Monotonic Queue — Coding Interview Notes (Light Theme)

General Pattern Template

```
from collections import deque

def fn(arr, k):
    dq = deque()  # will store indices; maintain monotonicity by values
    ans = []

    for right, x in enumerate(arr):
        # Pop from back while it violates monotonic property (for max: pop smaller)
        while dq and arr[dq[-1]] <= x:  # for min queue, flip <= to >=
            dq.pop()

        dq.append(right)

        # Remove out-of-window indices from front
        if dq[0] <= right - k:
            dq.popleft()

        # Record answer when window of size k has formed
        if right + 1 >= k:
            ans.append(arr[dq[0]])  # max in window (for min, use min-queue rule)

    return ans
```

Concept:
A **monotonic queue** (deque) maintains window candidates in sorted order by value while sliding a window. It supports O(1) amortized push/pop and O(1) query for the window's min/max.

Time Complexity: O(n) for all windows — each index enters and leaves the deque once.

Key Ideas

1. Store indices, compare by values; pop from back while invariant breaks.
2. Drop indices that fall out of the window from the front.
3. Use ≤/≥ carefully to avoid duplicate outdated candidates.
4. Works for both max-queue and min-queue by flipping inequalities.

Example 1: Sliding Window Maximum

Goal: For each window of size k, output the maximum.
Approach: Maintain a decreasing deque (front is max).

```
from collections import deque

def sliding_window_max(nums, k):
    dq = deque()
    ans = []
    for i, x in enumerate(nums):
        while dq and nums[dq[-1]] <= x:
            dq.pop()
        dq.append(i)
        if dq[0] <= i - k:
            dq.popleft()
        if i + 1 >= k:
            ans.append(nums[dq[0]])
    return ans
```

Example 2: Shortest Subarray with Sum ≥ K

Goal: Find the shortest subarray with sum ≥ K (handles negatives).
Approach: Use prefix sums and maintain an *increasing* deque of prefix indices.

```
from collections import deque

def shortest_subarray_at_least_k(nums, K):
    n = len(nums)
    prefix = [0]
    for x in nums:
        prefix.append(prefix[-1] + x)

    ans = n + 1
    dq = deque()

    for i, s in enumerate(prefix):
        # Try to satisfy sum >= K
        while dq and s - prefix[dq[0]] >= K:
            ans = min(ans, i - dq.popleft())

        # Maintain increasing deque of prefix sums
        while dq and prefix[dq[-1]] >= s:
            dq.pop()

        dq.append(i)

    return ans if ans <= n else -1
```

Example 3: First Negative Number in Every Window (Size k)

Goal: Report the first negative number for each window.
Approach: Maintain a deque of indices of negative numbers; pop from front when out of window.

```
from collections import deque

def first_negative_in_windows(nums, k):
    dq = deque()
    ans = []

    for i, x in enumerate(nums):
        if x < 0:
            dq.append(i)
        # remove out-of-window from front
        while dq and dq[0] <= i - k:
            dq.popleft()
        if i + 1 >= k:
            ans.append(nums[dq[0]] if dq else 0)  # 0 if none present
    return ans
```

Summary Table

ProblemDeque InvariantAnswer per WindowComplexity Sliding window maximumValues decreasingFront = maxO(n) Shortest subarray ≥ KPrefix sums increasingShrink from frontO(n) First negative per windowIndices of negativesFront = first negativeO(n)