Codility_

CodeCheck Report: trainingMBB3SR-EC5

Test Name:

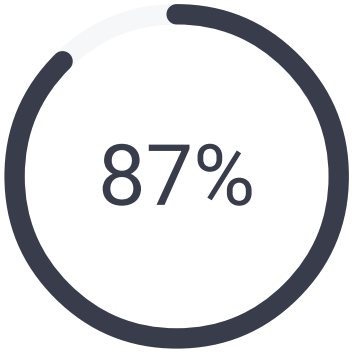
Check out Codility training tasks

Summary Timeline AI Assistant Transcript

Tasks summary

Task	Time spent	Score
NailingPlanks C++	45 min	87%

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
87%	75%	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:	C++
Total time used:	45 minutes
Effective time used:	45 minutes
Notes:	*not defined yet*

Task timeline

12:07:07 12:51:10

Code: 12:51:09 UTC, cpp, final, score: **87** show code in pop-up

```
1  // you can use includes, for example:
2  #include <algorithm>
```

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 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].


```cpp
#include <vector>

// you can write to stdout for debugging purpo
// cout << "this is a debug message" << endl;

bool canNailedAllPlanks(vector<int> &A, vecto
    int plank = A.size();
    int maxPosition = *max_element(C.begin(),

    // 못 위치에 따른 개수 누적합
    vector<int> prefixNails(maxPosition + 1, 0

    for (int i = 0; i < max; i++) { // 못 위치
        prefixNails[C[i]] = 1;
    }

    for (int i = 1; i <= maxPosition; i++) {
        prefixNails[i] += prefixNails[i - 1];
    }

    for (int i = 0; i < plank; i++) { // 누적합
        if (prefixNails[B[i]] - (A[i] > 0 ? pr
            return false;
    }

    return true;
}

int solution(vector<int> &A, vector<int> &B, v
    // 중간값 J를 통해 J개의 못으로 모든 판자를 박을 수
    int left = 1;
    int right = C.size();
    int result = -1;

    // 이진탐색
    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (canNailedAllPlanks(A, B, C, mid))
            result = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    return result;
}
```

Analysis summary

The following issues have been detected: wrong answers.

For example, for the input `([2], [2], [1])` the solution returned a wrong answer (got 1 expected -1).

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		✘ WRONG ANSWER got 1 expected -1

1. 0.001 s **OK**
2. 0.001 s **WRONG ANSWER**, got 1 expected -1

▶	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