;
    cout << pos - 1 << endl;
}
```

Handwritten notes:
→ increasing
→ calculates # of values strictly less than x

Problem Tutorial: "Annuity Payment Scheme"

Binary search the answer in floating-point numbers. For fixed value x you can simulate the process. You can pay the credit if you can pay p percents on each step and on some step your current credit is negative or zero. It means you can decrease x (i.e. assign $r = mid$). In opposite case, you should increase x ($l = mid$).

The complete code is:

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

using namespace std;

int main() {
    double s, p;
    int m;
    cin >> s >> m >> p;
    double l = 0.0, r = 1E20;
```

```
for (int i = 0; i < 200; i++) {  
    double x = (l + r) / 2.0;  
    double ss = s;  
    bool can = false;  
    for (int j = 0; j < m; j++) {  
        double d = ss * p / 100.0;  
        if (d > x)  
            break;  
        ss -= (x - d);  
        if (ss < 0) {  
            can = true;  
            break;  
        }  
    }  
    if (can)  
        r = x;  
    else  
        l = x;  
}  
printf("%.10lf\n", (l + r) / 2);  
}
```

Problem Tutorial: "The Meeting Place Cannot Be Changed"

We will apply binary search to solve this problem. Inside the binary search we have to check if it is possible to meet within t seconds. In this time, i -th friend can get anywhere within the segment $[x_i - tv_i, x_i + tv_i]$. For the meeting to be possible, there must be a point common to all these segments, that is, their intersection must be non-empty.

An easy way to intersect a number of segments $[l_1, r_1], \dots, [l_n, r_n]$ is to compute $L = \max l_i$ and $R = \min r_i$. If $L \leq R$, then $[L, R]$ is the intersection, otherwise, the intersection is empty.

Complexity: $O(n \log(\epsilon^{-1}))$ time and $O(n)$ memory. Here ϵ is the required relative precision.

do binary search on time $[0, 10^9]$

200 iterations

Problem Tutorial: "Bars" : *idea of inverting the problem*

Let's binary search the answer. Initially, the left border of lf is 0, and the right border of rg is $n - 2$, since Polycarp necessarily eats bars at the first and last minute and the break cannot be more than $n - 2$.

Let's construct a function $f(x)$ — is it possible to eat bars in such way, that the longest break is at most x ? You can use greedy approach to eat bars in such a way, that the delays are less or equal than x . Just do day by day and each a bar if you can't avoid it without breaking the main property (i.e. that the longest break is at most x).

It is easy to see that function $f(x)$ is monotonic, so use binary search to find such smallest x that $f(x)$ is true.

Problem Tutorial: "Warehouse Position"

It is easy to see that the total penalty is a convex function, since each addend is a convex function. Indeed, d , $d \cdot \ln(d + 1)$, $d \cdot \sqrt{d}$ and d^2 are convex function. So their sum is also a convex function.

Just ternary search the answer to minimize the penalty function.

Problem Tutorial: "Minimum Value of Polynomial"

A polynomial is a very smooth function (especially, if its degree and coefficients are small). Split search area into 1000 pieces of equal size. On each piece the function is convex with high probability. So on each

guess → use time limit to do this

piece use independent ternary search to find the minimum. After it return the global minimum taking the minimal value among minimums on pieces.

```
const int T = 987;
double result = 1E100;
for (int i = 0; i < T; i++) {
    double L = l + (r - l) / T * tt;
    double R = l + (r - l) / T * (tt + 1);
    for (int j = 0; j < 200; j++) {
        double m1 = L + (R - L) * 0.45;
        double m2 = R - (R - L) * 0.45;
        if (p(m1) < p(m2))
            R = m2;
        else
            L = m1;
    }
    result = min(result, p((L + R) / 2.0));
}
printf("%.10f\n", result);
```