Homework 1

A. Closest pair

给定 n 个二维欧几里得平面上的点 $p_1, p_2, \ldots, p_n$，请输出距离最近的两个点的距离。

Input

请从 stdin 读入。

输入第一行为一个正整数 $n (2 \leq n \leq 4 \times 10^5)$，表示点数。

接下来 n 行，第 i 行为用空格隔开的整数 $x_i, y_i (-10^7 \leq x_i, y_i \leq 10^7)$，表示 $p_i = (x_i, y_i)$。

输入保证：没有两个坐标完全相同的点。

Output

请输出到 stdout 中。

输出一行，包含一个整数 D^2，表示距离最近的两个点的距离的**平方**。

由于输入的点为整点，因此这个值一定是整数。

Sample Input

```
2
-10000000 -10000000
10000000 10000000
```

```
5
1 1
1 9
9 1
9 9
0 10
```

Sample Output

```
800000000000000
```

```
2
```

Constraints

Time Limit: 2s

Memory Limit: 128MB

Note

对于第二组样例，$(1,9)(0,10)$ 两个点最近，距离为 $\sqrt{2}$，因此你需要输出 2。

本题参考 Python 实现如下。非常遗憾，我们的 OJ 不支持提交 Python。

请注意：为了实现简单，这份代码的时间复杂度是 $O(n\log^2 n)$ 的。

```
class point:
    def __init__(self, x:int, y:int):
        self.x = x
        self.y = y
    def distance_2(self, other) -> int:
        return (self.x - other.x) ** 2 + (self.y - other.y) ** 2

def solve(a, l, r) -> int: # the square of closest distance between index range
[l, r)
    if l + 1 >= r:
        return 2 ** 64 # = INFINITY
    m = (l + r) // 2
    ret = min(solve(a, l, m), solve(a, m, r))
    strip = []
    for i in range(l, r):
        if (a[i].x - a[m].x) ** 2 < ret:
            strip.append(a[i])
    strip.sort(key=lambda p : p.y)
    for i in range(len(strip)):
        for j in range(i + 1, len(strip)):
            if (strip[i].y - strip[j].y) ** 2 >= ret:
                break
            ret = min(ret, strip[i].distance_2(strip[j]))
    return ret

n = int(input())
a = []
for i in range(n):
    x, y = map(int, input().split())
    a.append(point(x, y))
a.sort(key=lambda p : p.x)
print(solve(a, 0, n))
```

B. k-th Smallest Number

给定 n 个正整数，请找出其中的第 k 小的数。

输入可能有重复数字，此时第 k 小的值定义为唯一的 x，满足

$$(|\{y|y<x\}|<k)\wedge(|\{y|y\geq x\}|\geq n-k),$$

也即将整个序列从小到大排序后的第 k 个数。

Input

由于输入可能很大，本题采用奇怪的方式读入。你可以直接使用这段代码完成读入。

请注意，输入存储在 $a[1\ldots n]$ 里，0 不存内容。

其中，$1\leq k\leq n\leq 4\times 10^7$，$0\leq a_i<2^{31}$。

```
const int N = 4e7 + 1;
int n, k;
int a[N];
void read_input_data() {
    int m;
    cin >> n >> k >> m;
    for (int i = 1; i <= m; i++) {
        cin >> a[i];
    }
    unsigned int z = a[m];
    for (int i = m + 1; i <= n; i++) {
        z ^= z << 13;
        z ^= z >> 17;
        z ^= z << 5;
        a[i] = z & 0x7fffffff;
    }
}
```

Output

请输出到 stdout 中。

输出一行，包含一个整数，为你的答案。

Sample Input

```
3 3 3
2 3 3
```

```
5 4 1
1919810
```

Sample Output

```
3
```

```
737192472
```

Constraints

Time Limit: 1s

Memory Limit: 512MB

Note

第二组样例实际上代表的数是 $[1919810, 132030712, 737192472, 1757748577, 642384501]$。

输入格式解释：

输入第一行，三个正整数。输入第二行 m 个空格隔开的整数，表示 $a_1, \ldots, a_m$。

$a_{m+1}, \ldots, a_n$ 使用 xorshift 随机生成器生成，

$$a_{m+i} = z_i \bmod 2^{31},$$

其中，

$$
\begin{aligned}
z_0 &= a_m \\
x_i &= z_{i-1} \text{ xor } (z_{i-1} \times 2^{13}) && (\bmod 2^{32}) \\
y_i &= x_i \text{ xor } \left\lfloor \frac{x_i}{2^{17}} \right\rfloor && (\bmod 2^{32}) \\
z_i &= y_{i-1} \text{ xor } (y_{i-1} \times 2^5) && (\bmod 2^{32})
\end{aligned}
$$

C. Bubbling bubbles

给定一个长度为 n 的排列，元素标号为 $1 \ldots n$。

如果对这个排列进行冒泡排序，那么每个元素会被交换若干次。

请输出每个元素在进行冒泡排序时，参与了多少次交换。

我们将以以下代码为标准计算交换次数。

```
void bubble_sort(int a[], int n) {
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < n - i; j++) {
            if (a[j] > a[j + 1]) {
                swap(a[j], a[j + 1]);
            }
        } // after i-th inner iteration, a[n - i] is correct
    }
}
```

Input

请从 stdin 读入。

输入第一行一个正整数 $n(1 \leq n \leq 10^6)$ 。

第二行包含 n 个空格隔开整数表示排列，第 i 个数为排列第 i 位 $a_i(1 \leq a_i \leq n)$。输入保证 n 个数互不相同。

Output

请输出到 stdout 中。

输出一行 n 个由空格隔开的数，第 i 个数表示 i 被交换了多少次。

Sample Input

```
4
1 2 3 4
```

```
4
4 3 2 1
```

```
3
2 3 1
```

Sample Output

```
0 0 0 0
```

```
3 3 3 3
```

```
2 1 1
```

Constraints

Time Limit: 1s

Memory Limit: 512MB

D. Optimal sort

有一个 $1 \dots n$ 的排列，你需要对他进行排序。你无法直接获得他们的大小，仅能通过比较来得到大小关系。

由于比较排序的复杂度有 $\Omega(n \log n)$ 的下界，为了使得你的程序非常优秀，你只能进行至多

$$n \lceil \log_2(n) \rceil - (n-1)$$

次询问。

How do we judge your answer

本题是交互题。你仅需实现 `void optimal_sort(int n)` 函数，表示对一个长度为 n 的排列 $\{p_1, \dots, p_n\}$ 进行排序。

你的代码前需要声明函数原型 `int query(int, int);`。

在 `optimal_sort` 中，

1. 可以调用 `int query(int x, int y)` 表示比较标号为 x, y 的大小。返回值为 $\{-1, 0, 1\}$ 中的一个数，表示 $p_x < p_y, p_x = p_y$ 以及 $p_x > p_y$。
2. 确信答案正确后，请往 stdout 中输出一行由 n 个空格隔开的整数，表示排名为 i 的数在 p 中的位置下标是多少，并退出函数。

如果你的函数进行了第 $n \lceil \log_2(n) \rceil - n + 2$ 次询问，你会得到 `Wrong Answer`，并在 info 里可以看到 `operation limit exceeded`。

如果你的输出错误，你会得到 `Wrong Answer`。

如果你进行了非法的操作，你可能得到 `Wrong Answer`，并在 info 里看到 `invalid query`。

Constraints

本题有 10 组测试数据，每组 10 分，n 取值分别为 $2, 4, 8, \dots, 1024 = 2^{10}$。 评测器使用的排列不会因为你的回答而改变。

对于每组测试，我们会调用你的程序至多 100 次，请记得在每次调用时清空所用到的全局变量。

Time Limit: **3s for 100 cases**

Memory Limit: 128MB

Sample Interaction

```
n = 2, hidden array = [2, 1]
call optimal_sort(2):
    query(1, 2), returned 1
exit optimal_sort
n = 2, hidden array = [1, 2]
call optimal_sort(2):
    query(1, 2), returned -1
exit optimal_sort
```

期望输出：

```
2 1
1 2
```

Note

你可以使用以下程序来测试你的代码，方法是将你的代码将这段代码粘贴到你的代码最后。这段代码不会对你的行为进行任何检查。

请注意，提交的代码里不应该有 `main` 函数。

```
#include <iostream>
int __qcnt = 0;
int __hidden_array[1024];
int query (int x, int y) {
    __qcnt ++;
    int d = __hidden_array[x] - __hidden_array[y];
    return d > 0 ? 1 : (d == 0 ? 0 : -1);
}
int main() {
    int n;
    std::cin >> n;
    for (int i = 1; i <= n; i++) std::cin >> __hidden_array[i];
    optimal_sort(n);
    std::cout << "number of queries : " << __qcnt << std::endl;
}
```

以下代码是针对 $n = 2$ 的参考代码实现，你可以从中了解提交的格式。

```
#include <iostream>
using namespace std;
int a[1024 + 1];
int query(int x, int y);
void solve(int l, int r) {
    if (l >= r) return;
    if (r - l > 1) {
        cout << "Sorry, " << r - l + 1 << " is too difficult for me!" <<
std::endl;
        return;
    }
    if (query(a[l], a[r]) == 1) {
        swap(a[l], a[r]);
    }
```

```
}
void optimal_sort(int n) {
    for (int i = 1; i <= n; i++) a[i] = i;
    solve(1, n);
    for (int i = 1; i <= n; i++) printf("%d%c", a[i], i == n ? '\n' : ' ');
}
```

E. Probabilistic killer

随机化大师博先生认为 Closest pair 一题非常简单，因此他随手设计了如下算法来通过这个题：

将所有点按随机的角度绕原点旋转并按照 x 轴排序后，计算相邻两个点的距离。

为了防止运气太差，博先生的程序实际上会从初始角开始的半平面内均匀尝试 T 次：

1. 均匀随机地从 $[0,\pi)$ 中选择一个角度 θ_0，作为初始角；
2. 重复 T 次，将所有点从初始坐标绕原点逆时针旋转 $\theta_k=\theta_0+\frac{k-1}{T}\pi$ 并按照 x 轴排序，计算相邻两个点之间的距离。
3. 输出 T 次计算结果中出现过的距离最小值。

霏老师觉得博先生的做法是搞笑的，并请来了你构造数据卡掉博先生的做法。

Input

输入一个数，$T(1\le T\le 400)$，表示博先生的程序此时采用的参数。

Output

你需要按照 Closest pair 一题的输入格式输出一组数据。

输出第一行为一个正整数 $n(2\le n\le 4\times 10^5)$，表示点数。

接下来 n 行，第 i 行为用空格隔开的整数 $x_i,y_i(-10^7\le x_i,y_i\le 10^7)$，表示 $p_i=(x_i,y_i)$。

你的输出中，不应当包含两个坐标完全相同的点。

How do we judge your answer

本题有 20 组测试点，每组测试点 5 分。

对于第 i 组数据，

1. $T=i^2$；
2. 你输出的 n 应当满足： $2\le n\le \lfloor 4\times 10^5/T\rfloor$。$i=20$ 时，$n\le 1000$。

对于你的输出，我们将采用如下方法进行测试：

1. 首先使用正确的最近点对算法计算正确答案。
2. 固定一个隐藏的随机种子 SEED_i，使用随机数生成器生成 20 个初始角度运行博先生的程序。
3. 如果你成功让博先生的程序输出错误至少 2 次，你就赢了。

作为参考，运行一次 $T=1$ 的正确率降低至 $1/500$ 时，你有 >0.99 的概率通过最后一组测试数据。

Constraints

Time Limit: 3s

Memory Limit: 128MB

Sample Input

```
1
```

Sample Output

```
3
0 0
0 1
0 2
```

Note

很难不发现样例输出（除了格式）是完全错误的：由于 $(1,2)$ $(2,3)$ 的距离均为 1，因此所有排列均存在相邻两个点距离为 1。评测器会非常肯定地拒绝掉这个输入。

请注意，尽管概率测度是 0，实际中仍可能存在旋转后 x 坐标相同的点对，此时排序算法可能返回任意顺序。

你可以参考这份代码来了解博先生算法的细节，其中 `T` 和 `SEED` 将在评测时被更改。

```
#include <bits/stdc++.h>
using namespace std;
const int T = 10;
const int SEED = 114514;
const int N = 4e5;
const long double pi = acosl(-1);
int n, X[N], Y[N];
struct point {
    long double x, y;
    int i;
    bool operator < (const point &a) const {
        return x < a.x;
    }
    long long distance_2(const point &a) {
        return (long long)(X[a.i] - X[i]) * (X[a.i] - X[i]) + (long long)(Y[a.i]
- Y[i]) * (Y[a.i] - Y[i]);
    }
};
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) cin >> X[i] >> Y[i];
    long long ans = LLONG_MAX;
    mt19937 rng(SEED);
    uniform_real_distribution <long double> distribution(0, pi);
    long double theta = distribution(rng);
    for (int i = 0; i < T; i++) {
        vector < point > a;
        long double angle = theta + pi * i / T;
        for (int j = 0; j < n; j++) {
            long double x = X[j], y = Y[j];
            long double nx = cos(angle) * x + sin(angle) * y, ny = -sin(angle) *
x + cos(angle) * y;
            a.push_back({nx, ny, j});
        }
```

```
        sort(a.begin(), a.end());
        for (int j = 1; j < n; j++) {
            ans = min(ans, a[j - 1].distance_2(a[j]));
        }
    }
    cout << ans << endl;
}
```

One more thing ...

你能用一个错误的做法冲过 Closest pair 吗?

我们准备了很强的数据，欢迎尝试!