Codility_

CodeCheck Report: training7BZJH3-32W

Test Name:

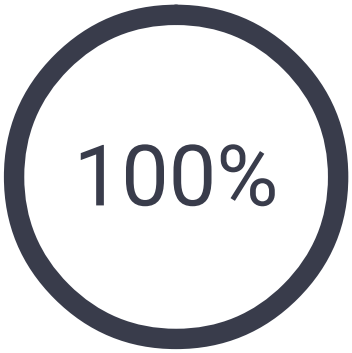
Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
NailingPlanks Java 8	4 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
```

Solution

Programming language used:	Java 8
Total time used:	4 minutes
Effective time used:	4 minutes
Notes:	*not defined yet*

Task timeline

21:45:28 21:48:51

Code: 21:48:51 UTC, java, final, score: **100** show code in pop-up

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

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:

```
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 and C is an integer within the range [1..2∗M];
- A[K] ≤ B[K].


```
3
4   // you can write to stdout for debugging purpo
5   // System.out.println("this is a debug message
6
7   class Solution {
8       public int solution(int[] A, int[] B, int
9           int N = A.length;
10          int M = C.length;
11
12          // two dimension array to save the ori
13          int[][] sortedNails = new int[M][2];
14          for (int i = 0; i < M; i++) {
15              sortedNails[i][0] = C[i]; // nail
16              sortedNails[i][1] = i; // nail inc
17          }
18
19          // sort nails by their positions
20          Arrays.sort(sortedNails, (int[] x, int
21
22          // max index between all min nail indi
23          int minNailIndex = -1;
24
25          for (int k = 0; k < N; k++) {
26              // find min nail index between A[k
27              minNailIndex = findMinNailIndex(so
28              if (minNailIndex == -1) {
29                  return -1;
30              }
31          }
32
33          return minNailIndex + 1;
34      }
35
36      /**
37          * Find the min index of nails that are
38          *
39          * @param sortedNails
40          * @param a
41          * @param b
42          * @return the min index if found, or -
43          */
44      public int findMinNailIndex(int[][] sorted
45          int left = 0;
46          int right = sortedNails.length - 1;
47          int minIndex = -1;
48
49          // search for most left nail that is 1
50          while (left <= right) {
51              int mid = (left + right) / 2;
52              int c = sortedNails[mid][0];
53              if (c < a) {
54                  left = mid + 1;
55              } else if (c > b) {
56                  right = mid - 1;
57              } else {
58                  right = mid - 1;
59                  minIndex = mid;
60
61              }
62          }
63
64          if (minIndex == -1) { // not found
65              return -1;
66          }
67
68          int minIndexOriginal = sortedNails[min
69
70          // testing neighbour nails
71          for (int i = minIndex; i < sortedNails
72              minIndexOriginal = Math.min(minInd
73              if (minIndexOriginal <= currentMin
74                  return currentMinIndex;
75              }
76          }
77
78          return minIndexOriginal;
79      }
80  }
```

Analysis summary

The solution obtained perfect score.

Analysis

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

collapse all — Example tests

▼ example — ✔ OK
example test

1. 0.008 s OK

collapse all — Correctness tests

▼ extreme_single — ✔ OK
single nail and single plank

1. 0.008 s OK
2. 0.004 s OK

▼ extreme_point — ✔ OK
nail is a point [1, 1]

1. 0.004 s OK
2. 0.004 s OK

▼ few_nails_in_the_same_place — ✔ OK
few nails are in the same place

1. 0.004 s OK
2. 0.004 s OK
3. 0.008 s OK

▼ random_small — ✔ OK
random sequence, length = ~100

1. 0.008 s OK

collapse all — Performance tests

▼ random_medium — ✔ OK
random sequence, length = ~10,000

1. 0.120 s OK
2. 0.404 s OK

▼ random_large — ✔ OK
random sequence, length = ~30,000

1. 0.240 s OK

▼ extreme_large_planks — ✔ OK
all large planks, length = ~30,000

1. 0.240 s OK

▼ large_point — ✔ OK
all planks are points, length = ~30,000

1. 0.360 s OK