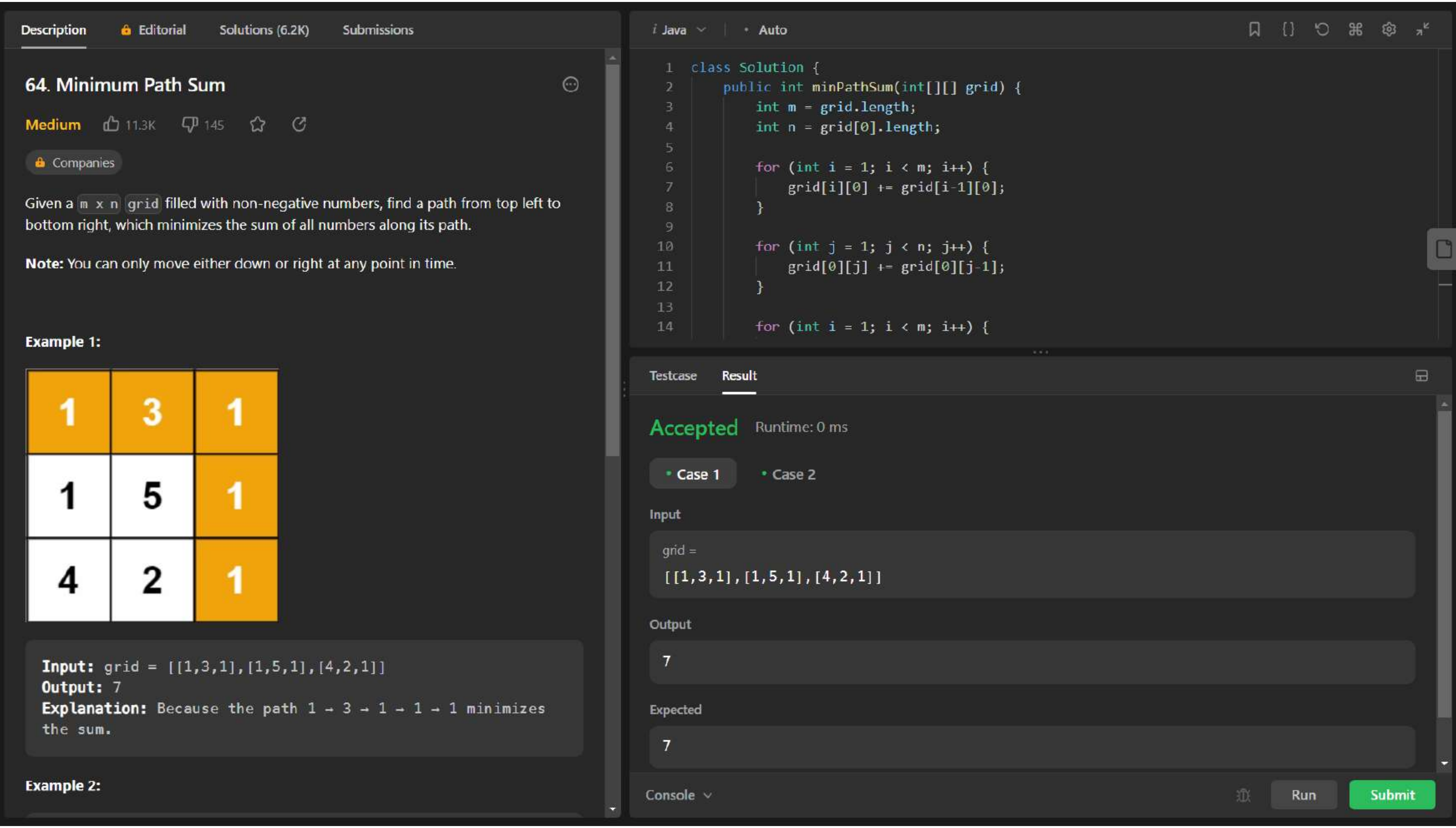


64. Minimum Path Sum

Given a `m x n` `grid` filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

Example 1:

1	3	1
1	5	1
4	2	1

```
Input: grid = [[1,3,1],[1,5,1],[4,2,1]]
Output: 7
Explanation: Because the path 1 → 3 → 1 → 1 → 1 minimizes
the sum.
```

Example 2:

```java
class Solution {
    public int minPathSum(int[][] grid) {
        int m = grid.length;
        int n = grid[0].length;

        for (int i = 1; i < m; i++) {
            grid[i][0] += grid[i-1][0];
        }

        for (int j = 1; j < n; j++) {
            grid[0][j] += grid[0][j-1];
        }

        for (int i = 1; i < m; i++) {
```

Accepted Runtime: 0 ms

Input

grid =

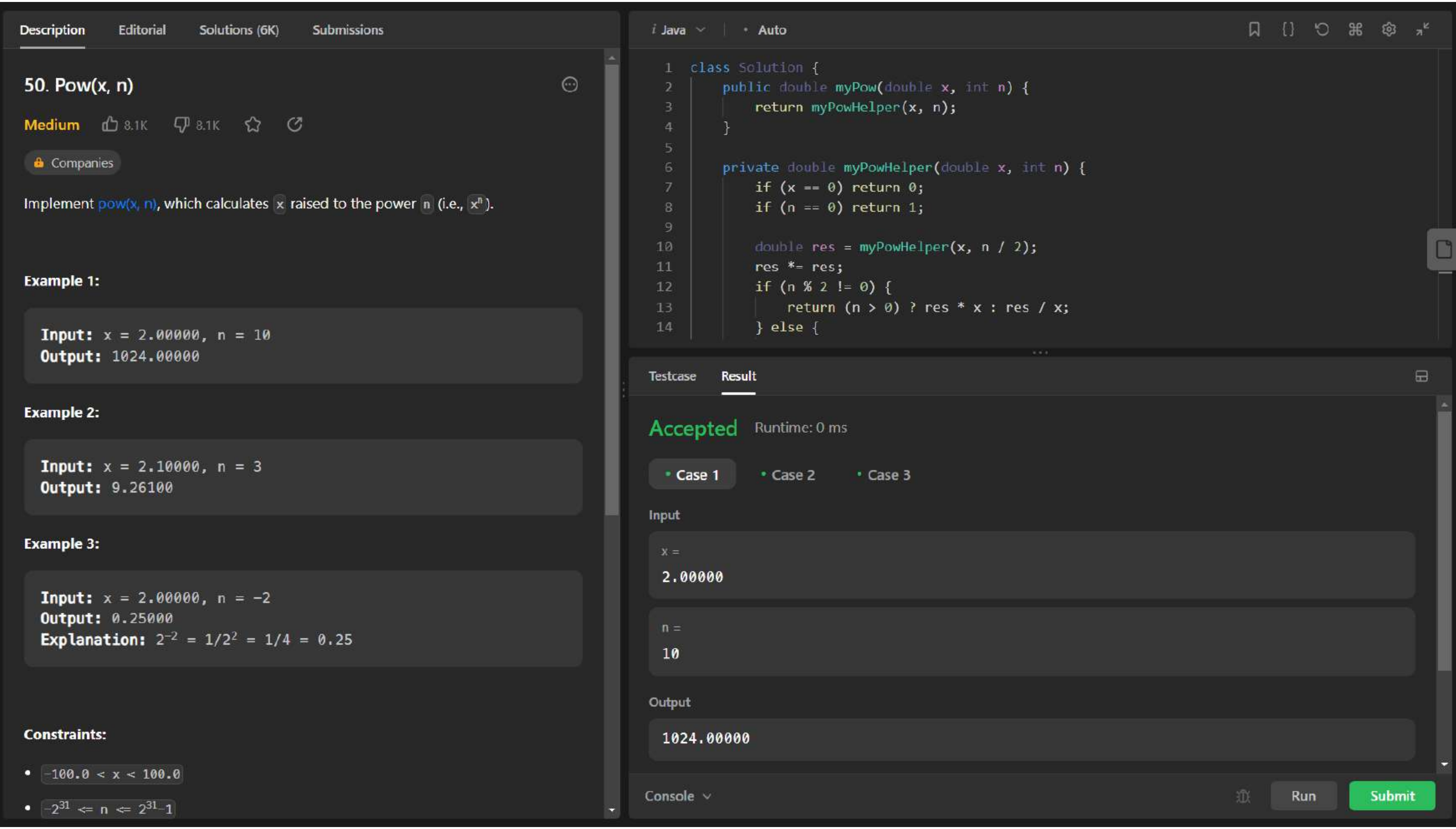
```
[[1,3,1],[1,5,1],[4,2,1]]
```

Output

```
7
```

Expected

```
7
```



50. Pow(x, n)

Implement pow(x, n), which calculates `x` raised to the power `n` (i.e., x^n).

Example 1:

```
Input: x = 2.00000, n = 10
Output: 1024.00000
```

Example 2:

```
Input: x = 2.10000, n = 3
Output: 9.26100
```

Example 3:

```
Input: x = 2.00000, n = -2
Output: 0.25000
Explanation: 2^-2 = 1/2^2 = 1/4 = 0.25
```

Constraints:

- $-100.0 < x < 100.0$
- $-2^{31} <= n <= 2^{31}-1$

```java
class Solution {
    public double myPow(double x, int n) {
        return myPowHelper(x, n);
    }

    private double myPowHelper(double x, int n) {
        if (x == 0) return 0;
        if (n == 0) return 1;

        double res = myPowHelper(x, n / 2);
        res *= res;
        if (n % 2 != 0) {
            return (n > 0) ? res * x : res / x;
        } else {
```

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

x =

```
2.00000
```

n =

```
10
```

Output

```
1024.00000
```