ACTIVITY 11

Important Code Submission Guideline for Students

- Please submit only the function definition in your source code.
- Don’t include the main() function or any input/output code

QUESTION 1

Given two integers dividend and divisor, divide two integers without using multiplication, division, and mod operator.
The integer division should truncate toward zero, which means losing its fractional part. For example, 8.345 would be truncated to 8, and -2.7335 would be truncated to -2. Return the quotient after dividing dividend by divisor.

You need to write a function which takes two integer arguments: **dividend and divisor.**

Expected Time Complexity: O(log N), where N is the absolute value of the dividend.

Note: Assume we are dealing with an environment that could only store integers within the 32-bit signed integer range: $[-2^{31}, 2^{31} - 1]$. For this problem, if the quotient is strictly greater than 2^{31}- 1, then return 2^{31}- 1, and if the quotient is strictly less than -2^{31}, then return -2^{31}.

Example 1:
Input: dividend = 10, divisor = 3
Output: 3
Explanation: 10/3 = 3.33333.. which is truncated to 3.
Example 2:
Input: dividend = 7, divisor = -3
Output: -2
Explanation: 7/-3 = -2.33333.. which is truncated to -2.

Constraints:

· -2^{31} <= dividend, divisor <= 2^{31}- 1
· divisor != 0

QUESTION 2

You are given an n x n integer matrix board where the cells are labeled from 1 to n2 in a Boustrophedon style starting from the bottom left of the board (i.e. board[n - 1][0]) and alternating direction each row.

You start on square 1 of the board. In each move, starting from square curr, do the following:

Choose a destination square next with a label in the range [curr + 1, min(curr + 6, n2)].

This choice simulates the result of a standard 6-sided die roll: i.e., there are always at most 6 destinations, regardless of the size of the board.

If next has a snake or ladder, you must move to the destination of that snake or ladder. Otherwise, you move to next.

The game ends when you reach the square n2.

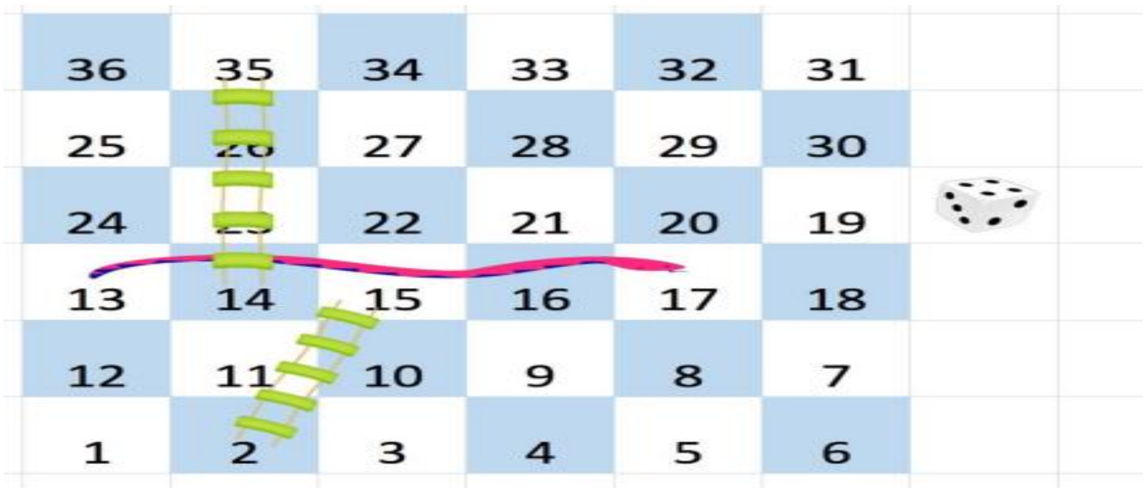
A board square on row r and column c has a snake or ladder if board[r][c] != -1. The destination of that snake or ladder is board[r][c]. Squares 1 and n2 are not the starting points of any snake or ladder.

Note that you only take a snake or ladder at most once per dice roll. If the destination to a snake or ladder is the start of another snake or ladder, you do not follow the subsequent snake or ladder.

For example, suppose the board is [[-1,4],[-1,3]], and on the first move, your destination square is 2. You follow the ladder to square 3, but do not follow the subsequent ladder to 4.

Return the least number of dice rolls required to reach the square n2. If it is not possible to reach the square, return -1.

You need to write a function **snakesAndLadders** that takes a 2D matrix *board* as an input.
Example 1:



Input: board =
[[-1,-1,-1,-1,-1,-1],[-1,-1,-1,-1,-1,-1],[-1,-1,-1,-1,-1,-1],[-1,35,-1,-1,13,-1],[-1,-1,-1,-1,-1,-1],[-1,15,-1,-1,-1,-1]]

Output: 4

Explanation:

In the beginning, you start at square 1 (at row 5, column 0).

You decide to move to square 2 and must take the ladder to square 15.

You then decide to move to square 17 and must take the snake to square 13.

You then decide to move to square 14 and must take the ladder to square

35. You then decide to move to square 36, ending the game.

This is the lowest possible number of moves to reach the last square, so return 4.
Example 2:

Input: board = [[-1,-1],[-1,3]]

Output: 1

Constraints:

n == board.length == board[i].length

2 <= n <= 20

board[i][j] is either -1 or in the range [1, n2].

The squares labeled 1 and n2 are not the starting points of any snake or ladder.