

Test Name:

Summary Timeline

Tasks summary

Task	Time spent	Score
FrogJump Java 8	32 min	100%

Total score

100%

Tasks Details

Easy

1. FrogJump
Count minimal number of jumps from position X to Y.

Task Score

100%

Correctness

100%

Performance

100%

Task description

A small frog wants to get to the other side of the road. The frog is currently located at position X and wants to get to a position greater than or equal to Y. The small frog always jumps a fixed distance, D.

Count the minimal number of jumps that the small frog must perform to reach its target.

Write a function:

```
class Solution { public int solution(int X, int Y, int D); }
```

that, given three integers X, Y and D, returns the minimal number of jumps from position X to a position equal to or greater than Y.

For example, given:

```
X = 10
Y = 85
D = 30
```

the function should return 3, because the frog will be positioned as follows:

- after the first jump, at position 10 + 30 = 40
- after the second jump, at position 10 + 30 + 30 = 70
- after the third jump, at position 10 + 30 + 30 + 30 = 100

Solution

Programming language used:

Java 8

Total time used:

32 minutes

?

Effective time used:

32 minutes

?

Notes:

not defined yet

Task timeline

17:35:55

18:07:36

Code: 18:07:35 UTC, java, final,
score: 100

[show code in pop-up](#)

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

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

- X, Y and D are integers within the range [1..1,000,000,000];
- $X \leq Y$.

Copyright 2009–2020 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

```
4 // you can write to stdout for debugging purposes, e.g.
5 // System.out.println("this is a debug message");
6
7 class Solution {
8     public int solution(int X, int Y, int D) {
9         // write your code in Java SE 8
10        int hops = 0;
11        if(X >= Y){
12            return hops;
13        }else if((X+D) >= Y){
14            return 1; //1 hops
15        }
16
17        int targetDistance = (Y-X);
18        int divisible = Math.floorDiv(targetDistance,D);
19        int coveredDistance = (divisible * D) + X;
20        int balance = Y - coveredDistance;
21        if( balance == 0){
22            hops = divisible;
23        }else if(balance >0 && balance <= D){
24            hops = divisible + 1;
25        }else{
26            //greater than D. thats not really possible
27            hops = divisible;
28        }
29        return hops;
30    }
31 }
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: O(1)	
expand all	Example tests
▶ example example test	✓ OK
expand all	Correctness tests
▶ simple1 simple test	✓ OK
▶ simple2	✓ OK
▶ extreme_position no jump needed	✓ OK
▶ small_extreme_jump one big jump	✓ OK
expand all	Performance tests
▶ many_jump1 many jumps, D = 2	✓ OK
▶ many_jump2 many jumps, D = 99	✓ OK
▶ many_jump3 many jumps, D = 1283	✓ OK
▶ big_extreme_jump maximal number of jumps	✓ OK
▶ small_jumps many small jumps	✓ OK

The PDF version of this report that may be downloaded on top of this site may contain sensitive data including personal information. For security purposes, we recommend you remove it from your system once reviewed.