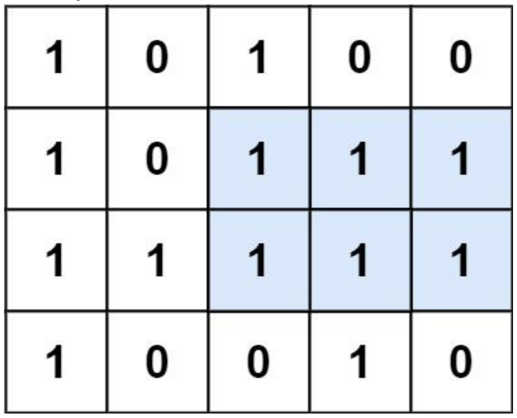
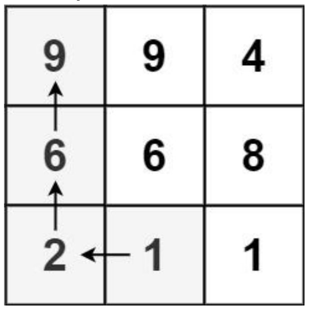
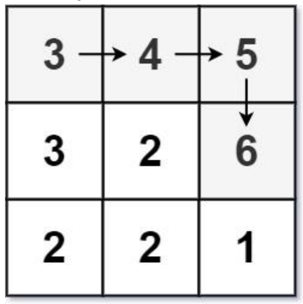
Wednesday 3/30/22

DP: https://www.youtube.com/watch?v=P8Xa2BitN3I&t=26s&ab_channel=HackerRank
https://www.youtube.com/watch?v=aPQY__2H3tE&ab_channel=Reducible

HW9:

1	Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses. Example 1: Input: n = 3 Output: ["((()))","(()())","(())()","()(())","()()()"] Example 2: Input: n = 1 Output: ["()"] Constraints: 1 <= n <= 8
2	Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area. Example 1:  Input: matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]] Output: 6 Explanation: The maximal rectangle is shown in the above picture. Example 2: Input: matrix = [["0"]]

	Output: 0 Example 3: Input: matrix = [["1"]] Output: 1
3	We can scramble a string s to get a string t using the following algorithm: 1. If the length of the string is 1, stop. 2. If the length of the string is > 1, do the following: ○ Split the string into two non-empty substrings at a random index, i.e., if the string is s, divide it to x and y where s = x + y. ○ Randomly decide to swap the two substrings or to keep them in the same order. i.e., after this step, s may become s = x + y or s = y + x. ○ Apply step 1 recursively on each of the two substrings x and y. Given two strings s1 and s2 of the same length, return true if s2 is a scrambled string of s1, otherwise, return false. Example 1: Input: s1 = "great", s2 = "rgeat" Output: true Explanation: One possible scenario applied on s1 is: "great" --> "gr/eat" // divide at random index. "gr/eat" --> "gr/eat" // random decision is not to swap the two substrings and keep them in order. "gr/eat" --> "g/r / e/at" // apply the same algorithm recursively on both substrings. divide at random index each of them. "g/r / e/at" --> "r/g / e/at" // random decision was to swap the first substring and to keep the second substring in the same order. "r/g / e/at" --> "r/g / e/ a/t" // again apply the algorithm recursively, divide "at" to "a/t". "r/g / e/ a/t" --> "r/g / e/ a/t" // random decision is to keep both substrings in the same order. The algorithm stops now, and the result string is "rgeat" which is s2. As one possible scenario led s1 to be scrambled to s2, we return true. Example 2: Input: s1 = "abcde", s2 = "caebd" Output: false Example 3: Input: s1 = "a", s2 = "a" Output: true Constraints: ● s1.length == s2.length ● 1 <= s1.length <= 30 ● s1 and s2 consist of lowercase English letters.

4	Given an m x n integers matrix, return the length of the longest increasing path in matrix. From each cell, you can either move in four directions: left, right, up, or down. You may not move diagonally or move outside the boundary (i.e., wrap-around is not allowed). Example 1:  Input: matrix = [[9,9,4],[6,6,8],[2,1,1]] Output: 4 Explanation: The longest increasing path is [1, 2, 6, 9]. Example 2:  Input: matrix = [[3,4,5],[3,2,6],[2,2,1]] Output: 4 Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed. Example 3: Input: matrix = [[1]] Output: 1 Constraints: ● m == matrix.length ● n == matrix[i].length ● 1 <= m, n <= 200 ● 0 <= matrix[i][j] <= 231 - 1
5	Given an integer n, return the least number of perfect square numbers that sum to n.

	A perfect square is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, 1, 4, 9, and 16 are perfect squares while 3 and 11 are not. Example 1: Input: n = 12 Output: 3 Explanation: 12 = 4 + 4 + 4. Example 2: Input: n = 13 Output: 2 Explanation: 13 = 4 + 9. Constraints: 1 <= n <= 104
6	A frog is crossing a river. The river is divided into some number of units, and at each unit, there may or may not exist a stone. The frog can jump on a stone, but it must not jump into the water. Given a list of stones' positions (in units) in sorted ascending order, determine if the frog can cross the river by landing on the last stone. Initially, the frog is on the first stone and assumes the first jump must be 1 unit. If the frog's last jump was k units, its next jump must be either k - 1, k, or k + 1 units. The frog can only jump in the forward direction. Example 1: Input: stones = [0,1,3,5,6,8,12,17] Output: true Explanation: The frog can jump to the last stone by jumping 1 unit to the 2nd stone, then 2 units to the 3rd stone, then 2 units to the 4th stone, then 3 units to the 6th stone, 4 units to the 7th stone, and 5 units to the 8th stone. Example 2: Input: stones = [0,1,2,3,4,8,9,11] Output: false Explanation: There is no way to jump to the last stone as the gap between the 5th and 6th stone is too large. Constraints: ● 2 <= stones.length <= 2000 ● 0 <= stones[i] <= 231 - 1 ● stones[0] == 0 ● stones is sorted in a strictly increasing order.