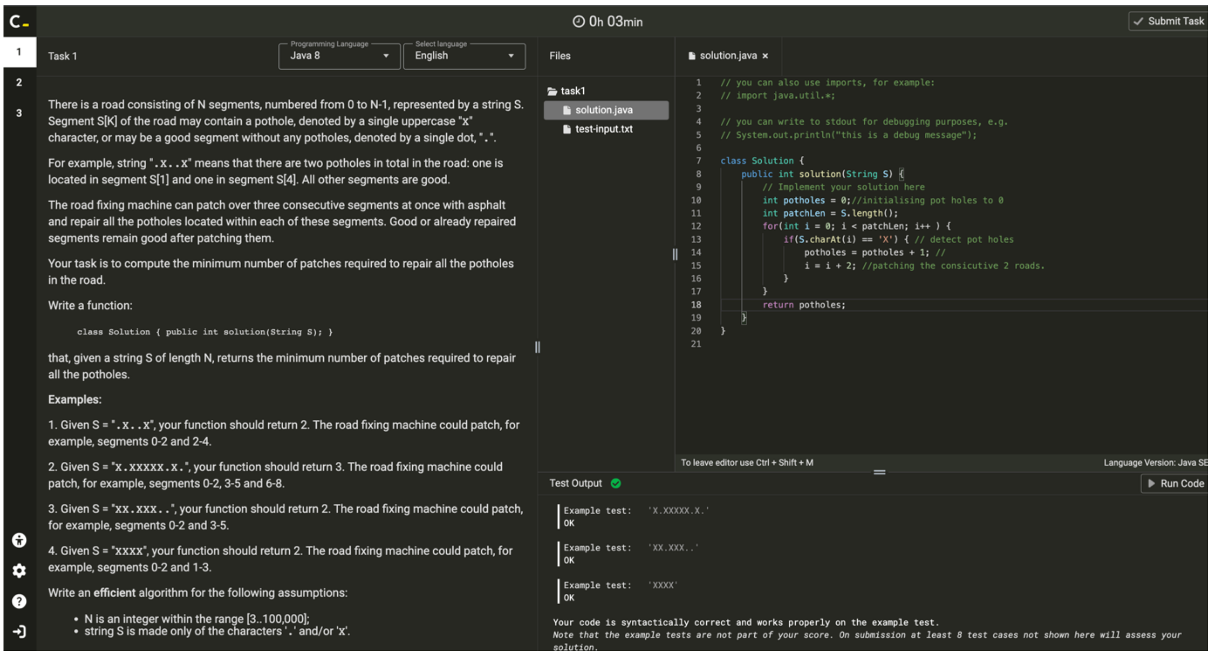
Task 1



Task 1

Programming Language: Java 8

Select language: English

There is a road consisting of N segments, numbered from 0 to N-1, represented by a string S. Segment S[K] of the road may contain a pothole, denoted by a single uppercase "x" character, or may be a good segment without any potholes, denoted by a single dot, ".".

For example, string ".x..x" means that there are two potholes in total in the road: one is located in segment S[1] and one in segment S[4]. All other segments are good.

The road fixing machine can patch over three consecutive segments at once with asphalt and repair all the potholes located within each of these segments. Good or already repaired segments remain good after patching them.

Your task is to compute the minimum number of patches required to repair all the potholes in the road.

Write a function:

```
class Solution { public int solution(String S); }
```

that, given a string S of length N, returns the minimum number of patches required to repair all the potholes.

Examples:

1. Given S = ".x..x", your function should return 2. The road fixing machine could patch, for example, segments 0-2 and 2-4.

2. Given S = "x.xxxxx.x.", your function should return 3. The road fixing machine could patch, for example, segments 0-2, 3-5 and 6-8.

3. Given S = "xx.xxx..", your function should return 2. The road fixing machine could patch, for example, segments 0-2 and 3-5.

4. Given S = "xxxx", your function should return 2. The road fixing machine could patch, for example, segments 0-2 and 1-3.

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

- N is an integer within the range [3..100,000];
- string S is made only of the characters '.' and/or 'x'.

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

// you can write to stdout for debugging purposes, e.g.
// System.out.println("this is a debug message");

class Solution {
    public int solution(String S) {
        // Implement your solution here
        int potholes = 0;//initialising pot holes to 0
        int patchLen = S.length();
        for(int i = 0; i < patchLen; i++ ) {
            if(S.charAt(i) == 'X') { // detect pot holes
                potholes = potholes + 1; //
                i = i + 2; //patching the consicutive 2 roads.
            }
        }
        return potholes;
    }
}

```

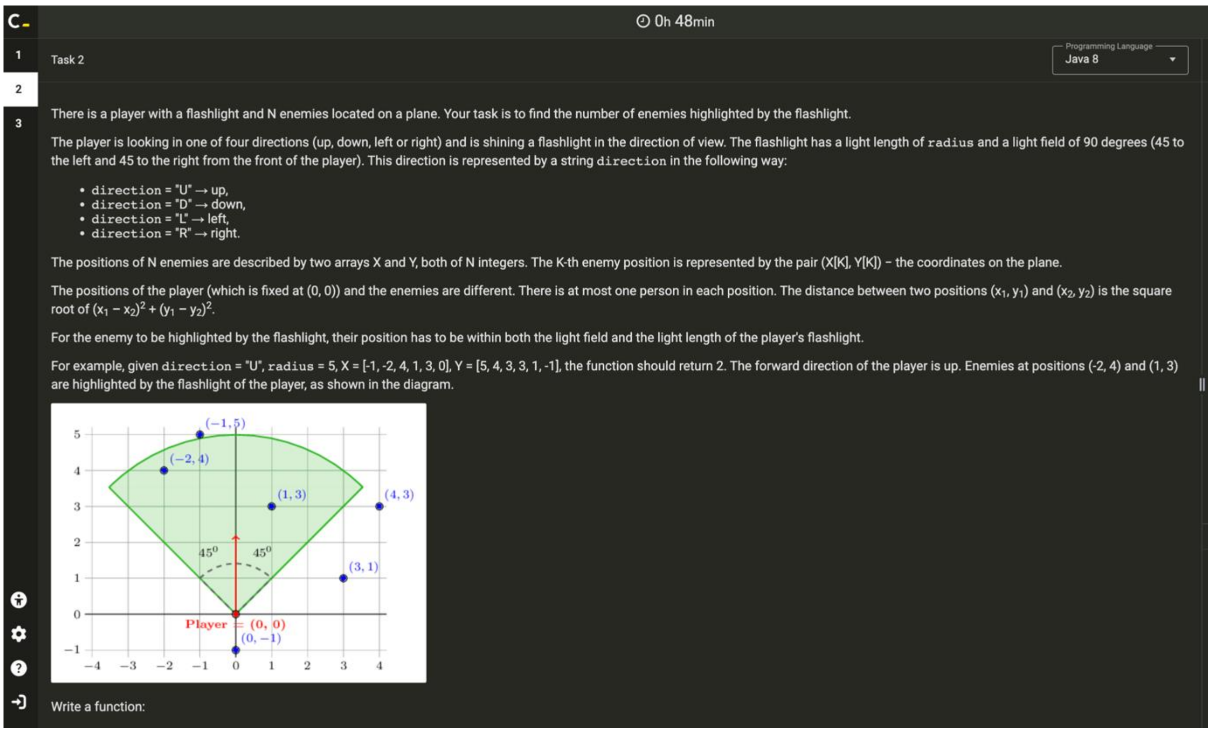
```
Example test:   'X.XXXXX.X.'
OK

Example test:   'XX.XXX..'
OK

Example test:   'XXXX'
OK

Your code is syntactically correct and works properly on the example test.
Note that the example tests are not part of your score. On submission at least 8 test cases not shown here will assess your solution.
```

Task 2 – (Not showing you my solution here since it didn't compile and also, it's embarrassing!)



Task 2

Programming Language: Java 8

There is a player with a flashlight and N enemies located on a plane. Your task is to find the number of enemies highlighted by the flashlight.

The player is looking in one of four directions (up, down, left or right) and is shining a flashlight in the direction of view. The flashlight has a light length of `radius` and a light field of 90 degrees (45 to the left and 45 to the right from the front of the player). This direction is represented by a string `direction` in the following way:

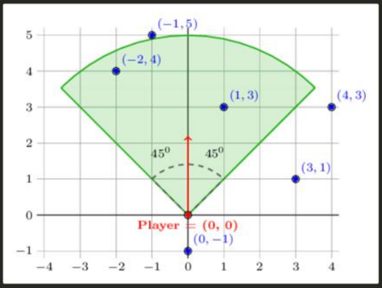
- `direction` = "U" → up,
- `direction` = "D" → down,
- `direction` = "L" → left,
- `direction` = "R" → right.

The positions of N enemies are described by two arrays X and Y, both of N integers. The K-th enemy position is represented by the pair (X[K], Y[K]) – the coordinates on the plane.

The positions of the player (which is fixed at (0, 0)) and the enemies are different. There is at most one person in each position. The distance between two positions (x_1, y_1) and (x_2, y_2) is the square root of $(x_1 - x_2)^2 + (y_1 - y_2)^2$.

For the enemy to be highlighted by the flashlight, their position has to be within both the light field and the light length of the player's flashlight.

For example, given `direction` = "U", `radius` = 5, X = [-1, -2, 4, 1, 3, 0], Y = [5, 4, 3, 3, 1, -1], the function should return 2. The forward direction of the player is up. Enemies at positions (-2, 4) and (1, 3) are highlighted by the flashlight of the player, as shown in the diagram.



Write a function:

0h 48min

Task 2

Programming Language Java 8

Write a function:

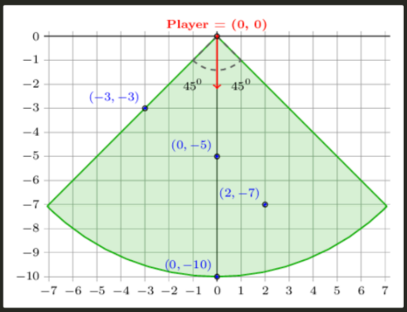
```
class Solution { public int solution(String direction, int radius, int[] X, int[] Y); }
```

that, given string `direction`, integer `radius`, and arrays X and Y of N integers, returns the number of enemies highlighted by the flashlight.

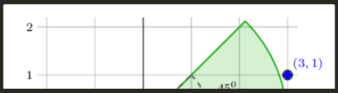
Examples:

1. Given `direction` = "U", `radius` = 5, X = [-1, -2, 4, 1, 3, 0], Y = [5, 4, 3, 3, 1, -1], the function should return 2 as explained above.

2. Given `direction` = "D", `radius` = 10, X = [0, -3, 2, 0], Y = [-10, -3, -7, -5], the function should return 4. The player's forward direction is down. Even though the player and enemies in positions (0, -5) and (0, -10) are in a line, the enemy at position (0, -10) is still visible, so all four enemies are highlighted by the player's flashlight.



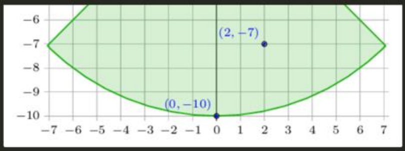
3. Given `direction` = "R", `radius` = 3, X = [-2, 3], Y = [0, 1], the function should return 0. The forward direction of the player is right. None of the enemies are highlighted by the player's flashlight.



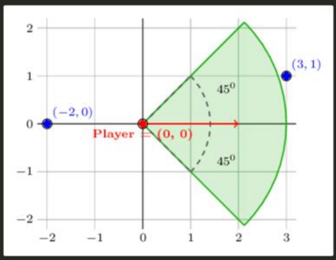
0h 48min

Task 2

Programming Language Java 8



3. Given `direction` = "R", `radius` = 3, X = [-2, 3], Y = [0, 1], the function should return 0. The forward direction of the player is right. None of the enemies are highlighted by the player's flashlight.

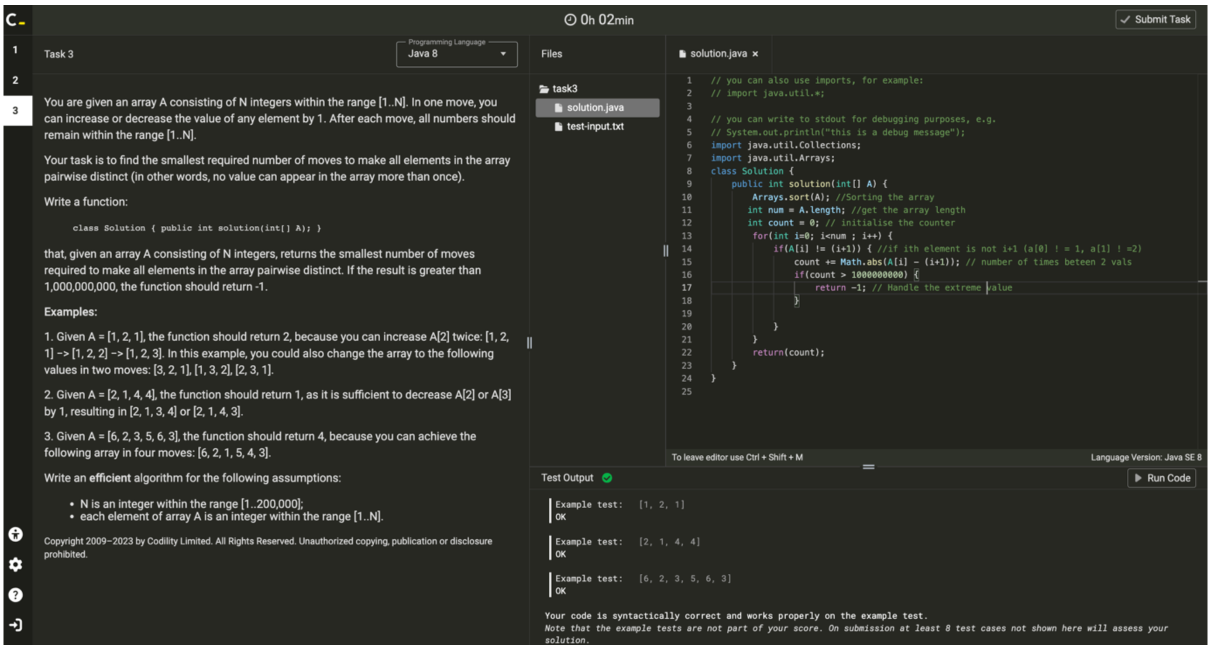


Assume that:

- N is an integer within the range [1..100];
- each element of arrays X and Y is an integer within the range [−10,000..10,000];
- `radius` is an integer within the range [1..10,000];
- `direction` is a string that can have one of the following values: "U", "D", "L", "R";
- size of X and Y arrays are equal;
- all enemies and player are located in different positions.

In your solution, focus on **correctness**. The performance of your solution will not be the focus of the assessment.

Task 3



You are given an array A consisting of N integers within the range [1..N]. In one move, you can increase or decrease the value of any element by 1. After each move, all numbers should remain within the range [1..N].

Your task is to find the smallest required number of moves to make all elements in the array pairwise distinct (in other words, no value can appear in the array more than once).

Write a function:

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

that, given an array A consisting of N integers, returns the smallest number of moves required to make all elements in the array pairwise distinct. If the result is greater than 1,000,000,000, the function should return -1.

Examples:

1. Given A = [1, 2, 1], the function should return 2, because you can increase A[2] twice: [1, 2, 1] -> [1, 2, 2] -> [1, 2, 3]. In this example, you could also change the array to the following values in two moves: [3, 2, 1], [1, 3, 2], [2, 3, 1].

2. Given A = [2, 1, 4, 4], the function should return 1, as it is sufficient to decrease A[2] or A[3] by 1, resulting in [2, 1, 3, 4] or [2, 1, 4, 3].

3. Given A = [6, 2, 3, 5, 6, 3], the function should return 4, because you can achieve the following array in four moves: [6, 2, 1, 5, 4, 3].

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

- N is an integer within the range [1..200,000];
- each element of array A is an integer within the range [1..N].

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

// you can write to stdout for debugging purposes, e.g.
// System.out.println("this is a debug message");
import java.util.Collections;
import java.util.Arrays;
class Solution {
    public int solution(int[] A) {
        Arrays.sort(A); //Sorting the array
       int num = A.length; //get the array length
       int count = 0; // initialise the counter
        for(int i=0; i<num ; i++) {
            if(A[i] != (i+1)) { //if ith element is not i+1 (a[0] ! = 1, a[1] ! =2)
                count += Math.abs(A[i] - (i+1)); // number of times beteen 2 vals
                if(count > 1000000000) {
                    return -1; // Handle the extreme value
                }

            }
        }
        return(count);
    }
}

```

To leave editor use Ctrl + Shift + M

Language Version: Java SE 8

Test Output

```
Example test:   [1, 2, 1]
OK

Example test:   [2, 1, 4, 4]
OK

Example test:   [6, 2, 3, 5, 6, 3]
OK

Your code is syntactically correct and works properly on the example test.
Note that the example tests are not part of your score. On submission at least 8 test cases not shown here will assess your solution.
```