Day 148/180 Queue Hard Problem

1:Gas Station:

```
class Solution {
public:
    // using Queue
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
        queue<int> q;  // Queue to store possible starting indices
        int gasSum = 0;  // Total gas available
        int costSum = 0;  // Total cost to travel to all stations

        // Calculate total gas and total cost
        for(int i = 0; i < cost.size(); i++) {
            gasSum += gas[i];
            costSum += cost[i];
        }

        // Check if total gas is less than total cost, if so, it's impossible to
complete the circuit
        if(gasSum < costSum) {
            return -1;
        }

        int remain = 0;  // Initialize remaining gas
        for(int i = 0; i < cost.size(); i++) {
            q.push(i);  // Push index into the queue as a possible starting station
            // Check if the cost is greater than the gas available at the station
            if(gas[i] + remain < cost[i]) {
                // Remove all previous stations from the queue
                while(!q.empty()) {
                    q.pop();
                }
                remain = 0;  // Reset remaining gas
            } else {
                remain += gas[i] - cost[i];  // Update remaining gas
            }
```

```
        }

        // Return the index of the starting gas station
        return q.front();
    }
};

Time Complexity :- O(n)
Space Complexity :- O(n)
```

```
class Solution {
public:
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
        int n = gas.size();  // Get the number of gas stations

        int curFuel = 0, ans = 0, fuel = 0;  // Initialize variables for current fuel, starting station index, and total fuel

        for(int i = 0; i < n; i++) {
            curFuel += gas[i];  // Add gas at the current station
            curFuel -= cost[i];  // Subtract the cost to travel to the next station
            fuel += gas[i] - cost[i];  // Calculate the total fuel
            if(curFuel < 0) {  // If current fuel becomes negative
                curFuel = 0;  // Reset current fuel
                ans = i + 1;  // Update the starting station to the next station
            }
        }

        return fuel < 0 ? -1 : ans;  // If total fuel is negative, return -1 (cannot complete circuit), otherwise return the starting station
    }
};

Time Complexity :- O(n)
Space Complexity :- O(1)
```

2:Count Subarrays With Fixed Bounds:

```
class Solution {
public:
    long long countSubarrays(vector<int>& nums, int minK, int maxK) {
        int n = nums.size();  // Get the size of the input array
        // Initialize two arrays to store the indices of the minimum and maximum
values
        // Set initial values to n (size of nums) to indicate that no min/max value
has been found yet
        vector<int> mn(n + 1, n), mx(n + 1, n);

        // Iterate over the input array backwards to find the indices of the
minimum and maximum values
        for (int i = n - 1; i >= 0; i -= 1) {
            // Update mn and mx arrays based on the current element
            mn[i] = mn[i + 1];  // Set mn[i] to the next element in the array
            mx[i] = mx[i + 1];  // Set mx[i] to the next element in the array
            // If the current element is equal to minK, update mn[i] to the current
index
            if (nums[i] == minK) mn[i] = i;
            // If the current element is equal to maxK, update mx[i] to the current
index
            if (nums[i] == maxK) mx[i] = i;
        }

        // Initialize a variable to store the final count of subarrays
        long long ans = 0;
        // Iterate over the input array to count the subarrays that meet the
condition
        for (int i = 0, j = 0; i < n; i += 1) {
            // Check if the current element is within the range [minK, maxK]
            if (nums[i] >= minK and nums[i] <= maxK) {
                j = max(j, i);  // Update j to the maximum of j and i to avoid
unnecessary iterations
                // Iterate until j reaches the end of the array or finds an element
outside the range [minK, maxK]
```

```
                while (j < n and nums[j] >= minK and nums[j] <= maxK) j += 1;
                // If both mn[i] and mx[i] are less than j, it means there is a
valid subarray starting from i
                if (mn[i] < j and mx[i] < j) {
                    // Add the count of subarrays starting from i and ending at j-1
to the answer
                    ans += j - max(mn[i], mx[i]);
                }
            }
        }
        return ans;  // Return the final count of subarrays
    }
};

Time complexity - O(n)
Space Complexity - O(n)
```