Codility_

CodeCheck Report: training387PW5-C2Y

Test Name:

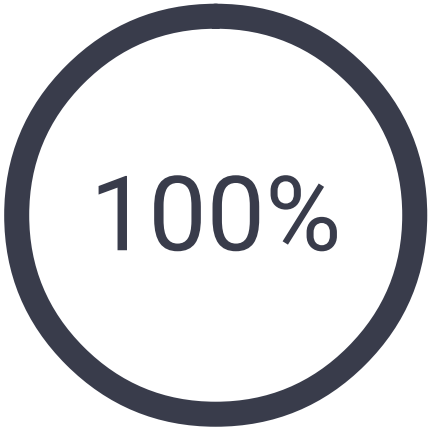
Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
TapeEquilibrium ⚠ Java 8	1 min	100%

Total score



Tasks Details

Easy

1. TapeEquilibrium

Minimize the value |(A[0] + ... + A[P-1]) - (A[P] + ... + A[N-1])|.

Task Score	Correctness	Performance
100%	100%	100%

Task description

A non-empty array A consisting of N integers is given. Array A represents numbers on a tape.

Any integer P, such that 0 < P < N, splits this tape into two non-empty parts: A[0], A[1], ..., A[P − 1] and A[P], A[P + 1], ..., A[N − 1].

The *difference* between the two parts is the value of: |(A[0] + A[1] + ... + A[P − 1]) − (A[P] + A[P + 1] + ... + A[N − 1])|

In other words, it is the absolute difference between the sum of the first part and the sum of the second part.

For example, consider array A such that:

```
A[0] = 3
A[1] = 1
A[2] = 2
A[3] = 4
A[4] = 3
```

We can split this tape in four places:

- P = 1, difference = |3 − 10| = 7
- P = 2, difference = |4 − 9| = 5
- P = 3, difference = |6 − 7| = 1
- P = 4, difference = |10 − 3| = 7

Write a function:

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

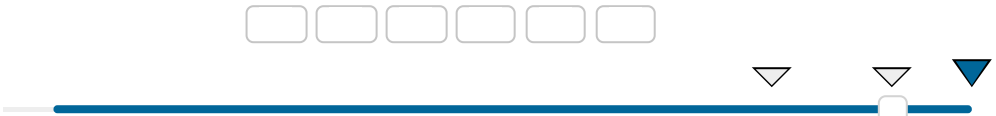
that, given a non-empty array A of N integers, returns the minimal difference that can be achieved.

For example, given:

Solution

Programming language used:	Java 8
Total time used:	1 minutes
Effective time used:	1 minutes
Notes:	*not defined yet*

Task timeline



22:03:03 22:03:51

Code: 22:03:51 UTC, java, final, show code in pop-up
score: **100**

```
1  import java.util.Arrays;
2
3  class Solution {
4      public int solution(int[] A) {
5                  int sum = Arrays.stream(A).sum();
6                  int minDifference = Integer.MAX_VALUE;
7                  int sumLeft = 0;
8                  for (int i = 0; i < A.length - 1; i++) {
9                          sumLeft += A[i];
10                         minDifference = Math.min(minDiffe
11                 }
12                 return minDifference;
13
```

```
A[0] = 3
A[1] = 1
A[2] = 2
A[3] = 4
A[4] = 3
```

the function should return 1, as explained above.

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

- N is an integer within the range [2..100,000];
- each element of array A is an integer within the range [−1,000..1,000].


```
}}
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N)**

collapse all	Example tests	
▼ example example test		✔ OK
1. 0.008 s	OK	

collapse all	Correctness tests	
▼ double two elements		✔ OK
1. 0.008 s	OK	
2. 0.008 s	OK	
3. 0.008 s	OK	
▼ simple_positive simple test with positive numbers, length = 5		✔ OK
1. 0.008 s	OK	
2. 0.012 s	OK	

▼ simple_negative ✔ OK

simple test with negative numbers, length = 5

1. 0.008 s OK
2. 0.008 s OK

▼ simple_boundary ✔ OK

only one element on one of the sides

1. 0.008 s OK
2. 0.008 s OK
3. 0.008 s OK
4. 0.008 s OK

▼ small_random ✔ OK

random small, length = 100

1. 0.008 s OK

▼ small_range ✔ OK

range sequence, length = ~1,000

1. 0.012 s OK

▼ small ✔ OK

small elements

1. 0.008 s OK

collapse all

Performance tests

▼ medium_random1 ✔ OK

random medium, numbers from 0 to 100, length = ~10,000

1. 0.032 s OK

▼

medium_random2 ✔ OK

random medium, numbers from -1,000 to 50, length = ~10,000

1. 0.032 s OK

▼ large_ones ✔ OK

large sequence, numbers from -1 to 1, length = ~100,000

1. 0.224 s OK
2. 0.224 s OK

▼ large_random ✔ OK

random large, length = ~100,000

1. 0.280 s OK
2. 0.280 s OK

▼ large_sequence ✔ OK

large sequence, length = ~100,000

1. 0.128 s OK

▼ large_extreme ✔ OK

large test with maximal and minimal values, length = ~100,000

1. 0.312 s OK
2. 0.312 s OK
3. 0.252 s OK