目录

Codeforces Round 1132 (Div. 2) 1
A1.15K4JR3B (Easy Version) 1
A2.15K4JR3B (Hard Version) 2
B.Introduction to the Niche Internet 4
C1.Disgusting + GCD (Constructive Version) 7
C2.Disgusting + GCD (Checker Version) 9

Codeforces Round 1132 (Div. 2)

A1.15K4JR3B (Easy Version)

题目描述

—— 你知道吗，如果把「文化公园站」、一个不在东北方向的点，以及地图上的另一个点连起来，你会看起来像个白痴。

—— 不，我不知道。真的吗？

本题是问题的简单版本。两个版本的区别在于：在这个版本中，一次操作恰好把某个点的一个坐标改变 1。

失业的 $*$ Angel 把三个点 A、B、C 放在原点。

你需要恰好执行 n 次如下操作：选择其中一个点，把它的某个坐标增加或减少 1。换句话说，如果被选中的点坐标为 (x, y)，那么一次操作可以将它移动到 $(x+1, y)$、$(x-1, y)$、$(x, y+1)$ 或 $(x, y-1)$。

失业的 $*$ Angel 想知道，执行完所有操作后，三角形 ABC 的二倍面积的最大值。因为她是你的偶像，你无法拒绝她。

由于失业的 $*$ Angel 非常聪明，她可以证明你所关心的这个值总是整数。

输入格式

每个测试包含多组测试数据。第一行包含测试数据组数 t（$1 \le t \le 1000$）。接下来是各组测试数据的描述。

每组测试数据仅有一行，包含一个整数 n（$2 \le n \le 1000$）——操作的次数。

输出格式

对于每组测试数据，输出一个整数——三角形 ABC 的二倍面积的最大值。

样例

1.in / 1.out

```
5
2
5
15
43
769
```

```
1
6
56
462
```

```
5  147840
```

在第一组测试数据中，把 B 的 x 坐标增加 1，再把 C 的 y 坐标增加 1。经过这 2 次操作后，三个点分别为 $A=(0,0)$、$B=(1,0)$、$C=(0,1)$，所得三角形的二倍面积为 1。

在第二组测试数据中，把 A 的 x 坐标减少 1，把 B 的 x 坐标增加两次 1，最后把 C 的 y 坐标增加两次 1。经过这 5 次操作后，三个点分别为 $A=(-1,0)$、$B=(2,0)$、$C=(0,2)$，所得三角形的二倍面积为 6。

题解

status: `00:07 +`

这个题就是很简单的构造呀，复健中……

思考这样一个问题，如果我们让 A 往上走，B 往左走，C 往右走，这和 A 往上走，B 往右走，C 不动的答案是没有区别的。

如果 B 或者 C 往下走，它带来的效果是不如 A 往上走的（思考高的变化）。（这句话不严谨，但直观能感觉到）

所以我们让 A 往上，B 往右，根据基本不等式我们得到 A 走 $\lfloor \frac{n}{2} \rfloor$ 步，C 走 $\lceil \frac{n}{2} \rceil$ 步面积最大，为 $\lfloor \frac{n}{2} \rfloor \cdot \lceil \frac{n}{2} \rceil$

```cpp
#include <cstdio>
#include <stdio.h>

int T;
int n;

int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        int a = n >> 1, b = n - a;
        printf("%d\n", a * b);
    }
    return 0;
}
```

A2.15K4JR3B (Hard Version)

题目描述

—— 你知道吗，如果把「文化公园站」、一个不在东北方向的点，以及地图上的另一个点连起来，你会看起来像个白痴。

—— 不，我不知道。真的吗？

本题是问题的困难版本。两个版本的区别在于：在这个版本中，一次操作把某个点的一个坐标改变 2^{a_i}。

失业的 $*$ Angel 把三个点 A、B、C 放在原点，并给了你一个长度为 n 的数组 a。

你需要恰好执行 n 次操作。第 i 次操作时，选择其中一个点，把它的某个坐标增加或减少 2^{a_i}。换句话说，如果被选中的点坐标为 (x,y)，那么第 i 次操作可以将它移动到 $(x+2^{a_i},y)$、$(x-2^{a_i},y)$、$(x,y+2^{a_i})$ 或 $(x,y-2^{a_i})$。

失业的 * Angel 想知道，执行完所有操作后，三角形 ABC 的二倍面积的最大值。因为她是你的偶像，你无法拒绝她。

由于失业的 * Angel 非常聪明，她可以证明你所关心的这个值总是整数。

输入格式

每个测试包含多组测试数据。第一行包含测试数据组数 t（$1 \le t \le 100$）。接下来是各组测试数据的描述。

每组测试数据的第一行包含一个整数 n（$2 \le n \le 100$）——数组 a 的长度，同时也是操作的次数。

第二行包含 n 个整数 $a_1, a_2, ..., a_n$（$0 \le a_i \le 24$）——数组 a 的元素。

输出格式

对于每组测试数据，输出一个整数——三角形 ABC 的二倍面积的最大值。

样例

1.in / 1.out

```
4
2
0 0
3
0 0 1
6
0 1 2 2 4 0
6
24 24 24 0 0 0
```

```
1
4
192
562950054084608
```

在第一组测试数据中，把 B 的 x 坐标增加 2^0，再把 C 的 y 坐标增加 2^0。经过这 2 次操作后，三个点分别为 $A = (0, 0)$、$B = (1, 0)$、$C = (0, 1)$，所得三角形的二倍面积为 1。

在第二组测试数据中，把 A 的 x 坐标减少 2^0，把 B 的 x 坐标增加 2^0，最后把 C 的 y 坐标增加 2^1。经过这 3 次操作后，三个点分别为 $A = (-1, 0)$、$B = (1, 0)$、$C = (0, 2)$，所得三角形的二倍面积为 4。

题解

status: `00:46 +`

在 A1 的基础上，我们知道要把移动步数拆成两个差尽可能小的量，求它们的乘积。

问题来到了差怎么求。

我第一个思路是枚举值域，想到总的计算量可能到了 10^{11} 量级就放弃了。

然后想的是如果有相同的值就分到两边，剩下的值一定满足 $2 + 2^2 + 2^3 + 2^4 + ... + 2^n < 2^{n+1}$，且是差最小的方案。

然后就被 $[2^{24}, 2^{24}, 2^{24}, 1, 1]$ 卡了，这里两个 1 要分到一侧。

然后就有一个乱搞做法：从大到小排序，哪个少加哪个。

想一想感觉能覆盖到上面的情况，正确性我们现在说明一下。

因为大的元素对值起到决定作用，所以我们这样可以防止两侧值本来相当，但加上一个大的值导致差异很大的情况。

```cpp
#include <stdio.h>
typedef long long i64;

#define maxn 107

int T;
int n, a[maxn];
int f[maxn];
int cnt[maxn], v, m, m1, m2;

int main() {
    scanf("%d", &T);
    while (T--) {
        m = 0;
        m1 = m2 = 0;
        for (int i = 0; i < 30; i++)
            cnt[i] = 0;

        scanf("%d", &n);
        for (int i = 1; i <= n; i++)
            scanf("%d", &v), cnt[v]++;

        for (int i = 30; i >= 0; i--) {
            while (cnt[i]) {
                cnt[i]--;
                if (m1 <= m2)
                    m1 += 1 << i;
                else
                    m2 += 1 << i;
            }
        }

        printf("%lld\n", 1ll * (m1) * (m2));
    }

    return 0;
}
```

B.Introduction to the Niche Internet

题目描述

在两个月内没能成为 grandmaster 之后，Nikolai 对竞技编程的职业生涯彻底失望，决定转行去做政治直播。

众所周知，没有心爱观众的支持，主播就无法生存，因此 Nikolai 的每一场直播都必然会收到一定数量的打赏，它们构成一个序列 a。注意，有很多黑客在看 Nikolai 的直播，所以有些打赏可能是负数。

Nikolai 认为第 m 次打赏非常令人愉快，当且仅当它的金额不小于此前所有打赏的平均值，即

$$a_m \cdot (m-1) \geq \sum_{i=1}^{m-1} a_i.$$

特别地，第一次打赏总是非常令人愉快的。

如果一场直播中 Nikolai 恰好收到 k 次非常令人愉快的打赏，他就认为这场直播回本了。

Nikolai 知道今天他会收到来自观众的 n 次打赏，但他不知道它们的顺序。请帮助观众们选择打赏的发送顺序，以取悦 Nikolai，让这场直播回本！

输入格式

每个测试包含多组测试数据。第一行包含测试数据组数 t（$1 \leq t \leq 10^4$）。接下来是各组测试数据的描述。

每组测试数据的第一行包含两个整数 n 和 k（$1 \leq k \leq n \leq 2 \cdot 10^5$）——打赏的次数与所需的非常令人愉快的打赏次数。

第二行包含 n 个整数 $a_1, a_2, ..., a_n$（$-10^9 \leq a_i \leq 10^9$）——每次打赏的金额。

保证所有测试数据中 n 之和不超过 $2 \cdot 10^5$。

输出格式

如果不存在合适的打赏发送顺序，输出 `NO`。

否则，第一行输出 `YES`，第二行输出 n 个整数：按发送顺序给出的打赏金额。如果有多组解，输出任意一组均可。

输出的大小写不限（例如 `yEs`、`yes`、`Yes`、`YES` 都会被判为肯定回答）。

样例

1.in / 1.out

```
4
3 1
1 2 3
3 1
5 5 5
3 2
-1 -2 -3
4 3
0 0 10 -10
```

```
YES
3 2 1
NO
YES
-1 -3 -2
YES
10 -10 0 0
```

在第一组测试数据中，一个合适的顺序是 $[3, 2, 1]$。第一次打赏总是非常令人愉快的；第二次不是，因为 $2 \cdot 1 < 3$；第三次也不是，因为 $1 \cdot 2 < 3 + 2$。因此恰好有 1 次非常令人愉快的打赏。

在第二组测试数据中，可以证明不存在合适的顺序。

在第三组测试数据中，一个合适的顺序是 $[-1,-3,-2]$。第一次打赏非常令人愉快；第二次不是，因为 $(-3)\cdot 1 < -1$；第三次是，因为 $(-2)\cdot 2 \geq -1+(-3)$。因此恰好有 2 次非常令人愉快的打赏。

题解

status: `01:42 +2`

这道题倒是自己想到了构造。

如果我们升序排列数列，一定有 $k=n$，这个答案是显然的。

如果我们降序排列数列，只能说明有 $1 \leq k \leq n$，因为我们的要求是**大于等于**前面的均值，意味着如果是相同的数字仍旧是满足要求的。

由于我们要构造 k 个满足的数字，我们不妨取一段长度为 k 的序列升序排列。

剩余的 $n-k$ 个数字，我们从后向前降序排列。

这样构造一定是最可能产生结果的方案，所以我们最后扫一遍看下满足不满足，如果满足就输出。

我的赛时实现有点丑，凑合看吧。

```cpp
#include <algorithm>
#include <cstdio>
#include <stdio.h>

typedef long long i64;
#define maxn 400007

int T, n, k;
int li[maxn], arr[maxn];

int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &k);
        k = n - k; // unpleasent num

        for (int i = 1; i <= n; i++)
            scanf("%d", li + i);

        std::sort(li + 1, li + n + 1);
        std::reverse(li + 1, li + 1 + k);

        for (int i = 1; i <= k; i++)
            li[n + i] = li[i];

        for (int i = 1; i <= n; i++)
            arr[i] = li[i + k];

        int f = 0;
        i64 s = 0;

        for (int i = 1; i <= n; i++) {
```

```
            if (s <= 1ll * arr[i] * (i - 1))
                f++;
            s += arr[i];
        }

        if (f == n - k) {
            puts("YES");
            for (int i = 1; i <= n; i++)
                printf("%d ", arr[i]);
            puts("");
        } else {
            puts("NO");
        }
    }

    return 0;
}
```

C1.Disgusting + GCD (Constructive Version)

题目描述

本题是构造版本。

在这个版本中，你是 kijjot。

定义一个数组 b 是「恶心的」，当且仅当存在下标 $i < j$，使得

$$1 < \gcd(b_i, b_j) < \min(b_i, b_j).$$

这里 $\gcd(x, y)$ 表示整数 x 与 y 的最大公约数。

给定一个长度为 n、由正整数组成的数组 a。为了避免永无止境的合并，你需要让它不再恶心。

为此，你可以执行如下操作：选择下标 $1 \le i \le n$ 和正整数 $1 \le x \le 10^9$，满足 $\gcd(a_i, x) = 1$，并把 a_i 增加 x。注意，该条件是对操作前 a_i 的当前值检验的，而不是它在原数组中的值。

你的体力正在耗尽，因此执行操作时还要满足以下限制：

- 最多执行 $n + 10$ 次操作。
- 所有操作中所选的 x 之和不能超过原数组 a 的元素之和。

请你求出任意一个经过这些操作后得到的合法数组。可以证明，在给定的约束下，这样的数组总是存在。

输入格式

每个测试包含多组测试数据。第一行包含测试数据组数 t（$1 \le t \le 10^4$）。接下来是各组测试数据的描述。

每组测试数据的第一行包含一个整数 n（$2 \le n \le 10^5$）——数组的长度。

第二行包含 n 个整数 $a_1, a_2, \ldots, a_n$（$2 \le a_i \le 10^5$）——数组的元素。

保证所有测试数据中 n 之和不超过 10^5。

输出格式

对于每组测试数据，输出 n 个整数 $b_1, b_2, \ldots, b_n$（$a_i \le b_i \le 10^9$）——经过操作后得到的数组 b。

数组 b 必须不恶心，并且必须能够由 a 通过上述允许的操作得到，同时满足两条限制。如果有多组解，输出任意一组均可。

样例

1.in / 1.out

```
3
2
7 12
3
7 9 12
7
8 76 49 26 24 80 99
```

```
7 12
7 25 12
9 77 50 27 25 81 100
```

在第一组测试数据中，我们可以不执行任何操作，因为数组已经不恶心了。

在第二组测试数据中，数组是恶心的，因为 $1 < \gcd(a_2, a_3) = \gcd(9, 12) = 3 < \min(a_2, a_3) = \min(9, 12) = 9$ 成立。我们可以选择 $i = 2$ 和 $x = 16$，因为 $\gcd(x, a_i) = \gcd(16, 9) = 1$。操作后，数组 a 变为 $[7, 25, 12]$，不再恶心。

题解

status: null

题目等价的意思是：对于任意两个数 $a, b \in A, (a < b)$，要么 $a \perp b$，要么 $a \mid b$。

这道题我本来想的是这样一个性质：

$$\text{let } c = \prod_{i=1}^{n} a_i, \forall i \in [1, n], c \perp a_i$$

但是构造质因子连乘明显乘爆炸，而且不满足我们题目加法的数量限制。

不过这之中有一个相当重要的观察：

$$\gcd(a, b) = \gcd(a, b - a)(b > a)$$

后来发现这个 $a|b$ 可以退化到 $a = b$ 的情况。

然后经过 itshawn 点拨，发现任何一个质数的可重集合都满足上述的性质。

所以，只需要对于每个数字都找到大于等于它的质数就好了。

```cpp
#include <algorithm>
#include <stdio.h>

#define maxn 200007
char check[maxn];
int prime[maxn], cnt, n, q, Tp, li[maxn];

void ola_prime() {
    for (int i = 2; i < maxn; i++) {
        if (!check[i])
```

```
            prime[++cnt] = i;
        for (int j = 1; j <= cnt; j++) {
            if (i * prime[j] >= maxn)
                break;
            check[i * prime[j]] = 1;
            if (!(i % prime[j]))
                break;
        }
    }
    return;
}

int T;
int main() {
    ola_prime();
    scanf("%d", &T);

    while (T--) {
        scanf("%d", &n);
        for (int i = 1; i <= n; i++)
            scanf("%d", li + i);

        for (int i = 1; i <= n; i++)
            printf("%d ", *std::lower_bound(prime + 1, prime + cnt + 1, li[i]));
        puts("");
    }
    return 0;
}
```

C2.Disgusting + GCD (Checker Version)

题目描述

在这个版本中，你需要实现构造版本中 checker 的一部分。

在这个版本中，你是 ogiyo。

kijjot 给了你两个整数 a 和 b，满足 $a \le b$。一次操作中，你可以选择一个正整数 x 满足 $\gcd(a, x) = 1$，然后把 a 增加 x。

你的任务是找到任意一个长度最短的操作序列，把 a 变成 b。可以证明，在给定的约束下，总是可以用至多 15 次操作把 a 变成 b。

这里 $\gcd(x, y)$ 表示整数 x 与 y 的最大公约数。

输入格式

每个测试包含多组测试数据。第一行包含测试数据组数 t（$1 \le t \le 10^4$）。接下来是各组测试数据的描述。

每组测试数据包含两个整数 a 和 b（$2 \le a \le 10^5$，$a \le b \le 10^9$）——kijjot 给你的两个数。

输出格式

对于每组测试数据，第一行输出最少的操作次数 k（$0 \le k \le 15$）。

第二行输出 k 个正整数 $x_1, x_2, ..., x_k$——按顺序给出的每次操作所选的 x。这些操作必须合法，并且能把 a 变成 b。

如果 $k = 0$，可以省略第二行。如果有多组最优解，输出任意一组均可。

样例

1.in / 1.out

```
4
2 2
15 25
2184 2200
100000 1000000000
```

```
0
2
1 9
3
11 2 3
2
1636673 998263327
```

在第一组测试数据中，$a = b = 2$，因此不需要任何操作。

在第二组测试数据中，我们可以先增加 1 得到 16，再增加 9 得到 25。两次操作都合法，因为 $\gcd(15, 1) = 1$ 且 $\gcd(16, 9) = 1$。可以证明，少于 2 次操作是不够的。

题解

status: `null`

题目要求给出构造 a 到 b 的方案，由我们在 C1 里得到的性质，如果让 a 得到一个质数 p，p 再得到 b，那么显然会有 $a \perp p - a, p \perp b - p$。（注：真的吗？再想想……）

但是如果 a 到 b 之间没有质数呢？由某某神秘定理可以得知两个质数之间的最大间隔是 $\log^2 n$ 这个量级的，相当小，直接用暴力的做法就好了。

这里的暴力做法具体指的是：

对于每个点，枚举差值，判断是否互质，记录最短“距离”（即用了几个数）和回溯路径。

但是！如果这道题到这里就完了，还是差了很多东西。

$\gcd(p, b - p) \neq 1$ 的情况是存在的：$p \mid b$ 就是一个反例。

所以如果这个质数用不了，我们就往后挪一个质数。

如果所有范围内的质数都用不了，说明 a 和 b 的差距也不会太大（p 很小，往后跳了几个就大于 b）。

注意边界情况，而且因为要使用最少的数，一定要按 $0, 1, 2, ...$ 的顺序来判断。

请看丑陋的实现：

```cpp
#include <algorithm>
#include <numeric>
#include <stdio.h>
#include <vector>

```

```cpp
#define maxn 200007
char check[maxn];
int prime[maxn], cnt, n, q, Tp, li[maxn];
int f[maxn], rev[maxn], dis[maxn];

void ola_prime() {
    for (int i = 2; i < maxn; i++) {
        if (!check[i])
            prime[++cnt] = i;
        for (int j = 1; j <= cnt; j++) {
            if (i * prime[j] >= maxn)
                break;
            check[i * prime[j]] = 1;
            if (!(i % prime[j]))
                break;
        }
    }
    return;
}

std::vector<int> traces;
void trace(int x) {
    if (x == 0)
        return;
    traces.push_back(rev[x]);
    trace(x - rev[x]);
}

int T;
int main() {
    ola_prime();
    scanf("%d", &T);

    while (T--) {
        int a, b;
        scanf("%d%d", &a, &b);

        if (a == b) {
            puts("0");
            continue;
        }
        if (std::gcd(b - a, a) == 1) {
            printf("1\n%d\n", b - a);
            continue;
        }

        auto ansp = std::upper_bound(prime + 1, prime + cnt + 1, a);
        auto ans = *ansp;
```

```
54
55          int fr = 0;
56          while (ans < b) {
57              if (ans < b and (std::gcd(ans - a, a) == 1) and
58                  (std::gcd(b - ans, b) == 1)) {
59                  fr = 1;
60                  printf("2\n%d %d\n", ans - a, b - ans);
61                  break;
62              }
63              ansp++;
64              ans = *ansp;
65          }
66
67          if (not fr) {
68              f[0] = 1;
69              for (int i = 1; i <= b - a; i++)
70                  f[i] = 0, rev[i] = 0, dis[i] = 100;
71
72              dis[0] = 0;
73              for (int i = 0; i <= b - a; i++)
74                  if (f[i])
75                      for (int d = b - a - i; d >= 1; d--)
76                          if (std::gcd((a + i), d) == 1 and
77                              dis[i] + 1 < dis[i + d])
78
79                              dis[i + d] = dis[i] + 1, rev[i + d] = d,
80                                      f[i + d] = 1;
81
82              traces.clear();
83              trace(b - a);
84
85              printf("%zu\n", traces.size());
86              for (auto it = traces.rbegin(); it != traces.rend(); it++)
87                  printf("%d ", *it);
88              puts("");
89          }
90      }
91      return 0;
92  }
```