A ICPC Penalty

判断后直接计算即可，注意 $Accepted$ 时 n 次提交有 $n-1$ 次罚时。

```cpp
void solve() {
    string s;
    int n, t;
    cin >> s >> n >> t;
    if (s == "Accepted") {
        cout << 20 * (n - 1) + t << "\n";
    } else {
        cout << 0 << "\n";
    }
}
```

Fence 1

B 小苯的无限数组

第一次筛去所有奇数；

第二次相当于对第一次的结果除以 2 后筛去所有奇数，剩下 4 的倍数。

以此类推，第 n 次剩下的数字就是 2^n 的倍数。

```cpp
vector<int> p2(11);

void init() {
    p2[0] = 1;
    for (int i = 1; i ≤ 10; ++i) p2[i] = p2[i - 1] + p2[i - 1];
}

void solve() {
    int n, k;
    cin >> n >> k;
    cout << p2[k]*n << "\n";
}
```

Fence 2

当然也可以直接打表找规律

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
2 4 6 8 10 12 14 16 18 20
4 8 12 16 20
8 16
16
```

Fence 3

C 小苯的真假游戏

记 x_i 为第 i 位是否说真话， s_i 为第 i 对第 $(i-1+n)$ 位是否说真话的断言。

- $x_i = 1$ ， $s_i = x_{i-1}$
- $x_i = 0$ ， $s_i = 1 - x_{i-1}$

也就是 $s_i = x_{i-1} \oplus (1 - x_i)$

记 $t_i = s_i \oplus 1$ （取反），如果一个串是合法的，必然有

$x_1 = x_1 \oplus (t_1 \oplus t_2 \oplus \cdots \oplus t_n)$

$t_1 \oplus t_2 \oplus \cdots \oplus t_n = 0$

即只有偶数数个 i 满足 $t_i = 1$ ，也就是字符串只有偶数个 0 。

然后对 x_1 赋值 0 和 1 即可得到唯二的两种合法情况。

```cpp
void solve() {
    int n, cnt = 0;
    string s;
    cin >> n >> s;
    for (auto v : s) cnt += v == '0';
    cout << (cnt & 1 ? 0 : 2) << "\n";
}
```

Fence 4

D 小苯的区间选数2.0

收集所有区间然后贪心放置即可，先用左小长度小的区间。

```cpp
void solve() {
    int n;
    cin >> n;
    vector<PLL> a(n);
    for (int i = 0; i < n; ++i) cin >> a[i].fi >> a[i].se;
    sort(all(a));
    priority_queue<ll> pq;
    int i = 0;
    ll cnt = 0;
    while (1) {
        while (i < n && a[i].fi <= cnt) {
            pq.push(-a[i].se);
            ++i;
        }
        while (!pq.empty() && pq.top() < cnt) pq.pop();
        if (pq.empty()) break;
        pq.pop();
        ++cnt;
        if (cnt > n) break;
    }
    cout << cnt << "\n";
}
```

Fence 5

E 小苯的GCD疑问1.0

猜一个全选就是最优解，举个例子：

$$3+3+3+6+6+6 \leq 3+4+5+6+7+8$$

```cpp
void solve() {
    ll l, r, k;
    cin >> l >> r >> k;
    ll n = r - l + 1;
    if (n == 1) {
        cout << 0 << "\n";
        return;
    }
    cout << (l + r) * n / 2 - r << "\n";
}
```

Fence 6

当然也可以暴力，因为涉及一些编译优化，所以将完整代码放到题解最后。

F 小苯的GCD疑问2.0

枚举 gcd 即可，感觉比 E 简单。

```cpp
void solve() {
    ll n, k;
    cin >> n >> k;
    ll g = 1, ans = 0;
    while (g <= n / k) {
        ll m = n / g;
        ll ng = min(n / m, n / k);
        ans = max(ans, ng * ng * m * (m + 1) / 2);
        g = ng + 1;
    }
    cout << ans << "\n";
}
```

Fence 7

头文件

```cpp
//Another
#include<bits/stdc++.h>
#include<bits/extc++.h>
#define pb push_back
#define eb emplace_back
#define fi first
#define se second
#define all(a) a.begin(), a.end()
#define rall(a) a.rbegin(), a.rend()
using namespace std;

typedef long long ll;
typedef long double ld;
typedef unsigned long long ull;
typedef __int128 i128;
typedef pair<int, int> PII;
typedef pair<ll, ll> PLL;
typedef tuple<ll, ll, ll> TLLL;
typedef __gnu_pbds::tree<PLL, __gnu_pbds::null_type, less<PLL>, __gnu_pbds::rb_tree_tag, __gnu_pbds::tree_order_statistics_node_update> Tree;
// typedef __gnu_pbds::tree<ll, __gnu_pbds::null_type, less<ll>, __gnu_pbds::rb_tree_tag, __gnu_pbds::tree_order_statistics_node_update> Tree;

constexpr int inf = (ll)1e9 + 7;
constexpr ll INF = (ll)2e18 + 9;
// constexpr ll INF = (ll)4e18;
// constexpr ll MOD = 1e9 + 7;
constexpr ll MOD = 998244353;
constexpr ld PI = acos(-1.0);
constexpr ld eps = 1e-10;

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
ull randint(ull l, ull r) {uniform_int_distribution<unsigned long long> dist(l, r); return dist(rng);}
```

```cpp

void init() {

}

void solve() {

}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    init();

    int t = 1;
    // cin >> t;
    for (int _ = 1; _ <= t; ++_) {
        solve();
    }
    return 0;
}
```

Fence 8

E题暴力

```
#include <bits/stdc++.h>
#pragma GCC optimize(3)
#pragma GCC target("avx")
#pragma GCC optimize("Ofast")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-ffast-math")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-funroll-loops")
#pragma GCC optimize("-fwhole-program")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-fstrict-overflow")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-skip-blocks")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
```

```cpp
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("-funsafe-loop-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
#pragma GCC optimize(2)
using namespace std;
using ll = long long;

static const size_t IN_BUF_SZ = 1 << 20;
static char inbuf[IN_BUF_SZ];
static size_t in_pos = 0, in_len = 0;

static inline int getchar_() __attribute__((always_inline));
static inline int getchar_() {
    if (in_pos >= in_len) {
        in_len = fread(inbuf, 1, IN_BUF_SZ, stdin);
        in_pos = 0;
        if (in_len == 0) return EOF;
    }
    return inbuf[in_pos++];
}

static inline ll rd() __attribute__((always_inline));
static inline ll rd() {
    int c = getchar_();
    while (__builtin_expect((c != '-') && (c < '0' || c > '9'), 1)) {
        c = getchar_();
        if (c == EOF) return 0;
    }
    int neg = 0;
    if (c == '-') { neg = 1; c = getchar_(); }
    ll x = 0;
    while (c >= '0' && c <= '9') {
        x = x * 10 + (c - '0');
        c = getchar_();
    }
    return neg ? -x : x;
}

static const size_t OUT_BUF_SZ = 1 << 20;
static char outbuf[OUT_BUF_SZ];
static size_t out_pos = 0;

static inline void flush() {
    if (out_pos) {
        fwrite(outbuf, 1, out_pos, stdout);
        out_pos = 0;
    }
}

static inline void pt(char c) {
    if (out_pos + 1 >= OUT_BUF_SZ) flush();
    outbuf[out_pos++] = c;
}

static inline void pt(ll x) {
    if (out_pos + 80 >= OUT_BUF_SZ) flush();
    if (x == 0) {
        outbuf[out_pos++] = '0';
        outbuf[out_pos++] = '\n';
        return;
    }
    if (x < 0) {
        outbuf[out_pos++] = '-';
        x = -x;
    }
    char tmp[64];
    int tp = 0;
    while (x > 0) {
        tmp[tp++] = char('0' + (int)(x % 10));
        x /= 10;
    }
    for (int i = tp - 1; i >= 0; --i) outbuf[out_pos++] = tmp[i];
    outbuf[out_pos++] = '\n';
}

int main() {
    int t = (int)rd();
    while (t--) {
        ll l = rd();
        ll r = rd();
        ll k = rd();
        if (k < 2) k = 2;
        if (k > r - l + 1) { pt('0'); pt('\n'); continue; }
        ll limit = r / k;
        if (limit <= 0) { pt('0'); pt('\n'); continue; }

        ll lm = (l > 0 ? l - 1 : 0);
        ll ans = 0;

        for (ll g = 1; g <= limit; ) {
            ll u = r / g;
            ll v = (lm == 0 ? 0 : (lm / g));
            ll r1 = (u == 0 ? r : (r / u));
            ll r2 = (v == 0 ? limit : (lm / v));
            ll ng = r1 < r2 ? r1 : r2;
            if (ng > limit) ng = limit;
            if (ng < g) ng = g;

            ll cnt = u - v;
            if (cnt < k) {
                break;
            }
            ll cand = ng * ng;
            cand *= u + v;
            cand *= (cnt - 1);
            cand /= 2;
            if (cand > ans) ans = cand;

            g = ng + 1;
        }

        pt(ans);
    }
    flush();
    return 0;
}
```

Fence 9