1. Maximum XOR of Two Non-Overlapping Subtrees

```
From collections import defaultdict

Def maximumXOR(n, edges, values):
  Tree = defaultdict(list)
  For a, b in edges:
    Tree[a].append(b)
    Tree[b].append(a)

  Subtree_sums = [0] * n
  Visited = [False] * n

  Def dfs(node):
    Visited[node] = True
    Subtree_sum = values[node]
    For neighbor in tree[node]:
      If not visited[neighbor]:
        Subtree_sum += dfs(neighbor)
    Subtree_sums[node] = subtree_sum
    Return subtree_sum

  Dfs(0)
  Def find_max_XOR(subtree_sums):
    Max_xor = 0
    N = len(subtree_sums)
    For I in range(n):
      For j in range(I + 1, n):
```

```
        Max_xor = max(max_xor, subtree_sums[i] ^ subtree_sums[j])

    Return max_xor
 Max_xor = find_max_XOR(subtree_sums[1:]
Return max_xor
N = 6
Edges = [[0,1],[0,2],[1,3],[1,4],[2,5]]
Values = [2,8,3,6,2,5]
Print(maximumXOR(n, edges, values))
N = 3
Edges = [[0,1],[1,2]]
Values = [4,6,1]
Print(maximumXOR(n, edges, values))
```

2. Form a Chemical Bond

3. Minimum Cuts to Divide a Circle

```
Def minCutsToDivideCircle(n):
  If n == 1:
    Return 0
  Return n if n % 2 != 0 else n // 2
Print(minCutsToDivideCircle(4))
Print(minCutsToDivideCircle(3))
```

4. Difference Between Ones and Zeros in Row and Column

```
Def best_closing_time(customers: str) -> int:
  N = len(customers)
  Penalty_if_closed_now = customers.count('Y')
  Min_penalty = penalty_if_closed_now
```

```
  Best_hour = 0
  Current_penalty = penalty_if_closed_now
  For I in range(n):
    If customers[i] == 'Y':
      Current_penalty -= 1  # One less customer after this hour, so decrement penalty
    Else:
      Current_penalty += 1  # One more no-customer hour if we keep the shop open

    If current_penalty < min_penalty:
      Min_penalty = current_penalty
      Best_hour = I + 1  # Best hour to close is after this hour

  Return best_hour
Print(best_closing_time("YYNY"))  # Output: 2
Print(best_closing_time("NNNNN"))  # Output: 0
Print(best_closing_time("YYYY"))  # Output: 4
```

5. Minimum Penalty for a Shop

```
Def best_closing_time(customers: str) -> int:
  N = len(customers)
  Penalty_if_closed_now = customers.count('Y')
  Min_penalty = penalty_if_closed_now
  Best_hour = 0
  Current_penalty = penalty_if_closed_now

  For I in range(n):
    If customers[i] == 'Y':
```

```
      Current_penalty -= 1  # Reduce penalty as one less 'Y' will be in the closed time
    Else:
      Current_penalty += 1  # Increase penalty as this 'N' will be in the open time

    If current_penalty < min_penalty:
      Min_penalty = current_penalty
      Best_hour = I + 1  # Closing after this hour

  Return best_hour
Print(best_closing_time("YYNY"))  # Output: 2
Print(best_closing_time("NNNNN"))  # Output: 0
Print(best_closing_time("YYYY"))  # Output: 4
```

6. Count Palindromic Subsequences

```
MOD = 10**9 + 7
Def count_palindromic_subsequences(s: str) -> int:
  N = len(s)
  Count1 = [[0] * 10 for _ in range(10)]  # count1[a][b] – counts pairs (a, b)
  Count2 = [[0] * 10 for _ in range(10)]  # count2[a][b] – counts pairs (a, b) to be extended to
(a, b, c, b, a)
  Result = 0
 For I in range(n):
    Digit = int(s[i])
    For a in range(10):
      For b in range(10):
        Result = (result + count2[a][b]) % MOD
```

```
    For a in range(10):
      Count2[a][digit] = (count2[a][digit] + count1[a][digit]) % MOD
    For a in range(10):
      Count1[a][digit] = (count1[a][digit] + 1) % MOD
Return result
Print(count_palindromic_subsequences("12321
```

7. Find the Pivot Integer

```
Import math
Def find_pivot_integer(n):
  A = 2
  B = 1
  C = -n * (n + 1)

  Discriminant = b**2 – 4 * a * c
  If discriminant < 0:
    Return -1

  Sqrt_discriminant = math.isqrt(discriminant)

  If sqrt_discriminant * sqrt_discriminant != discriminant:
    Return -1

  X1 = (-b + sqrt_discriminant) // (2 * a)
  X2 = (-b – sqrt_discriminant) // (2 * a)

  If 1 <= x1 <= n and (x1 * (x1 + 1)) == (n * (n + 1) – (x1 – 1) * x1):
```

```
    Return x1
  If 1 <= x2 <= n and (x2 * (x2 + 1)) == (n * (n + 1) – (x2 – 1) * x2):
    Return x2

  Return -1
Print(find_pivot_integer(8))
Print(find_pivot_integer(1))
Print(find_pivot_integer(4))
```

8. Append Characters to String to Make Subsequene

```
Def append_characters(s: str, t: str) -> int:
  N, m = len(s), len(t)
  I, j = 0, 0

  While I < n and j < m:
    If s[i] == t[j]:
      J += 1
    I += 1

  Return m – j
Print(append_characters("coaching", "coding"))
Print(append_characters("abcde", "a"))
Print(append_characters("z", "abcde"))
```

9. Remove Nodes From Linked List

```
Class ListNode:
  Def __init__(self, val=0, next=None):
    Self.val = val
```

```
        Self.next = next


    Def removeNodes(head: ListNode) -> ListNode:
      Def reverse_list(node: ListNode) -> ListNode:
        Prev = None
        While node:
          Next_node = node.next
          Node.next = prev
          Prev = node
          Node = next_node
        Return prev


      Reversed_head = reverse_list(head)
       Stack = []
      Max_val = float('-inf')
      Current = reversed_head
      While current:
        If current.val >= max_val:
          Stack.append(current)
          Max_val = current.val
        Current = current.next
      Dummy = ListNode(0)
      Tail = dummy
      While stack:
        Node = stack.pop()
        Tail.next = node
```

```
      Tail = tail.next
    Tail.next = None
  Return dummy.next
Def print_list(head: ListNode):
    Current = head
    While current:
      Print(current.val, end=" -> ")
      Current = current.next
    Print("None")
Head = ListNode(5, ListNode(2, ListNode(13, ListNode(3, ListNode(8)))))
Print("Original list:")
Print_list(head)
Modified_head = removeNodes(head)
Print("Modified list:")
Print_list(modified_head)
```

10. Count Subarrays With Median K

```
def count_subarrays_with_median_k(nums, k):
    n = len(nums)
    left = 0
    count = 0

    for right in range(n):
      while right < n and nums[right] < k:
        right += 1
      count += (right - left + 1) // 2
```

```
      while right < n and nums[right] >= k:
        right += 1
        left = right

  return count
print(count_subarrays_with_median_k([3,2,1,4,5], 4))
print(count_subarrays_with_median_k([2,3,1], 3))
```