Located on a line are N segments, numbered from 0 to N − 1, whose positions are given in zero-indexed arrays A and B. For each I (0 ≤ I < N) the position of segment I is from A[I] to B[I] (inclusive). The segments are sorted by their ends, which means that B[K] ≤ B[K + 1] for K such that 0 ≤ K < N − 1.

Two segments I and J, such that I ≠ J, are *overlapping* if they share at least one common point. In other words, A[I] ≤ A[J] ≤ B[I] or A[J] ≤ A[I] ≤ B[J].

We say that the set of segments is *non-overlapping* if it contains no two overlapping segments. The goal is to find the size of a non-overlapping set containing the maximal number of segments.

For example, consider arrays A, B such that:

A[0] = 1 B[0] = 5

A[1] = 3 B[1] = 6

A[2] = 7 B[2] = 8

A[3] = 9 B[3] = 9

A[4] = 9 B[4] = 10

The segments are shown in the figure below.

https://codility-frontend-prod.s3.amazonaws.com/media/task_img/max_nonoverlapping_segments/media/auto/mp0184a7d365d4ab553d45451d43cec2c9.png

The size of a non-overlapping set containing a maximal number of segments is 3. For example, possible sets are {0, 2, 3}, {0, 2, 4}, {1, 2, 3} or {1, 2, 4}. There is no non-overlapping set with four segments.

Write a function:

class Solution { public int solution(int[] A, int[] B); }

that, given two zero-indexed arrays A and B consisting of N integers, returns the size of a non-overlapping set containing a maximal number of segments.

For example, given arrays A, B shown above, the function should return 3, as explained above.

Assume that:

* N is an integer within the range [0..30,000];
* each element of arrays A, B is an integer within the range [0..1,000,000,000];
* A[I] ≤ B[I], for each I (0 ≤ I < N);
* B[K] ≤ B[K + 1], for each K (0 ≤ K < N − 1).

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.