Two Pointers (Opposite Ends) Pattern — Coding Interview Notes

General Pattern Template

```
def fn(arr):
    left = ans = 0
    right = len(arr) - 1

    while left < right:
        # Do some logic involving arr[left], arr[right]
        if CONDITION:
            left += 1
        else:
            right -= 1

    return ans
```

Concept:
The **Two Pointers (Opposite Ends)** pattern initializes two pointers — *left* and *right* — at opposite ends of a list or string. They move toward each other based on specific conditions. This pattern helps solve many array and string problems efficiently in **O(n)** time.

Key Ideas

1 Works best when the array is sorted or symmetric.

2 Each iteration moves at least one pointer → guarantees O(n) time.

3 Use `while left < right:` as loop condition.

4 Move the pointer that helps reach the goal faster (increase or decrease value).

Example 1: Two Sum II (Sorted Input)

Goal: Find two numbers in a sorted array that sum to a target.
Approach: Move the left pointer right if sum is too small, right pointer left if sum is too large.

```
def two_sum_sorted(nums, target):
    left, right = 0, len(nums) - 1
    while left < right:
        s = nums[left] + nums[right]
        if s == target:
            return [left, right]
        elif s < target:
            left += 1  # Need a larger sum
        else:
            right -= 1  # Need a smaller sum
```

Example 2: Valid Palindrome

Goal: Check if a string reads the same backward and forward.
Approach: Compare characters from both ends and move pointers inward.

```
def is_palindrome(s):
    s = ''.join(ch.lower() for ch in s if ch.isalnum())
    left, right = 0, len(s) - 1

    while left < right:
        if s[left] != s[right]:
            return False
        left += 1
        right -= 1
    return True
```

Example 3: Container With Most Water

Goal: Find two lines that together with the x-axis form a container that holds the most water.
Approach: Move the pointer pointing to the shorter line.

```
def max_area(height):
    left, right = 0, len(height) - 1
    ans = 0

    while left < right:
        width = right - left
        h = min(height[left], height[right])
        ans = max(ans, width * h)

        if height[left] < height[right]:
            left += 1
        else:
            right -= 1

    return ans
```

Summary Table

ConceptExampleComplexity Find target pair in sorted arrayTwo Sum IIO(n) Check mirrored propertyPalindromeO(n) Optimize area/productContainer With Most WaterO(n)