> 由我命制的题的题解和标准程序代码已经全部完成。

写在前面

本次考核主要是考察大家这段时间的学习成果，一级对C++语言和基础算法的掌握情况。同时，由于算法集训队和语法集训队的队伍进度有所区别，为了给大家举办一个相对公平比赛，出题组决定不将算法作为考核的重点，而是更多侧重于对**语法和思维能力**的考察，因此在我命题的7个题中，只有3个题考察了算法（分别是A L N），其他的题主要是考察思维能力和数学基础。但是从结果上看，好像情况不是特别理想（除了E之外好像其他题都没有人完全做对）。原因可能是大家第一次做题目难度乱序的比赛，没有人敢去开题或者说是大家疑似知道我出的题很难？导致我的几个送分题没有把分送出去（QAQ）。总之，希望你能够通过本次测试，发现自己的不足，得到进步，同时感受算法之美xD。希望我出的题能给你带来收获！

概述

题目	A. KNN算法	B. Orange的泡泡堂	E. 一个简单的数学问题	G. Orange讨厌下雪	K. Orange的比特币交易	L. 蕈兽大赛	N. Orange的宝石手链
难度	1600	1300	1300	1400	1500	1500	1400
知识点	二分	猜结论	数学	猜结论	差分，数学结论	DFS	前缀和，逆元， 快速幂

题目难度排序：EB->GN->KL->A

E. 一个简单的数学问题

通过高中的数学知识，我们实际上是要将 $\frac{1}{n}$ 裂项成两个分数的和，只需要通过如下变幻即可：

$$\frac{1}{n}=\frac{n+1}{n(n+1)}=\frac{n}{n(n+1)}+\frac{1}{n(n+1)}=\frac{1}{n+1}+\frac{1}{n(n+1)}$$

因此，$x=n+1, y=n(n+1)$，注意开long long即可。然后需要注意的是，当 $n=1$ 的时候无解。

时间复杂度: $O(1)$

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int T;
    cin >> T;
    while (T--)
    {
        long long n;
        cin >> n;
        if(n == 1) cout << "-1 -1\n";
        else cout << n + 1 << ' ' << n(n + 1) << '\n';
    }
}
```

B. Orange的泡泡堂

本题实际上是一个猜测结论的题，我们每次一开始选择一条较短的边，将他炸开之后，我们就可以使用短边长度个炸弹摧毁所有宝石了，这一定是最优的。

当然，本题需要注意到的是：

- 自己的出生点是没有宝石的，因此如果 $n=1$ 且 $m=1$，则答案为 0.
- 如果 n 或者 m 为 1，则只需要一个炸弹就可以摧毁所有宝石。
- 否则，答案就为 $\min(n,m)+1$

时间复杂度: $O(1)$

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int T;
    cin >> T;
    while (T--)
    {
        int n, m, x, y;
        cin >> n >> m >> x >> y;
        if(n == 1 and m == 1) cout << 0 << '\n';
        else if(n == 1 or m == 1) cout << 1 << '\n';
        else cout << min(n, m) + 1 << '\n';
    }
}
```

G. Orange讨厌下雪

本题实际上是一个诈骗题，通过画出每天积雪量的变化曲线，我们发现，实际上**答案只与第一天的积雪量和最后一天积雪量有关，且答案就为** $a_1 - a_n$ 。我们有一次机会去交换两天的积雪量，一定是去将某一天的积雪量与第一天或者最后一天进行交换，因此我们只要去枚举 a_i 与 a_1 或者 a_n 交换的所有情况取一个最大值即可。形式化的说，我们的答案可以表示成如下公式：

$$ans = \max_{i=1}^{n} \max(a_i - a_n, a_1 - a_i)$$

```cpp
void solve()
{
    int n;
    cin >> n;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];
    int ans = max(a[1] - a[n], a[n] - a[1]);
    for (int i = 1; i <= n; i++)
    {
        ans = max(ans, a[i] - a[n]);
        ans = max(ans, a[1] - a[i]);
    }
    cout << ans << '\n';
}
```

N. Orange的宝石手链

本题实际上是考察快速回答一段区间内所有数的乘积，本质上是一个前缀和的裸题，但是将其中的和改成了前缀乘积。记 ar_i 表示 a_i 的前缀乘积，则每次询问的答案则为 $\frac{ar_r}{ar_{l-1}}$。 同时我们注意到，整个乘法是建立在mod意义下的，因此不存在除法这一个操作，而是使用等价的逆元运算来完成。发现取MOD的数是一个质数，由费马小定理可得，一个数 x 的逆元为 x^{mod-2}，这一个操作可以用 快速幂 完成。注意一下，乘法可能爆int，因此要先转化成longlong进行计算。

时间复杂度:$O(nlogn)$

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

const int mod = 1e9+7;

int quickpow(int a,int k)
{
    int res = 1;
    while(k)
    {
        if(k & 1) res = (long long)res * a % mod;
        a = (long long)a * a % mod;
        k >>= 1;
    }
    return res;
}

int main()
{
    int n, m;
    cin >> n >> m;
    vector<int> ar(n + 1, 1);
    for(int i = 1; i <= n; i ++)
    {
        cin >> ar[i];
        ar[i] = (long long)ar[i] * ar[i - 1] % mod;
    }
    while(m--)
    {
        int l, r;
        cin >> l >> r;
        cout << (long long)ar[r] * quickpow(ar[l - 1], mod - 2) % mod << '\n';
    }
}
```

K. Orange的比特币交易

其实本题也是诈骗题，手玩几组样例可以发现，最优情况一定是**第 i 天买入，第 $i+1$ 天卖出**。

考虑使用差分证明：

我们钦定: $d_i = a_i - a_{i-1}$，特殊的，$d_1 = a_1$。

$$f(j,i) = \frac{a_j - a_i}{j-i} = \frac{\sum_{k=i+1}^{j} d_k}{j-i}$$

不难发现，实际上 $f(j,i)$ 实际上就是在求 d_{i+1} 到 d_j 的平均数，而很容易可以想到，平均数最优的情况一定就是单个数的情况，也就是 $i+1=j$ 的情况。因此，最后，答案就是差分数组的最大值:

$$\max_{i=1}^{j} d_i$$

最后记得跟 0 取较大值（也就是不进行交易）。

```cpp
signed main()
{
    int n;
    cin >> n;
    vector<int> a(n + 1), d(n + 1);
    a[0] = 1e9;
    int ans = 0;
    for(int i = 1; i <= n; i ++)
    {
        cin >> a[i];
        d[i] = a[i] - a[i - 1];
        ans = max(d[i], ans);
    }
    cout << ans << '\n';
    return 0;
}
```

L. 蕈兽大赛

~~本题主要考察贪心，不会真的有人信子吧~~

如果考虑贪心的话，我们可以得出：

1. 如果 1 存在于对局当中，则一定让 1 赢
2. 如果对局的双方同时大于或者同时小于 1，则让他们都变成平局（方便被 1 追上或者防止追上 1）
3. 但是当如果双方一个大于 1 或者 一个小于 1，此时就无法确定到底是让一方赢一方输或者让双方平局更优了。

注意到数据范围：$n \leq 10$ 且 $m \leq 10$，实际上我们完全可以去暴力枚举 **情况3** 的所有情况，对所有情况取最优即可。

> ps:其实本题考虑到是新生题，故意放过了暴力枚举所有情况的做法，即放过了 $O(T3^n)$的做法。好吧本题实际上也是想让大家用这种做法去做（因为考的是DFS）。

参考代码如下（暴力枚举所有情况）：

```
int ans = 10;
int n, m;
vector<pii> v;
vector<int> a;
void dfs(int i, int cnt)
{
    if (cnt == m)
    {
        int rk = 1;
        for(int i = 1; i <= n; i ++) if (a[i] > a[1]) rk++;
        ans = min(rk, ans);
        return;
    }
    auto[x, y] = v[i];
    if (x == 1 || y == 1) // 如果1存在于当前对局中，则移动让1赢
    {
        a[x == 1 ? x : y] += 3;
        dfs(i + 1, cnt + 1);
        a[x == 1 ? x : y] -= 3;
    }
    else
    {
        // 平局
        a[x]++,a[y]++;
        dfs(i + 1, cnt + 1);
        a[x]--,a[y]--;

        // x赢
        a[x] += 3;
        dfs(i + 1, cnt + 1);
        a[x] -= 3;

        // y赢
        a[y] += 3;
        dfs(i + 1, cnt + 1);
        a[y] -= 3;
    }
}

void solve()
{
    ans = 10;
    cin >> n >> m;
    a.resize(n + 1);
    for(int i = 1; i <= n; i ++) cin >> a[i];
    v.resize(m + 1);
    for(int i = 1; i <= n; i ++) cin >> v[i].first >> v[i].second;
    dfs(1, 0);
    cout << ans << '\n';
}
```

A. KNN算法

给定一个数轴和 n 个点构成的点集 S，每次查询一个数轴上的点 p(可能不在 S)中，回答集合 S 中距离 p 第 k 近的点到 p 的距离。

不难发现，我们要去求的距离 p 第 k 近的点(记为 x)的距离，实际上不需要将 x 真正求出来，而是只要去求距离就好了。我们假设这个距离为 D，则说明距离 p 第 k 近的点距离 p 的距离刚好是 D，而距离 p 第 $1\sim k-1$ 的点与 p 的距离都小于 D，而剩下的点的距离都大于 D。**换句话说，在距离 p 不超过 D 的长度内，一定刚好有 k 个点。** 而且，我们很容易能发现，D 是具有单调性的，而且 D 的合法性很好检验，因此实际上，我们可以采用 二分答案 求出 D。

接下来考虑如何设计 check 函数，我们每次要去验证距离 p 不超过 D 的点的个数是否大于等于 k。也就是说，我们要去查询点集 S 中有多少个点落在 $[p-D,p+D]$ 之中。这一步实际上可以使用二分查找完成，我们首先对 S 进行排序，每次去 S 中二分查找 $p-D$ 和 $p+D$，那么落在 $[p-D,p+D]$ 之中的点的数量就是二分查找得到的两个下标之差。

注意一下，二分答案的边界和数据范围开longlong即可。

```cpp
using i64 = long long;
signed main()
{
    cin.tie(0) -> sync_with_stdio(0);
    int n, q;
    cin >> n >> q;
    vector<int> a(n);
    for (int i = 0; i < n; i++)
        cin >> a[i];
    sort(a.begin(), a.end());
    while (q--)
    {
        int x, k;
        cin >> x >> k;
        auto check = [&](i64 mid)
        {
            int lc = lower_bound(a.begin(), a.end(), x - mid) - a.begin();
            int rc = upper_bound(a.begin(), a.end(), x + mid) - a.begin();
            return rc - lc >= k;
        };
        i64 l = 0, r = 2e9;
        while(l < r)
        {
            i64 mid = l + r >> 1ll;
            if(check(mid)) r = mid;
            else l = mid + 1;
        }
        cout << r << '\n';
    }
    return 0;
}
```