1.1 Q: Given a fifty-seat room where each seat is numbered, people entered the room in order. The first one is drunk and he will take a seat randomly. The rest people will take their own seat as long as it is not taken. Calculate the probability that last two people can take their own seats.
A: The number 1, $n-1$ and n have equal probabilities to be taken first. The last 2 people will get correct seats if and only if seat 1 is taken before $n-1$ and n. So the probability is $1/3$ when $n \geq 3$.

1.2 Q: There are 5 girls who need to guard a store from Mon-Fri. Calculate the number of possible arrangements given the constraint: (a) Everyone works twice a week. (b) Two girls guard the store in a single day and no two girls can work together twice.
A: First, split the girls into 5 groups of 2 each, satisfying both constraints. There are $A_5^5 = 5! = 120$ permutations to assign 5 groups to 5 days.
Second, let the girls be nodes and connect each pair of girls in the same group with edges. Each node has 2 edges. Since no 2 nodes can be connected twice, the only way is to have a cycle with 5 nodes. There are 4! possibilities of such cycles with directions, and thus $4!/2 = 12$ combinations of such cycles without directions.
Thus, the answer is

$$5! \times \frac{4!}{2} = 1440$$

1.3 Q: Follow up, what if there are 7 girls and 7 days schedule to be set up, what is the number of possible arrangements?
A: Same argument as above. There are $7! = 5040$ permutations to assign 7 groups to 7 days. As for the groups, there're two kinds of cycles: (1) a cycle with 7 nodes, (2) two cycles with 3 and 4 nodes each. Thus, the total number of possible arrangements:

$$7! \times \left[\frac{6!}{2} + \binom{7}{3}\left(\frac{2!}{2} \times \frac{3!}{2}\right)\right] = 5040 \times 465 = 2{,}343{,}600$$

1.8 Q: Consider the basketball shooting game, define success rate as number of successful shoots divided by number of total shoots. Assume the successful rate rising from below 0.5 to above 0.5, is there a moment which has exactly success rate 0.5?
A: Yes. Assume we have a successful shoots out of total shoots of b. Before jump, we have

$$\frac{a}{b} < 0.5$$

and after jump, we have:

$$\frac{a+1}{b+1} > 0.5$$

Rearranging the above two inequalities, we will get

$$b - 1 < 2a < b$$

which is not possible. Thus, there must exist a moment the success rate is exactly 0.5.

1.9 Q: Given two strategies A and B, as well as the corresponding P&L of these strategies on each day. If one is going to be shut down, how to decide which one to shut?
A: There are many aspects to consider, including tracking record, client demand, operation cost, competitive advantage, capacity constraints, capital allocation efficiency, regulatory risks, etc.

1.10 Q: Design an algorithm to check is a point is inside the triangle or not.
A:
https://www.geeksforgeeks.org/check-whether-a-given-point-lies-inside-a-triangle-or-not/

1.11 Q: Design an algorithm to check palindrome sequence.
A:

```python
class Solution:
    def isPalindrome(self, s: str) -> bool:
        if len(s) <= 1:
            return True

        left, right = 0, len(s) - 1
        while left < right:
            while left < right and not s[left].isalnum():
                left += 1

            while left < right and not s[right].isalnum():
                right -= 1

            if left < right and s[left].lower() != s[right].lower():
                return False

            left += 1
            right -= 1

        return True
```

1.12 Q: Given an array, design an algorithm to return the longest decreasing sub-array.
A:

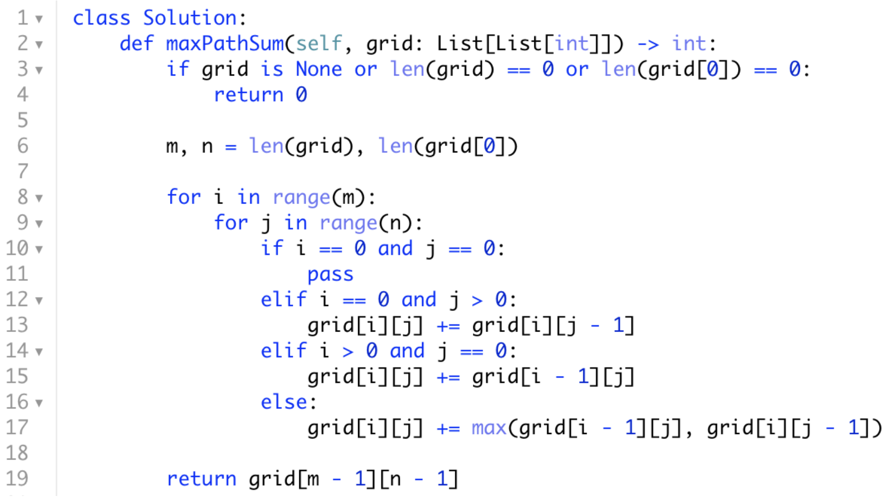
```python
class Solution:
    def lengthOfLDS(self, nums: List[int]) -> int:
        if nums is None or len(nums) == 0:
            return 0

        length = len(nums)
        f = [1] * length
        for curr in range(length):
            for prev in range(curr):
                if nums[prev] > nums[curr]:
                    f[curr] = max(f[curr], f[prev] + 1)

        return max(f)
```

1.13 Q: Implement the Sudoko algorithm.
A:

```python
from collections import defaultdict

class Solution:
    def solveSudoku(self, board):
        """
        :type board: List[List[str]]
        :rtype: void Do not return anything, modify board in-place instead.
        """
        def could_place(d, row, col):
            """
            Check if one could place a number d in (row, col) cell
            """
            return not (d in rows[row] or d in columns[col] or \
                    d in boxes[box_index(row, col)])

        def place_number(d, row, col):
            """
            Place a number d in (row, col) cell
            """
            rows[row][d] += 1
            columns[col][d] += 1
            boxes[box_index(row, col)][d] += 1
            board[row][col] = str(d)

        def remove_number(d, row, col):
            """
            Remove a number which didn't lead
            to a solution
            """
            del rows[row][d]
            del columns[col][d]
            del boxes[box_index(row, col)][d]
            board[row][col] = '.'

        def place_next_numbers(row, col):
            """
            Call backtrack function in recursion
            to continue to place numbers
            till the moment we have a solution
            """
            # if we're in the last cell
            # that means we have the solution
            if col == N - 1 and row == N - 1:
                nonlocal sudoku_solved
                sudoku_solved = True
            #if not yet
            else:
                # if we're in the end of the row
                # go to the next row
                if col == N - 1:
                    backtrack(row + 1, 0)
                # go to the next column
                else:
                    backtrack(row, col + 1)


        def backtrack(row = 0, col = 0):
            """
            Backtracking
            """
            # if the cell is empty
            if board[row][col] == '.':
                # iterate over all numbers from 1 to 9
                for d in range(1, 10):
                    if could_place(d, row, col):
                        place_number(d, row, col)
                        place_next_numbers(row, col)
                        # if sudoku is solved, there is no need to backtrack
                        # since the single unique solution is promised
                        if not sudoku_solved:
                            remove_number(d, row, col)
            else:
                place_next_numbers(row, col)

        # box size
        n = 3
        # row size
        N = n * n
        # lambda function to compute box index
        box_index = lambda row, col: (row // n ) * n + col // n

        # init rows, columns and boxes
        rows = [defaultdict(int) for i in range(N)]
        columns = [defaultdict(int) for i in range(N)]
        boxes = [defaultdict(int) for i in range(N)]
        for i in range(N):
            for j in range(N):
                if board[i][j] != '.':
                    d = int(board[i][j])
                    place_number(d, i, j)

        sudoku_solved = False
        backtrack()
```

1.14 Q: Consider a chess board with each cell has a value on it, you can only walk right or down, find the path from up-left to down-right which has largest value.

A:



```python
class Solution:
    def maxPathSum(self, grid: List[List[int]]) -> int:
        if grid is None or len(grid) == 0 or len(grid[0]) == 0:
            return 0

        m, n = len(grid), len(grid[0])

        for i in range(m):
            for j in range(n):
                if i == 0 and j == 0:
                    pass
                elif i == 0 and j > 0:
                    grid[i][j] += grid[i][j - 1]
                elif i > 0 and j == 0:
                    grid[i][j] += grid[i - 1][j]
                else:
                    grid[i][j] += max(grid[i - 1][j], grid[i][j - 1])

        return grid[m - 1][n - 1]
```