16. 3Sum Closest

We will do the same thing as 3sum except that we will compare current difference with minimum difference. In the end we will return the result causing minimum difference. TC is O(n^2)

```
class Solution:
    def threeSumClosest(self, nums: List[int], target: int) -> int:
        nums.sort()
        length = len(nums)
        res = float('inf')
        for i in range(0, length - 2):
            l, r = i + 1, length - 1
            while l < r:
                cur_sum = nums[i] + nums[l] + nums[r]
                if cur_sum == target:
                    return target
                elif cur_sum < target:
                    l += 1
                else:
                    r -= 1
                if abs(cur_sum - target) < abs(res - target):
                    res = cur_sum
        return res
```

36. Valid Sudoku

We will use 27 set to store all rows, columns and small cells. TC is O(1)

```
class Solution:
    def isValidSudoku(self, board: List[List[str]]) -> bool:
        memo = [set() for i in range(27)]
        for i in range(9):
            for j in range(9):
                if board[i][j] != '.':
                    if board[i][j] in memo[i] or board[i][j] in memo[9
+ j] or board[i][j] in memo[18 + i // 3 * 3 + j // 3]:
                        return False
                    else:
                        memo[i].add(board[i][j])
                        memo[9 + j].add(board[i][j])
                        memo[18 + i // 3 * 3 + j // 3].add(board[i][j])
        return True
```

55. Jump Game

We will iterate through all elements and add it with its index and always remember the farthest index we could go. When the farthest index is equal or further than the end. We will return True. When nums[index] == 0 and farthest index is the current index, we will return False. TC is O(n)

```
class Solution:
    def canJump(self, nums: List[int]) -> bool:
```

```
        last = len(nums) - 1
        fatherest = 0
        for idx, i in enumerate(nums):
            cur = idx + i
            if cur >= last:
                return True
            fatherest = max(fatherest, cur)
            if i == 0 and fatherest <= idx:
                return False
```

59. Spiral Matrix II

We will insert our accumulating number to our matrix in spiral direction. TC is O(n^2)

```
class Solution:
    def generateMatrix(self, n: int) -> List[List[int]]:
        result = [[0 for j in range(n)] for i in range(n)]
        count = 1
        dest = n * n + 1
        l, r, t, b = 0, n - 1, 0, n - 1
        while count < dest:
            for j in range(l, r + 1):
                result[t][j] = count
                count += 1
            t += 1

            for i in range(t, b + 1):
```

```
                result[i][r] = count
                count += 1
            r -= 1

            for j in range(r, l - 1, -1):
                result[b][j] = count
                count += 1
            b -= 1

            for i in range(b, t - 1, -1):
                result[i][l] = count
                count += 1
            l += 1

        return result
```

60. Permutation Sequence

We will add candidates based on their rest and mod each time k divmod (n - 1)! TC is O(1)

```
class Solution:
    def getPermutation(self, n: int, k: int) -> str:
        prod = 1
        candidates = list(range(1, n + 1))
        result = ''
        k -= 1
        for i in range(1, n):
```

```
    prod *= i
for i in range(n - 1, 0, -1):
    res, mod = divmod(k, prod)
    prod = prod // i
    k = mod
    num = sorted(candidates)[res]
    candidates.remove(num)
    result += str(num)
result += str(candidates[0])
return result
```