1.Maximum XOR of Two Non-Overlapping Subtrees

```
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.children = []

def dfs(node, parent, values, subtree_sums):
    subtree_sum = values[node]
    for child in node.children:
        if child != parent:
            subtree_sum += dfs(child, node, values, subtree_sums)
    subtree_sums[node] = subtree_sum
    return subtree_sum

def max_xor_score(n, edges, values):
    tree = [TreeNode(val) for val in values]

    for edge in edges:
        tree[edge[0]].children.append(tree[edge[1]])
        tree[edge[1]].children.append(tree[edge[0]])

    subtree_sums = [0] * n
    dfs(tree[0], None, values, subtree_sums)
    max_score = 0
    for i in range(n):
        for j in range(i + 1, n):
            if i == 0 or j == 0 or (subtree_sums[i] < subtree_sums[0] and subtree_sums[j] < subtree_sums[0]):
                max_score = max(max_score, subtree_sums[i] ^ subtree_sums[j])

    return max_score

# Example usage:
n = 6
edges = [[0,1],[0,2],[1,3],[1,4],[2,5]]
values = [2,8,3,6,2,5]
print(max_xor_score(n, edges, values))  # Output: 24
```

3. Minimum Cuts to Divide a Circle

```
class Solution:
    def numberOfCuts(self, n: int) -> int:
        if n == 1:
            return 0
        return n // 2 if n % 2 == 0 else n
```

4. Difference Between Ones and Zeros in Row and Column

```
class Solution:
    def onesMinusZeros(self, grid: List[List[int]]) -> List[List[int]]:
        m=len(grid)
        n=len(grid[0])
        onerow=[]
        onecol=[]
        zerorow=[]
        zerocol=[]
        one=0
        zero=0
        for i in range (m):
            j=0
            while(j<n):
                if(grid[i][j]==0):
                    zero += 1
                else:
                    one += 1
                j += 1
            zerorow.append(zero)
            onerow.append(one)
            zero=0
            one=0
```

```
zero = 0

one = 0

for j in range(n):

    i=0

    while(i<m):

        if(grid[i][j]==0):

            zero+=1

        else:

            one+=1

        i+=1

    zerocol.append(zero)

    onecol.append(one)

    zero=0

    one=0

diff = [[0 for j in range(n)] for i in range(m)]



for i in range(m):

    for j in range(n):

        diff[i][j] = onerow[i] + onecol[j] - zerorow[i] - zerocol[j]

return diff
```

5. Minimum Penalty for a Shop

```
def min_penalty_hour(customers):
    n = len(customers)
    min_penalty = float('inf')
    best_hour = 0

    for hour in range(n + 1):
        penalty = 0
        for i in range(n):
            if (i < hour and customers[i] == 'Y') or (i >= hour and customers[i] == 'N'):
                penalty += 1
        if penalty < min_penalty:
            min_penalty = penalty
            best_hour = hour

    return best_hour
customers1 = "YYNY"
customers2 = "NNNNN"
customers3 = "YYYY"
print("Optimal closing time for customers1:", min_penalty_hour(customers1))
print("Optimal closing time for customers2:",
min_penalty_hour(customers2))

print("Optimal closing time for customers3:",
min_penalty_hour(customers3))
```

6. Count Palindromic Subsequences

```
MOD = 10**9 + 7


def count_palindromic_subsequences(s):
    n = len(s)
    dp = [[0] * n for _ in range(n)]

    for length in range(1, 6):
        for i in range(n - length + 1):
            j = i + length - 1

            if length == 1:
                dp[i][j] = 1
            elif length == 2:
                dp[i][j] = 2 if s[i] == s[j] else 1
            else:
                dp[i][j] = (2 * dp[i + 1][j - 1]) % MOD
                if s[i] == s[j]:
                    dp[i][j] = (dp[i][j] + 2) % MOD
    count = 0
    for i in range(n):
        for j in range(i + 4, n):
            count = (count + dp[i][j]) % MOD

    return count

# Example usage:
s1 = "103301"
s2 = "0000000"
s3 = "9999900000"
print("Number of palindromic subsequences in s1:", count_palindromic_subsequences(s1))
print("Number of palindromic subsequences in s2:", count_palindromic_subsequences(s2))
print("Number of palindromic subsequences in s3:", count_palindromic_subsequences(s3))
```

7. Find the Pivot Integer

```
class Solution:
    def pivotInteger(self, n: int) -> int:
        i=1
        j=n
        res=-1
        if i==j:
            return i
        else:
            s1=0
            s2=0
            while(i<=j):
                if s1<=s2:
                    s1+=i
                    i+=1
                elif s2<=s1:
                    s2+=j
                    j-=1
                if j==i and s1==s2:
                    res=i
            return res
```

8. Append Characters to String to Make Subsequene

```
class Solution:
    def appendCharacters(self, s: str, t: str) -> int:
        i, j = 0, 0

            if s[i] == t[j]:  # If characters match
        j += 1
      i += 1

    return len(t) - j
```

9. Remove Nodes From Linked List

```
   class Solution:
  def removeNodes(self, head: Optional[ListNode]) ->
Optional[ListNode]:
    if not head:
      return None
    node = head
    # Gives next greater node
    nxt_greater = self.removeNodes(node.next)

    node.next = nxt_greater
    if not nxt_greater or node.val >= nxt_greater.val:
      return node
    return nxt_greater
```

10. Count Subarrays With Median K

```
class Solution:
  def countSubarrays(self, nums: List[int], k: int) -> int:
    idx = nums.index(k)
```

```
freq = Counter()
prefix = 0
for i in reversed(range(idx+1)):
    prefix += int(nums[i] > k) - int(nums[i] < k)
    freq[prefix] += 1
ans = prefix = 0
for i in range(idx, len(nums)):
    prefix += int(nums[i] > k) - int(nums[i] < k)
    ans += freq[-prefix] + freq[-prefix+1]
return ans
```