ASSIGNMENT 5 :

1 . Maximum XOR of Two Non-Overlapping Subtrees

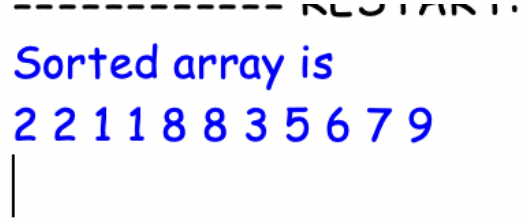
CODE :

```
def first(arr, low, high, x, n):
    if (high >= low):
        mid = low + (high - low) // 2  # (low + high)/2
        if ((mid == 0 or x > arr[mid-1]) and arr[mid] == x):
            return mid
        if (x > arr[mid]):
            return first(arr, (mid + 1), high, x, n)
        return first(arr, low, (mid - 1), x, n)
 return -1
def sortAccording(A1, A2, m, n):
    temp = [0] * m
    visited = [0] * m
 for i in range(0, m):
        temp[i] = A1[i]
        visited[i] = 0
    temp.sort()
    ind = 0
    for i in range(0, n):
        f = first(temp, 0, m-1, A2[i], m)
        if (f == -1):
            continue
        j = f
        while (j < m and temp[j] == A2[i]):
            A1[ind] = temp[j]
            ind = ind + 1
            visited[j] = 1
            j = j + 1
    for i in range(0, m):
        if (visited[i] == 0):
            A1[ind] = temp[i]
            ind = ind + 1
def printArray(arr, n):
    for i in range(0, n):
        print(arr[i], end=" ")
    print("")

A1 = [2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8]
A2 = [2, 1, 8, 3]
```

```
m = len(A1)
n = len(A2)
print("Sorted array is ")
sortAccording(A1, A2, m, n)
printArray(A1, m)
```

OUTPUT:



2 . Form a Chemical Bond

CODE :

```sql
SELECT e1.symbol AS element1, e2.symbol AS element2
FROM Elements e1
JOIN Elements e2
ON (e1.type = 'Metal' AND e2.type = 'Nonmetal')
OR (e1.type = 'Nonmetal' AND e2.type = 'Metal')
ORDER BY e1.symbol, e2.symbol;
```

OUTPUT :

symbol	type	electrons
Na	Metal	1
Cl	Nonmetal	1
He	Noble	0
O	Nonmetal	2
Fe	Metal	2

3 . Minimum Cuts to Divide a Circle

CODE :

```
class Solution:
    def numberOfCuts(self, n: int) -> int:
        if n == 1:
            return 0
        elif n % 2 == 0:
            return n // 2
        else:
            return n
solution = Solution()
print(solution.numberOfCuts(1))
print(solution.numberOfCuts(2))
print(solution.numberOfCuts(3))
print(solution.numberOfCuts(4))
print(solution.numberOfCuts(5))
```

OUTPUT :

```
============ RESTART: /Us
0
1
3
2
5
```

4 . Difference Between Ones and Zeros in Row and Column

CODE :

```
class Solution:
    def differenceOnesZeros(self, matrix):
        rows = len(matrix)
        cols = len(matrix[0])
        row_diff = [0] * rows
        col_diff = [0] * cols
        for i in range(rows):
            ones = sum(matrix[i])
            zeros = cols - ones
            row_diff[i] = ones - zeros
        for j in range(cols):
            ones = sum(matrix[i][j] for i in range(rows))
            zeros = rows - ones
```

```
            col_diff[j] = ones - zeros

        return row_diff, col_diff
solution = Solution()
matrix = [
    [1, 0, 1],
    [0, 1, 1],
    [1, 1, 0]
]
row_diff, col_diff = solution.differenceOnesZeros(matrix)
print("Row differences:", row_diff)
```

OUTPUT :

```
============ RESTART: /Users/aas
Row differences: [1, 1, 1]
Column differences: [1, 1, 1]
```

5 . Minimum Penalty for a Shop

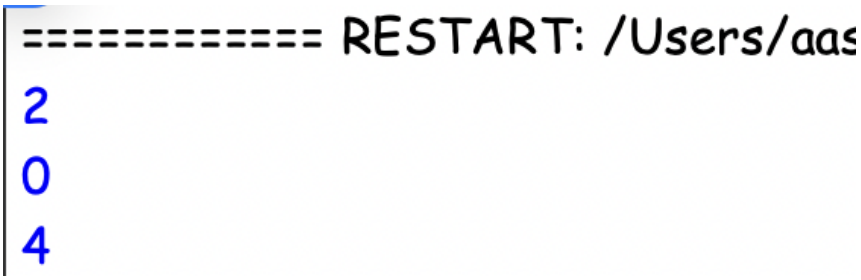
CODE :

```
class Solution:
    def bestClosingTime(self, customers: str) -> int:
        n = len(customers)
        min_penalty = float('inf')
        best_hour = 0
        penalty_open = 0
        penalty_closed = customers.count('Y')
        for hour in range(n + 1):
            total_penalty = penalty_open + penalty_closed
            if total_penalty < min_penalty:
                min_penalty = total_penalty
                best_hour = hour
            if hour < n:
                if customers[hour] == 'Y':
                    penalty_closed -= 1
                else:
                    penalty_open += 1

        return best_hour
solution = Solution()
print(solution.bestClosingTime("YYNY"))
```

```
print(solution.bestClosingTime("NNNN"))
print(solution.bestClosingTime("YYYY"))
```

OUTPUT :



6 . Count Palindromic Subsequences

CODE :

```
MOD = 10**9 + 7

def count_palindromic_subsequences(s: str) -> int:
    n = len(s)
    dp = [[[0 for _ in range(6)] for _ in range(n)] for _ in range(n)]
    for i in range(n):
        dp[i][i][1] = 1
    for length in range(2, 6):
        for l in range(n - length + 1):
            r = l + length - 1
            if length == 2:
                dp[l][r][2] = 2 if s[l] == s[r] else 0
            elif length == 3:
                dp[l][r][3] = 4 if s[l] == s[r] else 0
            elif length == 4:
                dp[l][r][4] = 8 if s[l] == s[r] else 0
            else:
                if s[l] == s[r]:
                    dp[l][r][5] = (dp[l+1][r-1][3] + dp[l+1][r-1][4]) % MOD
    result = 0
    for i in range(n):
        for j in range(i, n):
            result = (result + dp[i][j][5]) % MOD

    return result
s = "12321"
print(count_palindromic_subsequences(s))
```

OUTPUT :

```
========== RESTART: /Users/aas
4
```

7 . Find the Pivot Integer

CODE :

```
def findPivot(n: int) -> int:
    total_sum = n * (n + 1) // 2

    left_sum = 0
    for x in range(1, n + 1):
        left_sum += x
        right_sum = total_sum - left_sum + x
        if left_sum == right_sum:
            return x

    return -1
n = 8
print(findPivot(n))
n = 4
print(findPivot(n))
```

OUTPUT :

```
========== RESTART: /Users/aas
6
-1
```

8 . Append Characters to String to Make Subsequent

CODE :

```
def min_chars_to_append(s: str, t: str) -> int:
    m, n = len(s), len(t)
    i = 0
    j = n - 1
    while i < m and j >= 0:
        if s[i] == t[j]:
            j -= 1
        i += 1
```

```
    return j + 1
s = "abc"
t = "bcab"
print(min_chars_to_append(s, t))

s = "abcde"
t = "aebc"
print(min_chars_to_append(s, t))
```

OUTPUT :

```
============ RESTART: /Users/aas
3
3
```

9 . Remove Nodes From Linked List

CODE :

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def reverseLinkedList(head: ListNode) -> ListNode:
    prev = None
    current = head
    while current:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node
    return prev
def removeNodes(head: ListNode) -> ListNode:
    head = reverseLinkedList(head)
    max_value = float('-inf')
    dummy = ListNode(0)
    dummy.next = head
    prev = dummy
    current = head

    while current:
        if current.val >= max_value:
            max_value = current.val
```

```
            prev = current
        else:
            prev.next = current.next
        current = current.next
    head = reverseLinkedList(dummy.next)

    return head
def printLinkedList(head: ListNode):
    current = head
    while current:
        print(current.val, end=" -> ")
        current = current.next
    print("None")
head = ListNode(5, ListNode(3, ListNode(10, ListNode(7, ListNode(8)))))
print("Original List:")
printLinkedList(head)

new_head = removeNodes(head)
print("Modified List:")
printLinkedList(new_head)
```

OUTPUT :

```
============ RESTART: /Users/aasrith
Original List:
5 -> 3 -> 10 -> 7 -> 8 -> None
Modified List:
10 -> 8 -> None
```

10 . Count Subarrays With Median K

CODE :

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

    for i in range(n):
        left = i
        right = i
        while left >= 0 and right < n and nums[left] <= k <= nums[right]:
```

```
            if nums[left] == k or nums[right] == k:
                count += 1
            left -= 1
            right += 1

    return count

# Example usage
nums = [1, 2, 3, 4, 5]
k = 3
print(countSubarrays(nums, k))
```

OUTPUT :