# Weekly Contest - 10

## 463. Island Perimeter

Solved

Easy · Topics · Companies

You are given `row x col` `grid` representing a map where `grid[i][j] = 1` represents land and `grid[i][j] = 0` represents water.

Grid cells are connected **horizontally/vertically** (not diagonally). The `grid` is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells).

The island doesn't have "lakes", meaning the water inside isn't connected to the water around the island. One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

**Example 1:**

**Input:** grid = [[0,1,0,0],[1,1,1,0],[0,1,0,0],[1,1,0,0]]
**Output:** 16
**Explanation:** The perimeter is the 16 yellow stripes in the image above.

**Example 2:**

**Input:** grid = [[1]]
**Output:** 4

Check each cell, add to perimeter if the side of the cell connected to water or connected to boundary

```
public static int islandPerimeter(int[][] grid) {
    int total = 0;
    int rows = grid.length;
    int cols = grid[0].length;

    for (int r=0; r<rows; r++) {
        for (int c=0; c<cols; c++) {
            if (grid[r][c] == 1) {
                // if land on the lower grid
                if (r<rows-1 && grid[r+1][c] == 0 || r+1==rows)
                    total++;
                // if land on the upper grid
                if (r>0 && grid[r-1][c] == 0 || r==0)
                    total++;
                // if land on the left grid
                if (c>0 && grid[r][c-1] == 0 || c==0)
                    total++;
                // if land on the right grid
                if (c<cols-1 && grid[r][c+1] == 0 || c+1==cols)
                    total++;
            }
        }
    } return total;
}
```

## 462. Minimum Moves to Equal Array Elements II

Solved

Medium · Topics · Companies

Given an integer array `nums` of size `n`, *return the minimum number of moves required to make all array elements equal*.

In one move, you can increment or decrement an element of the array by `1`.

Test cases are designed so that the answer will fit in a **32-bit** integer.

**Example 1:**

**Input:** nums = [1,2,3]
**Output:** 2
**Explanation:**
Only two moves are needed (remember each move increments or decrements one element):
[1,2,3] => [2,2,3] => [2,2,2]

**Example 2:**

**Input:** nums = [1,10,2,9]
**Output:** 16

**Constraints:**

- `n == nums.length`
- `1 <= nums.length <= 10^5`
- `-10^9 <= nums[i] <= 10^9`

1 10 2 9 sort 1 2 9 10

find the median, nums[2/2] = 2/1 = 2

→ nums(2) → 9

1 2 7 9 13 → nums(R) - median

↳ Median - nums(L)

= nums(R) - median + median - nums(L)

= nums(R) - nums(L)

```
public static int minMoves2(int[] nums) {
    Arrays.sort(nums);
    int ops = 0;
    int median = nums[nums.length/2];

    for (int num : nums)
        ops += Math.abs(median - num);
    return ops;
}
```

## 464. Can I Win

Solved

Medium · Topics · Companies

In the "100 game" two players take turns adding, to a running total, any integer from `1` to `10`. The player who first causes the running total to **reach or exceed** 100 wins.

What if we change the game so that players **cannot** re-use integers?

For example, two players might take turns drawing from a common pool of numbers from 1 to 15 without replacement until they reach a total >= 100.

Given two integers `maxChoosableInteger` and `desiredTotal`, return `true` if the first player to move can force a win, otherwise, return `false`. Assume both players play **optimally**.

**Example 1:**

**Input:** maxChoosableInteger = 10, desiredTotal = 11
**Output:** false
**Explanation:**
No matter which integer the first player choose, the first player will lose.
The first player can choose an integer from 1 up to 10.
If the first player choose 1, the second player can only choose integers from 2 up to 10.
The second player will win by choosing 10 and get a total = 11, which is >= desiredTotal.
Same with other integers chosen by the first player, the second player will always win.

**Example 2:**

**Input:** maxChoosableInteger = 10, desiredTotal = 0
**Output:** true

Bit masking to track the used numbers

K → 10 9 8 7 6 5 4 3 2 1 0

0 0 0 0 1 0 1 0 0 0 0

mask 1 → the number is used

↳ 0 → unused, can be used

Edge Case

① desired total ≤ max choosable numbers → player 1 takes the max(K, total) WIN

② sum of numbers (1 K) ≤ total → Game is unplayable

1 2 3 4 5 6 7 8 . . . max

2 3 4 . . 8 — Try all numbers in the range

and check if the opponent can win the game with the remaining numbers

```
private static Boolean[] memo;  5 usages
public static boolean canIWin(int max, int desiredTotal) {
    if (desiredTotal <= max) return true;
    // Check if the game is playable
    // Sum of the all the numbers from 1 to max -> n * (n+1) / 2
    if (max * (max + 1) / 2 < desiredTotal)
        return false;

    memo = new Boolean[1 << max + 1];
    // Masking -> used mask -> initially 0 -> when the number is used -> set the bit to 1
    // bit 0 -> unused, bit 1 -> used
    return solve(max, desiredTotal, usedMask: 0);
}
private static boolean solve(int max, int total, int usedMask) {
    // if the desired total is already reached, the current player loses to prev player
    if (total <= 0) return false;
    if (memo[usedMask] != null) return memo[usedMask];

    for (int i=1; i<=max; i++) {
        int currentBit = 1 << i;
        if ((usedMask & currentBit) == 0) { // numbers cannot be reused
            boolean opponent = solve(max, total: total-i, usedMask: usedMask | currentBit);
            if (!opponent) {
                memo[usedMask] = true;
                return true;
            }
        }
    }
    memo[usedMask] = false;
    return false;
}
```