1175. Prime Arrangements
We will get the number of prime numbers and our result would be num! * (n - num)! TC is O(n ** n)

```
class Solution:
    def numPrimeArrangements(self, n: int) -> int:
        prime_number = 0
        result = 1
        for i in range(2, n + 1):
            if self.isPrime(i):
                prime_number += 1
        for i in range(1, n - prime_number + 1):
            result *= i
        for i in range(1, prime_number + 1):
            result *= i
        return result % (10**9 + 7)
    def isPrime(self, num):
        if num == 2:
            return True
        for i in range(2, num):
            if num % i == 0:
                return True
        return False
```

1176. Diet Plan Performance
We will use slide window to get every sub window's sum and compare it with our upper and lower. And add associated points to result. TC is O(n)

```
class Solution:
    def dietPlanPerformance(self, calories: List[int], k: int, lower: int, upper: int) -> int:
        points = 0
        cur = sum(calories[:k - 1] + [0])
        for i in range(k - 1, len(calories)):
            cur += calories[i]
            if cur > upper:
                points += 1
            elif cur < lower:
                points -= 1
            cur -= calories[i - k + 1]
        return points
```

1177. Can Make Palindrome from Substring
We will use a nested array to remember each character's index in s. Then We will use binary search to add each query's element's number % 2 to count. In the end, we will return the

number of characters we need to replace and compare it with k. If they are equal or k is larger, we will return True, otherwise false. TC is O(mlogn)

```
from collections import Counter
from collections import defaultdict
from bisect import *
class Solution:
    def canMakePaliQueries(self, s: str, queries: List[List[int]]) -> List[bool]:
        result = []
        self.memo = {}
        self.dict = defaultdict(list)
        for index, i in enumerate(s):
            self.dict[i].append(index)
        for start, end, k in queries:
            if (end - start + 1) // 2 <= k:
                result.append(True)
            else:
                if self.parDif(start, end, s) <= k:
                    result.append(True)
                else:
                    result.append(False)
        return result

    def parDif(self, start, end, s):
        if (start, end) in self.memo:
            return self.memo[(start, end)]
        count = 0
        for i in 'abcdefghijklmnopqrstuvwxyz':
            count += (bisect(self.dict[i], end) - bisect_left(self.dict[i], start)) % 2

        if (end - start + 1) % 2 == 0:
            self.memo[(start, end)] = count // 2
            return self.memo[(start, end)]
        else:
            self.memo[(start, end)] = (count - 1) // 2
            return self.memo[(start, end)]
```

73. Set Matrix Zeroes

We will use two sets to record which column and row are set zero. Then iterate these rows and columns and set them to zero. TC is O(mn)

```
class Solution:
    def setZeroes(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
```

```
        """
        rows_set, cols_set = set(), set()
        if not matrix or not matrix[0]:
            return matrix
        rows = len(matrix)
        cols = len(matrix[0])
        for i in range(rows):
            for j in range(cols):
                if matrix[i][j] == 0:
                    rows_set.add(i)
                    cols_set.add(j)

        for i in rows_set:
            for j in range(cols):
                matrix[i][j] = 0

        for j in cols_set:
            for i in range(rows):
                matrix[i][j] = 0
```

226. Invert Binary Tree
We will invert our binary tree recursively. TC is O(n)

```
class Solution:
    def invertTree(self, root: TreeNode) -> TreeNode:
        self.helper(root)
        return root

    def helper(self, node):
        if node:
            node.left, node.right = node.right, node.left
            if node.left:
                self.helper(node.left)
            if node.right:
                self.helper(node.right)
```