Two Pointers (Two Inputs, Exhaust Both) — Coding Interview Notes

General Pattern Template

```
def fn(arr1, arr2):
    i = j = ans = 0

    while i < len(arr1) and j < len(arr2):
        # do some logic here
        if CONDITION:
            i += 1
        else:
            j += 1

    while i < len(arr1):
        # do logic
        i += 1

    while j < len(arr2):
        # do logic
        j += 1

    return ans
```

Concept:
The **Two Pointers (Two Inputs, Exhaust Both)** pattern is used when we need to process or compare **two sorted inputs** — typically arrays, lists, or strings — at the same time. Each pointer moves independently based on logical comparisons, ensuring both inputs are fully traversed.

Time Complexity: O(n + m) where n and m are lengths of the inputs.
Use Cases: Merging, intersection, union, and comparing sorted data streams.

Key Ideas

1. Both inputs are traversed completely (fully exhausted).
2. Best suited for sorted or ordered sequences.
3. Each pointer moves based on comparison logic.
4. Post-processing loops handle remaining elements in either list.

Example 1: Merge Two Sorted Arrays

Goal: Merge two sorted arrays into one sorted array.
Approach: Compare the current elements of both arrays, pick the smaller one, and move that pointer

forward. Then append remaining elements.

```
def merge_sorted(arr1, arr2):
    i = j = 0
    merged = []

    while i < len(arr1) and j < len(arr2):
        if arr1[i] < arr2[j]:
            merged.append(arr1[i])
            i += 1
        else:
            merged.append(arr2[j])
            j += 1

    while i < len(arr1):
        merged.append(arr1[i])
        i += 1

    while j < len(arr2):
        merged.append(arr2[j])
        j += 1

    return merged
```

Example 2: Intersection of Two Sorted Arrays

Goal: Return elements present in both arrays.
Approach: If elements match, add to result and move both pointers; otherwise, move the pointer pointing to the smaller element.

```
def intersection_sorted(arr1, arr2):
    i = j = 0
    result = []

    while i < len(arr1) and j < len(arr2):
        if arr1[i] == arr2[j]:
            result.append(arr1[i])
            i += 1
            j += 1
        elif arr1[i] < arr2[j]:
            i += 1
        else:
            j += 1

    return result
```

Example 3: Compare Strings (Find Common Characters)

Goal: Given two sorted strings (e.g., from sorted letters), find common characters.
Approach: Use the same two-pointer logic to find characters that appear in both strings.

```
def common_chars(str1, str2):
    i = j = 0
    result = []

    while i < len(str1) and j < len(str2):
        if str1[i] == str2[j]:
            result.append(str1[i])
            i += 1
            j += 1
        elif str1[i] < str2[j]:
            i += 1
        else:
            j += 1

    return ''.join(result)
```

Summary Table

ConceptExampleComplexity Merging two sorted sequencesMerge Two Sorted ArraysO(n + m) Finding common elementsIntersection of Two Sorted ArraysO(n + m) Character comparisonCompare Strings for Common CharactersO(n + m)