Codility_

CodeCheck Report: training7KHH9X-WKP

Test Name:

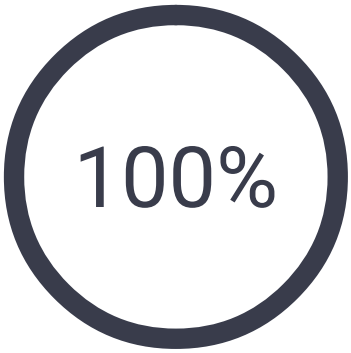
Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
NailingPlanks Java 8	30 min	100%

Total score



Tasks Details

Medium

1. NailingPlanks

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

Task Score	Correctness	Performance
100%	100%	100%

Task description

You are given two non-empty 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 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].

Solution

Programming language used:	Java 8
Total time used:	30 minutes
Effective time used:	30 minutes
Notes:	*not defined yet*

Task timeline

05:46:50 06:16:34

Code: 06:16:33 UTC, java, final, score: **100** show code in pop-up

```
1  // you can also use imports, for example:
2  // import java.util.*;
3
```

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:

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

that, given two non-empty arrays A and B consisting of N integers and a non-empty 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.

Write an **efficient** algorithm for the following assumptions:

- 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].


```
4  // you can write to stdout for debugging purposes,
5  // System.out.println("this is a debug message");
6  import java.util.Arrays;
7  class Solution {
8      public int solution(int[] A, int[] B, int[] C)
9          int N = A.length;
10         int M = C.length;
11         int[][] sortedNail = new int[M][2];
12         for (int i = 0; i < M; i++) {
13             sortedNail[i][0] = C[i];
14             sortedNail[i][1] = i;
15         }
16         Arrays.sort(sortedNail, (int x[], int y[])
17         int result = 0;
18         for (int i = 0; i < N; i++) {
19             result = getMinIndex(A[i], B[i], sorte
20             if (result == -1)
21                 return -1;
22         }
23         return result + 1;
24     }
25     public int getMinIndex(int startPlank, int end
26         int min = 0;
27         int max = nail.length - 1;
28         int minIndex = -1;
29         while (min <= max) {
30             int mid = (min + max) / 2;
31             if (nail[mid][0] < startPlank)
32                 min = mid + 1;
33             else if (nail[mid][0] > endPlank)
34                 max = mid - 1;
35             else {
36                 max = mid - 1;
37                 minIndex = mid;
38             }
39         }
40         if (minIndex == -1)
41             return -1;
42         int minIndexOrigin = nail[minIndex][1];
43         for (int i = minIndex; i < nail.length; i+
44             if (nail[i][0] > endPlank)
45                 break;
46             minIndexOrigin = Math.min(minIndexOrig
47             if (minIndexOrigin <= preIndex)
48                 return preIndex;
49         }
50         return minIndexOrigin;
51     }
52 }
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O((N + M) * log(M))**

expand all	Example tests	
▶	example example test	✓ OK
expand all	Correctness tests	
▶	extreme_single single nail and single plank	✓ OK
▶		

extreme_point nail is a point [1, 1]	✓ OK	
▶ few_nails_in_the_same_place few nails are in the same place	✓ OK	
▶ random_small random sequence, length = ~100	✓ OK	

expand all Performance tests

▶ random_medium random sequence, length = ~10,000	✓ OK	
▶ random_large random sequence, length = ~30,000	✓ OK	
▶ extreme_large_planks all large planks, length = ~30,000	✓ OK	
▶ large_point all planks are points, length = ~30,000	✓ OK	