Prefix Sum Patterns & Easy Identification Cheat Sheet with Pseudocode

---

1️⃣Key Patterns That Signal Prefix Sum

A. Range Sum / Query Problems

- **Pattern:** Calculate sum of a subarray `[i, j]` repeatedly.
- **Clues:** “sum of elements in range,” “query sum,” “total,” “interval sum.”
- **Formula:** `prefix[i] = sum(arr[0..i])`, then `sum[i..j] = prefix[j] - prefix[i-1]`
- **Pseudocode:**

```
prefix[0] = arr[0]
for i = 1 to n-1:
    prefix[i] = prefix[i-1] + arr[i]
rangeSum(L, R) = prefix[R] - (prefix[L-1] if L>0 else 0)
```

B. Subarray With Target Sum

- **Pattern:** Count subarrays with sum = k.
- **Clues:** “subarray sum,” “number of subarrays,” “equals k.”
- **Approach:** Prefix sum + hashmap
- **Pseudocode:**

```
prefixSum = 0
map = {0:1}  // sum 0 occurs once
count = 0
for num in arr:
    prefixSum += num
    if prefixSum - k in map:
        count += map[prefixSum - k]
    map[prefixSum] = map.get(prefixSum, 0) + 1
return count
```

C. Continuous Difference / Balance Problems

- **Pattern:** Differences or balance in array or string.
- **Clues:** “difference,” “equilibrium,” “balance,” “pivot.”
- **Pseudocode:**

```
map = {0:-1}  // prefix sum -> index
prefixSum = 0
for i in 0..n-1:
    val = transform(arr[i])  // e.g., 0->-1, 1->1
    prefixSum += val
    if prefixSum in map:
        update answer using map[prefixSum] and i
    else:
        map[prefixSum] = i
```

D. 2D / Matrix Sum Problems

- **Pattern:** Compute sum of submatrix quickly.
- **Pseudocode:**

```
prefix[i][j] = matrix[i][j] + prefix[i-1][j] + prefix[i][j-1] - prefix[i-1]
[j-1]
submatrixSum(x1, y1, x2, y2) = prefix[x2][y2] - prefix[x1-1][y2] -
prefix[x2][y1-1] + prefix[x1-1][y1-1]
```

E. Frequency / Counting Problems

- **Pattern:** Counting occurrences in ranges.
- **Pseudocode:**

```
for val in all_possible_values:
    freq[0][val] = (arr[0] == val ? 1 : 0)
for i = 1 to n-1:
    for val in all_possible_values:
        freq[i][val] = freq[i-1][val] + (arr[i] == val ? 1 : 0)
query(L, R, val) = freq[R][val] - (freq[L-1][val] if L>0 else 0)
```

F. XOR / Bitwise Prefix

- **Pattern:** XOR subarrays or cumulative XOR.
- **Pseudocode:**

```
prefixXOR[0] = arr[0]
for i = 1 to n-1:
    prefixXOR[i] = prefixXOR[i-1] ^ arr[i]
subarrayXOR(L, R) = prefixXOR[R] ^ (prefixXOR[L-1] if L>0 else 0)
```

G. Sliding Window / Fixed Size Subarray

- **Pattern:** Maximum / minimum / sum in a window of size k.
- **Pseudocode:**

```
windowSum[i..i+k-1] = prefix[i+k-1] - (prefix[i-1] if i>0 else 0)
```

2️⃣Easy Method to Identify Prefix Sum Problems

Step	Question to Ask	Action
1	Need sum/total over range?	Use prefix sum
2	Count subarrays meeting sum/condition?	Prefix sum + hashmap
3	Matrix & range queries?	Use 2D prefix sum
4	Running balance/difference problem?	Map values to +1/-1, use prefix sum
5	Frequency/occurrences in range?	Prefix frequency array
6	XOR/bitwise accumulation?	Prefix XOR
7	Sliding window sums efficiently?	Prefix sum for window

3️⃣Quick Tips / Tricks

- Precompute prefix array once to avoid recomputation.
- Extend 1D prefix sum logic to 2D matrices.
- Combine **hashmap + prefix sum** for counting subarrays.
- Transform array for balance or modular constraints.
- Sliding window + prefix sum reduces O(k) computation per window.

Summary: Look for cumulative info, ranges, subarrays, frequency, matrix sums, XOR, or sliding windows — prefix sum is likely the right tool.