AMBITIOUS

NailingPlanks

START

Count the minimum number of nails that allow a series of planks to be nailed.

Programming language: C++

You are given two non-empty zero-indexed arrays A and B consisting of N integers. These arrays represent N planks. More precisely, A[K] is the start and B[K] the end of the K−th plank.

Next, you are given a non-empty zero-indexed array C consisting of M integers. This array represents M nails. More precisely, C[I] is the position where you can hammer in the I−th nail.

We say that a plank (A[K], B[K]) is nailed if there exists a nail C[I] such that A[K] ≤ C[I] ≤ B[K].

The goal is to find the minimum number of nails that must be used until all the planks are nailed. In other words, you should find a value J such that all planks will be nailed after using only the first J nails. More precisely, for every plank (A[K], B[K]) such that 0 ≤ K < N, there should exist a nail C[I] such that I < J and A[K] ≤ C[I] ≤ B[K].

For example, given arrays A, B such that:

```
A[0] = 1    B[0] = 4
A[1] = 4    B[1] = 5
A[2] = 5    B[2] = 9
A[3] = 8    B[3] = 10
```

four planks are represented: [1, 4], [4, 5], [5, 9] and [8, 10].

Given array C such that:

```
C[0] = 4
C[1] = 6
C[2] = 7
C[3] = 10
C[4] = 2
```

if we use the following nails:

- 0, then planks [1, 4] and [4, 5] will both be nailed.
- 0, 1, then planks [1, 4], [4, 5] and [5, 9] will be nailed.
- 0, 1, 2, then planks [1, 4], [4, 5] and [5, 9] will be nailed.
- 0, 1, 2, 3, then all the planks will be nailed.

Thus, four is the minimum number of nails that, used sequentially, allow all the planks to be nailed.

Write a function:

```
int solution(vector<int> &A, vector<int> &B,
vector<int> &C);
```

that, given two non-empty zero-indexed arrays A and B consisting of N integers and a non-empty zero-indexed array C consisting of M integers, returns the minimum number of nails that, used sequentially, allow all the planks to be nailed.

If it is not possible to nail all the planks, the function should return −1.

For example, given arrays A, B, C such that:

```
A[0] = 1    B[0] = 4
A[1] = 4    B[1] = 5
A[2] = 5    B[2] = 9
A[3] = 8    B[3] = 10

C[0] = 4
C[1] = 6
C[2] = 7
C[3] = 10
C[4] = 2
```

the function should return 4, as explained above.

Assume that:

- N and M are integers within the range [1..30,000];
- each element of arrays A, B, C is an integer within the range [1..2*M];
- A[K] ≤ B[K].

Complexity:

- expected worst-case time complexity is O((N+M)*log(M));
- expected worst-case space complexity is O(M), beyond input storage (not counting the storage

Future training

required for input arguments).

Elements of input arrays can be modified.