E. Two Platforms

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

There are nn points on a plane. The ii-th point has coordinates (xi,yi)(xi,yi). You have two horizontal platforms, both of length kk. Each platform can be placed anywhere on a plane but it should be placed **horizontally** (on the same yy-coordinate) and have **integer borders**. If the left border of the platform is (x,y)(x,y) then the right border is (x+k,y)(x+k,y) and all points between borders (including borders) belong to the platform.

Note that platforms can share common points (overlap) and it is not necessary to place both platforms on the same yy-coordinate.

When you place both platforms on a plane, all points start falling down decreasing their yy-coordinate. If a point collides with some platform at some moment, the point stops and is **saved**. Points which never collide with any platform are lost.

Your task is to find the maximum number of points you can **save** if you place both platforms optimally.

You have to answer tt independent test cases.

For better understanding, please read the **Note** section below to see a picture for the first test case.

**Input**

The first line of the input contains one integer tt (1≤t≤2⋅1041≤t≤2⋅104) — the number of test cases. Then tt test cases follow.

The first line of the test case contains two integers nn and kk (1≤n≤2⋅1051≤n≤2⋅105; 1≤k≤1091≤k≤109) — the number of points and the length of each platform, respectively. The second line of the test case contains nn integers x1,x2,…,xnx1,x2,…,xn (1≤xi≤1091≤xi≤109), where xixi is xx-coordinate of the ii-th point. The third line of the input contains nn integers y1,y2,…,yny1,y2,…,yn (1≤yi≤1091≤yi≤109), where yiyi is yy-coordinate of the ii-th point. All points are distinct (there is no pair 1≤i<j≤n1≤i<j≤n such that xi=xjxi=xj and yi=yjyi=yj).

It is guaranteed that the sum of nn does not exceed 2⋅1052⋅105 (∑n≤2⋅105∑n≤2⋅105).

**Output**

For each test case, print the answer: the maximum number of points you can save if you place both platforms optimally.

**Example**

**input**

**Copy**

4

7 1

1 5 2 3 1 5 4

1 3 6 7 2 5 4

1 1

1000000000

1000000000

5 10

10 7 5 15 8

20 199 192 219 1904

10 10

15 19 8 17 20 10 9 2 10 19

12 13 6 17 1 14 7 9 19 3

**output**

**Copy**

6

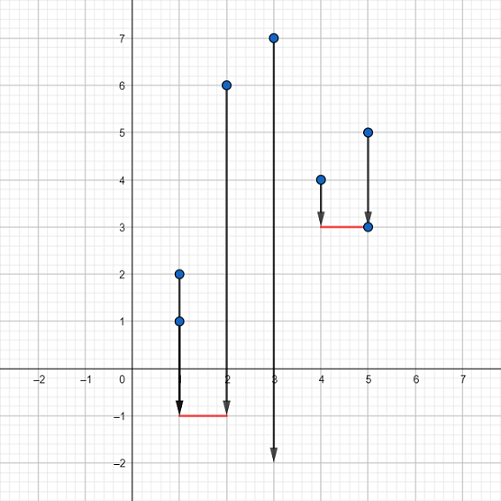
1

5

10

**Note**

The picture corresponding to the first test case of the example:



Blue dots represent the points, red segments represent the platforms. One of the possible ways is to place the first platform between points (1,−1)(1,−1) and (2,−1)(2,−1) and the second one between points (4,3)(4,3) and (5,3)(5,3). Vectors represent how the points will fall down. As you can see, the only point we can't save is the point (3,7)(3,7) so it falls down infinitely and will be lost. It can be proven that we can't achieve better answer here. Also note that the point (5,3)(5,3) doesn't fall at all because it is already on the platform.

Idea: **[vovuh](https://codeforces.com/profile/vovuh" \o "Master vovuh)**

Tutorial

[1409E - Two Platforms](https://codeforces.com/contest/1409/problem/E)

Firstly, we obviously don't need yy-coordinates at all because we can place both platforms at y=−∞y=−∞. Let's sort all xx-coordinates in non-decreasing order.

Calculate for each point ii two values lili and riri, where lili is the number of points to the left from the point ii (including ii) that are not further than kk from the ii-th point (i.e. the number of such points jj that |xi−xj|≤k|xi−xj|≤k). And riri is the number of points to the right from the point ii (including ii) that are not further than kk from the ii-th point. Both these parts can be done in O(n)O(n) using two pointers.

Then let's build suffix maximum array on rr and prefix maximum array on ll. For ll, just iterate over all ii from 22 to nn and do li:=max(li,li−1)li:=max(li,li−1). For rr, just iterate over all ii from n−1n−1 to 11 and do ri:=max(ri,ri+1)ri:=max(ri,ri+1).

The question is: what? What did we do? We did the following thing: the answer always can be represented as two non-intersecting segments of length kk such that at least one endpoint of each segment is some input point (**except the case** n=1n=1). Now, let's fix this border between segments. Iterate over all ii from 11 to n−1n−1 and update the answer with max(li,ri+1)max(li,ri+1). So we took **some** segment that starts at some point to the left from ii (including ii) and goes to the left and took some segment that starts further than i+1i+1 (including i+1i+1) and goes to the right. With this model, we considered all optimal answers that can exist.

Time complexity: O(nlogn)O(nlog⁡n).

Solution

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

**using** **namespace** std;

**int** main() {

**#ifdef** \_DEBUG

freopen(**"input.txt"**, **"r"**, stdin);

**// freopen("output.txt", "w", stdout);**

**#endif**

**int** t;

cin >> t;

**while** (t--) {

**int** n, k;

cin >> n >> k;

vector**<int>** x(n), y(n);

**for** (**auto** &it : x) cin >> it;

**for** (**auto** &it : y) cin >> it;

sort(x.**begin**(), x.**end**());

**int** j = n - 1;

vector**<int>** l(n), r(n);

**for** (**int** i = n - 1; i >= 0; --i) {

**while** (x[j] - x[i] > k) --j;

r[i] = j - i + 1;

**if** (i + 1 < n) r[i] = max(r[i], r[i + 1]);

}

j = 0;

**for** (**int** i = 0; i < n; ++i) {

**while** (x[i] - x[j] > k) ++j;

l[i] = i - j + 1;

**if** (i > 0) l[i] = max(l[i], l[i - 1]);

}

**int** ans = 1;

**for** (**int** i = 0; i < n - 1; ++i) {

ans = max(ans, r[i + 1] + l[i]);

}

cout << ans << endl;

}

**return** 0;

}