Shri Vile Parle Kelavani Mandal's

INSTITUTE OF TECHNOLOGY

DHULE (M.S.)

DEPARMENT OF COMPUTER ENGINEERING

Subject : Competitive Programmimg Lab

Name : Jaykishan Natwar Varma **Roll No. :** 68

Class : TY. Comp. Engg. **Batch :** T4 **Division:**

Expt. No. : **Date :**

Title : Steps

Remark

Signature

ASSIGNMENT/EXPERIMENT: _____		
Date of Performance:	**Date of Submission:**	
Marks Split Up	**Maximum Marks**	**Marks Obtained**
Performance/Conduction	**3**	
Report Writing	**3**	
Attendance	**2**	
Viva/Oral	**2**	
Total Marks	**10**	
Signature of Subject Teacher		

Title: Steps

Aim: One steps through integer points of the straight line. The length of a step must be nonnegative and can be by one bigger than, equal to, or by one smaller than the length of the previous step. What is the minimum number of steps in order to get from x to y? The length of the first and the last step must be 1.

Language used: Python

Platform Used: Jupyter Notebook

Sample Input and Sample Output: Input consists of a line containing n, the number of test cases. For each test case, a line follows with two integers: $0 \leq x \leq y < 2$ 31. For each test case, print a line giving the minimum number of steps to get from x to y.

Example:

Sample Input :

3

45 48

45 49

45 50

Sample Output:

3

3

4

Algorithm/Flowchart:

1. **Dynamic Programming Approach**: Use dynamic programming to compute the minimum number of steps (`min_steps`) required to reach each point up to yy from xx.
2. **Initialization**:
    - Initialize an array `min_steps` where `min_steps[i]` represents the minimum number of steps needed to reach point ii from xx.
    - Set `min_steps[x] = 0` since no steps are needed to stay at xx.

3. **Fill the Array**:

   - Iterate from xx up to yy and for each position ii, calculate the potential positions you can move to based on the step constraints.
   - For each position ii, consider the previous possible positions $i-1i-1$, ii, and $i+1i+1$ (if within bounds) and update `min_steps` accordingly: \text{min_steps}[i] = \min(\text{min_steps}[i], \text{min_steps}[i-1] + 1, \text{min_steps}[i+1] + 1)
   - Ensure that the length of the first and last step is fixed at 1 (`min_steps[x+1] = 1` and `min_steps[x] = 0`).

4. **Output the Result**:

   - For each test case with given xx and yy, retrieve the value of `min_steps[y]` which represents the minimum number of steps required to reach yy from xx.

Code:

```
def min_steps_to_reach(x, y):
    if x == y:
        return 0

    # Initialize the minimum steps array
    max_val = max(x, y) + 1
    min_steps = [float('inf')] * (max_val + 1)

    # Base cases
    min_steps[x] = 0
    min_steps[x + 1] = 1

    # Fill the min_steps array using dynamic programming
    for i in range(x + 2, y + 1):
        min_steps[i] = min(min_steps[i], min_steps[i - 1] + 1)
        if i + 1 <= y:
            min_steps[i + 1] = min(min_steps[i + 1], min_steps[i] + 1)
        if i - 1 >= x:
            min_steps[i - 1] = min(min_steps[i - 1], min_steps[i] + 1)
```

```
    # Return the minimum steps to reach y
    return min_steps[y]

# Example usage:
if __name__ == "__main__":
    n = int(input("Enter the number of test cases: "))

    for _ in range(n):
        x, y = map(int, input().split())
        result = min_steps_to_reach(x, y)
        print(result)
```

Input:-

3

45 48

45 49

45 50

Output:-

3

3

4

Conclusion: In this way we implement The Steps Problem using loops and conditional statements.