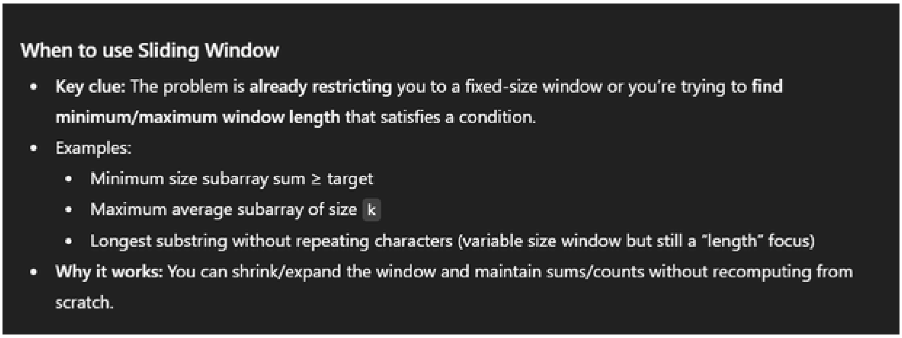
SUBARRAYS PATTERNS

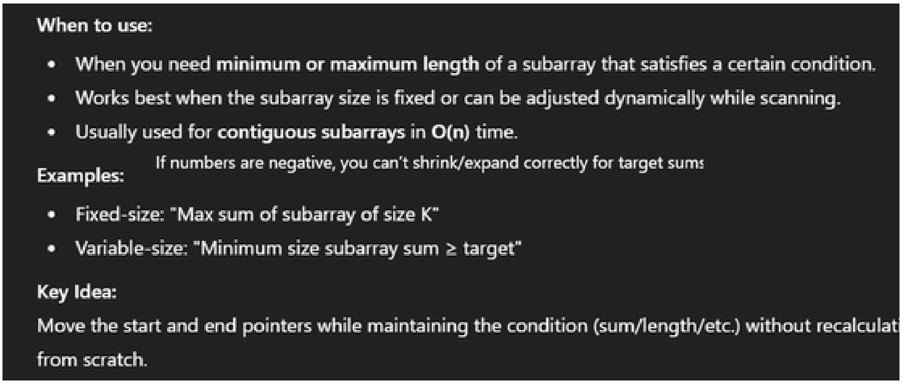
1)SLIDING WINDOW



When to use Sliding Window

- **Key clue:** The problem is **already restricting** you to a fixed-size window or you're trying to **find minimum/maximum window length** that satisfies a condition.
- Examples:
  - Minimum size subarray sum ≥ target
  - Maximum average subarray of size `k`
  - Longest substring without repeating characters (variable size window but still a "length" focus)
- **Why it works:** You can shrink/expand the window and maintain sums/counts without recomputing from scratch.

another view



When to use:

- When you need **minimum or maximum length** of a subarray that satisfies a certain condition.
- Works best when the subarray size is fixed or can be adjusted dynamically while scanning.
- Usually used for **contiguous subarrays** in **O(n)** time.

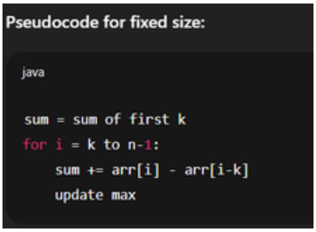
If numbers are negative, you can't shrink/expand correctly for target sums

Examples:

- Fixed-size: "Max sum of subarray of size K"
- Variable-size: "Minimum size subarray sum ≥ target"

Key Idea:

Move the start and end pointers while maintaining the condition (sum/length/etc.) without recalculati from scratch.



Pseudocode for fixed size:

```java
sum = sum of first k
for i = k to n-1:
    sum += arr[i] - arr[i-k]
    update max
```

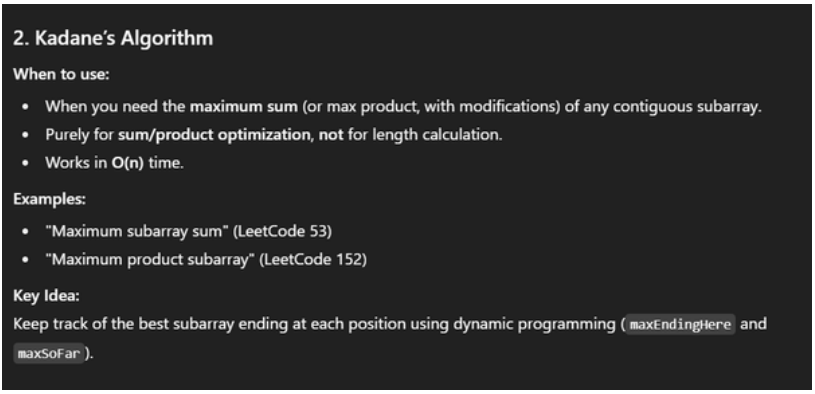
SOME QUESTIONS-

1)Maximum Average Subarray I (Fixed Size)

2)Minimum Size Subarray Sum (Variable Size)

3)Longest Substring Without Repeating Characters

2) Kadane’s Algorithm



2. Kadane’s Algorithm

When to use:

- When you need the **maximum sum** (or max product, with modifications) of any contiguous subarray.
- Purely for **sum/product optimization**, **not** for length calculation.
- Works in **O(n)** time.

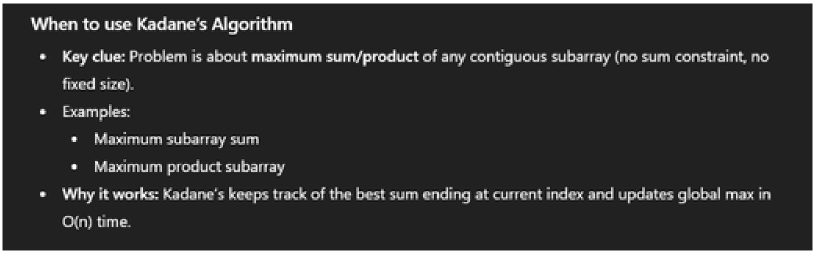
Examples:

- "Maximum subarray sum" (LeetCode 53)
- "Maximum product subarray" (LeetCode 152)

Key Idea:

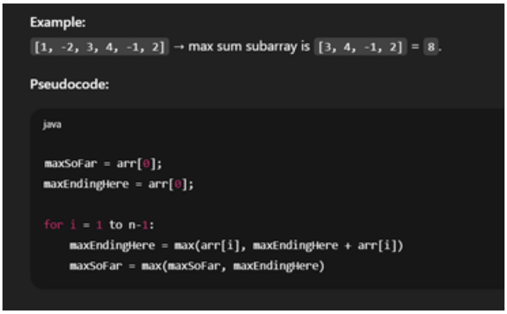
Keep track of the best subarray ending at each position using dynamic programming (`maxEndingHere` and `maxSoFar`).

another view



When to use Kadane’s Algorithm

- **Key clue:** Problem is about **maximum sum/product** of any contiguous subarray (no sum constraint, no fixed size).
- Examples:
  - Maximum subarray sum
  - Maximum product subarray
- **Why it works:** Kadane’s keeps track of the best sum ending at current index and updates global max in O(n) time.



Example:

`[1, -2, 3, 4, -1, 2]` → max sum subarray is `[3, 4, -1, 2]` = `8`.

Pseudocode:

```java
maxSoFar = arr[0];
maxEndingHere = arr[0];

for i = 1 to n-1:
    maxEndingHere = max(arr[i], maxEndingHere + arr[i])
    maxSoFar = max(maxSoFar, maxEndingHere)
```

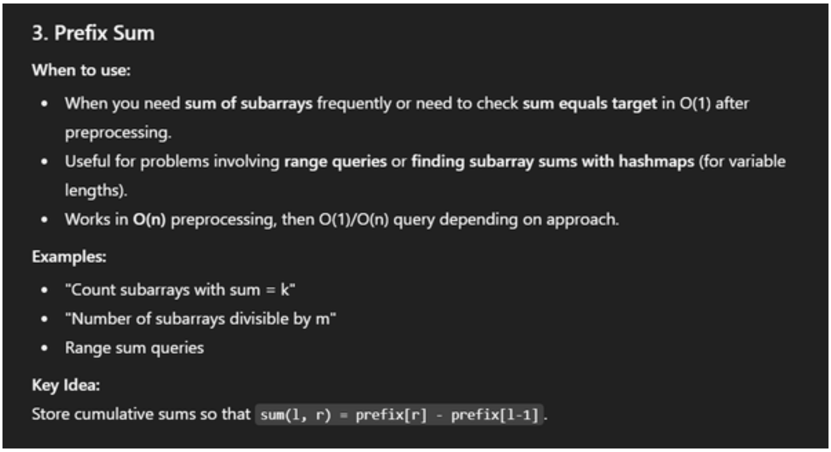
SOME QUESTIONS-

1)Maximum Subarray

2)Maximum Sum Circular Subarray

3)Maximum Product Subarray (Kadane’s variant for product)

3) Prefix Sum



3. Prefix Sum

When to use:

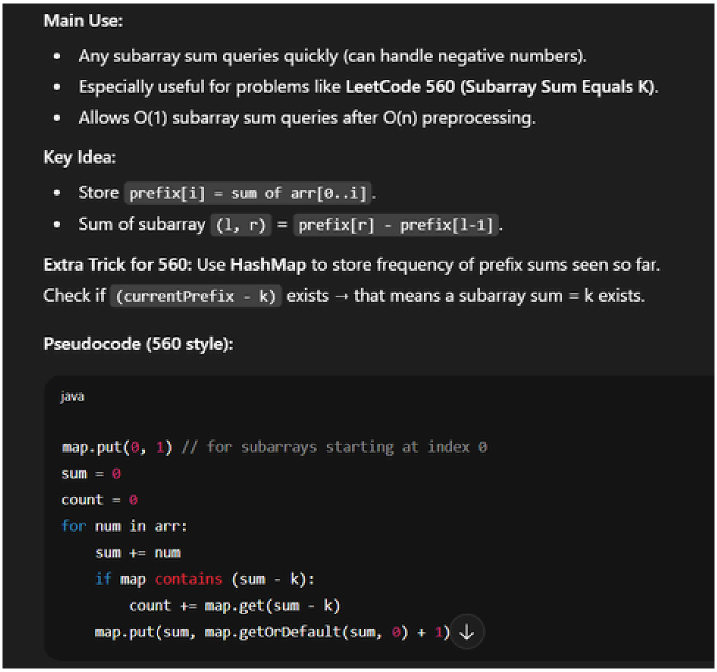
- When you need **sum of subarrays** frequently or need to check **sum equals target** in O(1) after preprocessing.
- Useful for problems involving **range queries** or **finding subarray sums with hashmaps** (for variable lengths).
- Works in **O(n)** preprocessing, then O(1)/O(n) query depending on approach.

Examples:

- "Count subarrays with sum = k"
- "Number of subarrays divisible by m"
- Range sum queries

Key Idea:

Store cumulative sums so that `sum(l, r) = prefix[r] - prefix[l-1]`.



Main Use:

- Any subarray sum queries quickly (can handle negative numbers).
- Especially useful for problems like **LeetCode 560 (Subarray Sum Equals K)**.
- Allows O(1) subarray sum queries after O(n) preprocessing.

Key Idea:

- Store `prefix[i] = sum of arr[0..i]`.
- Sum of subarray `(l, r)` = `prefix[r] - prefix[l-1]`.

Extra Trick for 560: Use HashMap to store frequency of prefix sums seen so far. Check if `(currentPrefix - k)` exists → that means a subarray sum = k exists.

Pseudocode (560 style):

java

```java
map.put(0, 1) // for subarrays starting at index 0
sum = 0
count = 0
for num in arr:
    sum += num
    if map contains (sum - k):
        count += map.get(sum - k)
    map.put(sum, map.getOrDefault(sum, 0) + 1)
```

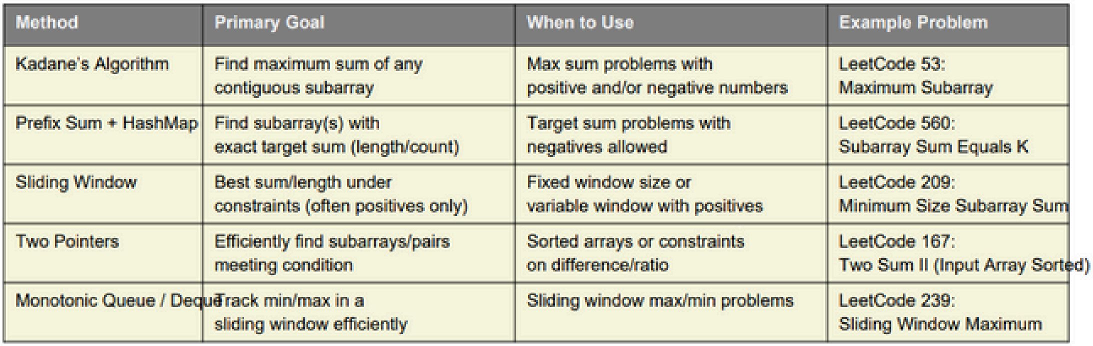
SOME QUESTIONS-

1) Subarray Sum Equals K

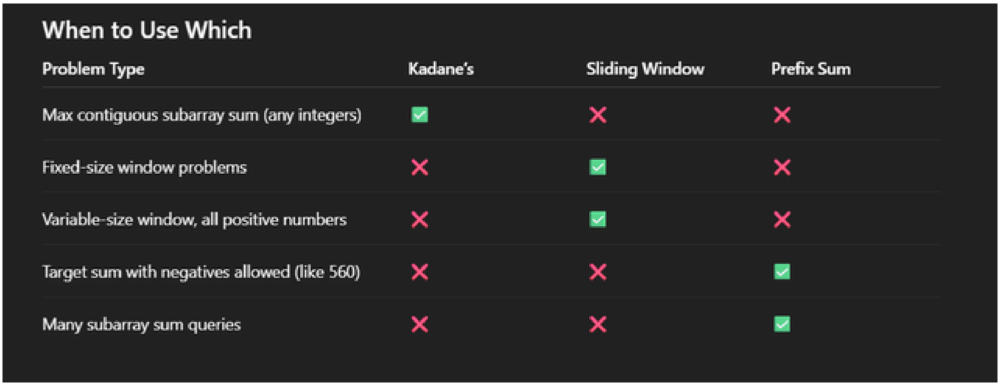
2) Continuous Subarray Sum

3) Range Sum Query - Immutable

Comparison of Subarray Techniques



Method	Primary Goal	When to Use	Example Problem
Kadane's Algorithm	Find maximum sum of any contiguous subarray	Max sum problems with positive and/or negative numbers	LeetCode 53: Maximum Subarray
Prefix Sum + HashMap	Find subarray(s) with exact target sum (length/count)	Target sum problems with negatives allowed	LeetCode 560: Subarray Sum Equals K
Sliding Window	Best sum/length under constraints (often positives only)	Fixed window size or variable window with positives	LeetCode 209: Minimum Size Subarray Sum
Two Pointers	Efficiently find subarrays/pairs meeting condition	Sorted arrays or constraints on difference/ratio	LeetCode 167: Two Sum II (Input Array Sorted)
Monotonic Queue / Deque	Track min/max in a sliding window efficiently	Sliding window max/min problems	LeetCode 239: Sliding Window Maximum



When to Use Which

Problem Type	Kadane's	Sliding Window	Prefix Sum
Max contiguous subarray sum (any integers)	✅	❌	❌
Fixed-size window problems	❌	✅	❌
Variable-size window, all positive numbers	❌	✅	❌
Target sum with negatives allowed (like 560)	❌	❌	✅
Many subarray sum queries	❌	❌	✅