A: Biscuit Generator

$T+0.5$ 秒後までに生産される回数は、T を A で割った商 $\lfloor \frac{T}{A} \rfloor$ (T を A で割った切り捨て) です。これは、例えば次のような $A=3$ のときの表を書いてみると分かりやすいかもしれません。

T	0	1	2	3	4	5	6	7	8	9	$\cdots$
答え	0	0	0	B	B	B	$2B$	$2B$	$2B$	$3B$	$\cdots$

1 回ごとに B 個生産されるので、答えは $\lfloor \frac{T}{A} \rfloor \times B$ になります。C++ での実装例を示します。

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

using namespace std;

int main() {
  int A, B, T; cin >> A >> B >> T;
  int generated_times = T / A;
  int ans = generated_times * B;
  cout << ans << endl;
}
```

B: Resale

i 番目の宝石を選ぶことによって $X-Y$ は V_i-C_i だけ変化します。したがって、コストよりも価値が高いような宝石のみを手に入れるのが最適です。

実装では、はじめに全ての入力を受け取ってから処理する方法をとるのが良いでしょう。なぜなら、V_1 を知ったとしても C_1 を知る前に $V_2, V_3, ..., V_N$ を読み込む必要があるためです。$V_1, V_2, ..., V_N$ は配列に保存しておくと後から値を知ることができます。C++ での実装例を挙げます。

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

using namespace std;

int main() {
  // input
  int N; cin >> N;
  vector<int> V(N);
  for (int i = 0; i < N; ++i) {
    cin >> V[i];
  }
  vector<int> C(N);
  for (int i = 0; i < N; ++i) {
    cin >> C[i];
  }
  // compute
  int ans = 0;
  for (int i = 0; i < N; ++i) {
    if (V[i] > C[i]) {
      ans += V[i] - C[i];
    }
  }
  // output
  cout << ans << endl;
}
```

C: GCD on Blackboard

方針としては、A_i を書き換えることにした場合の最大公約数の最大値 M_i を全て求めることを高速に行い、M_i の最大値を出力します。

まず、A_i を書き換える場合、残りの $N-1$ 個の最大公約数に書き換えれば、「整数 A_i を書き換える」操作は「整数 A_i を消す」操作と等価になります。

さらに、整数 X と 整数 Y の最大公約数を $gcd(X,Y)$ と表すことすると、gcd は結合則が成り立ちます。すなわち、任意の整数 X,Y,Z について $gcd(gcd(X,Y),Z) = gcd(X,gcd(Y,Z))$ です。簡単に言えば、gcd には「どこから計算しても結果は変わらない性質」があります[*1](例えば、整数や行列同士の加算や乗算もそうですが、減算は前から計算しなければなりません) このとき、次のテクニックが使えます。$i = 1,2,...,N+1$ について、$L(i), R(i)$ を次のように定義します。

$$L(i) := A_1, A_2, ..., A_{i-1}\text{の最大公約数}$$
$$R(i) := A_i, A_{i+1}, ..., A_N\text{の最大公約数}$$

便宜上、$L(0) = R(N+1) = 0$ とし、全ての整数 X に対して $gcd(0,X) = gcd(X,0) = X$ とすることとします。すると、A_i を書き換える (すなわち、A_i を消す) ことにした場合、残りの $N-1$ 個の整数の最大公約数 M_i は $M_i = gcd(L(i), R(i+1))$ になります。さらに、$L(i), R(i)$ は次の漸化式を計算することで高速に $O(NlogA_1 + NlogA_N)$ で求められます[*2]。

$$\begin{aligned} L(0) &= 0 \\ L(i+1) &= gcd(L(i), A(i)) \\ R(N+1) &= 0 \\ R(i) &= gcd(R(i+1), A(i)) \end{aligned}$$

$M_1, M_2, ..., M_N$ が求まれば、答えはこれらの最大値 $\max(M_1, M_2, ..., M_N)$ になります。全体として計算量は $O(NlogA_1 + NlogA_N)$ となります。

[*1] gcd については「計算順序を入れ替えて良い」という交換則も成り立ちますが、今回は必要ありません。

[*2] $gcd(A,B)$ はユークリッドの互除法により $O(log\min(A,B))$ で求められます

D: Flipping Signs

かっこいい解法

考察 1(重要). この問題のポイントは、「i を選んで A_i, A_{i+1} をそれぞれ $-A_i, -A_{i+1}$ にする」操作を繰り返すことで、「x, y $(x < y)$ を選んで A_x, A_y をそれぞれ $-A_x, -A_y$ にする」ことがどの x, y についても可能であるということです。実際、$i = x, x+1, ..., y-1$ と選んで $y-x$ 回操作を行えば A_x, A_y の符号のみが反転します。

考察 2. 操作を通して、A_i は A_i または $-A_i$ にしかなり得ません。

考察 3. $A_i = 0$ となる i が存在しない場合、操作によって $A_i < 0$ となる i の数の偶奇は変化しません。

したがって、初期状態で負の数が偶数個の場合、または $A_i = 0$ となる i が存在する場合、全てを非負にできるため、$S = |A_1| + |A_2| + ... + |A_N|$ とすると、答えは S になります。そうでない場合、全てを非負にはできませんが、好きな 1 つを除いて全てを非負にできます。負のままにしておく数は、絶対値が最小のものを選ぶのが最適なので、答えは $S - 2 \times \min\{|A_1|, |A_2|, ..., |A_N|\}$ になります。時間計算量は $O(N)$ なので十分間に合います。

アルゴリズムで殴る解法

実は、このような性質を見つけなくとも動的計画法で殴ることができます。同じ i を選んで反転することは無駄なので、各 i について高々 1 回反転することを左から順に考えていきます。

$i \in \{0, 1, ..., N\}, j \in \{0, 1\}$ に対して $dp(i, j)$ を、$A_1, A_2, ..., A_i$ を確定し、$j = 1$ なら i を選んで反転した場合 (すなわち A_{i+1} の符号が反転している状態) の $A_1 + A_2 + ... + A_i$ の最大値と定義します。すると、次の漸化式を計算することで各 i, j に対する $dp(i, j)$ を求めることができます。

$$
\begin{aligned}
dp(0, 0) &= 0 \\
dp(0, 1) &= -\infty \text{ (invalid)} \\
dp(i+1, 0) &= \max(dp(i, 0) + A_i, dp(i, 1) - A_i) \\
dp(i+1, 1) &= \max(dp(i, 0) - A_i, dp(i, 1) + A_i)
\end{aligned}
$$

N を選んで操作することはできないため、答えは $dp(N, 0)$ です。時間計算量はかっこいい解法と同じく $O(N)$ なので十分間に合います。