📘 DAY 3 — TWO POINTER PATTERN (NOTES)

⭐ 1. Two Pointer Pattern – Basic Idea

🔷 Definition:

Two pointers = ek hi array/string/list par **two indices** ko move karte hue optimized solution निकालना.

🔷 Generally Used In:

- **Sorted Arrays**
- **Strings**
- **Linked Lists**
- **Linear search patterns**

🔷 Why Two Pointers?

- Brute force O(N²) hota hai
- Two pointer se O(N) ya O(N logN) me ho jata hai

⭐ 2. Very Important Rule (Core of Two Pointers)

👉 If sum < target → left++

→ Kyunki left pointer small value par hota hai → agar chota sum banna hai to left badhao.

👉 If sum > target → right--

→ Kyunki right pointer large value par hota hai → agar sum zyada aa raha hai to right reduce karo.

👉 If sum == target → answer mil gaya

💡 यह पूरा logic इस बात से आता है कि array sorted है.

⭐⭐⭐ PROBLEM 1: Merge Sorted Array

(LeetCode 88)

📌 Problem Summary

- nums1 me m elements actual
- Last me n zeros diye gaye as space
- nums2 me n elements
- Final sorted merged array → **nums1 me hi store karna hai**

🧠 Key Insight

Agar hum **front se merge** karenge → important elements overwrite ho जाएंगे → shifting O(N²).

✔ Correct solution → merge from back

📝 Algorithm (Backwards Two Pointers)

Initialize:

```ini
i = m-1        // nums1 valid part ka end
j = n-1        // nums2 ka end
k = m+n-1      // final position (nums1 end)
```

Loop:

```lua
while(i >= 0 && j >= 0):
    if nums1[i] > nums2[j]:
        nums1[k] = nums1[i]
        i--
    else:
        nums1[k] = nums2[j]
        j--
    k--
```

After loop:

```lua
while j >= 0:
    nums1[k] = nums2[j]
    j--, k--
```

(i wale ko copy karne ki zarurat nahi hoti because woh already correct order me hota hai)

📊 Dry Run (Important)

nums1 = [1,2,3,0,0,0]
nums2 = [2,5,6]
Pointers:

i	j	k	Compare	Action
2	2	5	3 vs 6	nums1[5]=6, j--

2	1	4	3 vs 5	nums1[4]=5, j--
2	0	3	3 vs 2	nums1[3]=3, i--
1	0	2	2 vs 2	nums1[2]=2, j--
1	-	1	loop ends	

⏱️ **Time Complexity: O(m+n)**

🧠 **Space Complexity: O(1)**

⭐⭐⭐ PROBLEM 2: Count Pairs Whose Sum < Target

(LeetCode 2824)

📌 Problem Summary

Count number of pairs (i, j) such that:

```
css
```

Copy code

```
nums[i] + nums[j] < target
i < j
```

🧠 Key Idea

1. **Sort array**
2. Two pointers:
   - left = 0
   - right = n-1
3. Check sum:

✔ If nums[left] + nums[right] < target

→ मतलब left ke saath **jitne bhi elements left+1...right** hain → **sab valid pairs**

Hence:

```
sql
```

Copy code

```
count += (right - left)
left++
```

✔ Else (sum >= target)

```sql
right--
```

🧠 Why “right - left”? (Most Important Concept)

Example:
Sorted nums = [0,1,2,3,4]
target = 5
Check:

left=0, right=4 → 0+4=4 < 5
So valid pairs:

(0,1)
(0,2)
(0,3)
(0,4)
Count = right - left = 4
Because array sorted hai → agar 0+4 < 5,
toh 0 + sab chote values < 5 hoga × guaranteed.

🧪 Dry Run

nums = [-1,1,2,3,1], target = 2
Sorted = [-1,1,1,2,3]
Process karo wohi two pointer se.

⏱ Time Complexity: O(n log n) (sorting)

🧠 Space: O(1)

⭐⭐⭐ PROBLEM 3: Two Sum

(LeetCode 1)

📌 Goal

Find indices i, j such that:

nums[i] + nums[j] = target

🧠 Optimal Approach: Hashmap

Rule:

- Traverse array
- For each num:

- check target - num in map
- If found → answer
- Else → store num with its index

📝 Pseudo Code

📝 Pseudo Code (C++ Style) for Two Sum Using Hashmap

cpp Copy code

```cpp
unordered_map<int, int> mp;

for (int i = 0; i < n; i++) {

    int need = target - nums[i];

    if (mp.find(need) != mp.end()) {
        return { mp[need], i };
    }

    mp[nums[i]] = i;
}
```

⏱️ Time: O(n)
🧠 Space: O(n)

⭐⭐⭐ PROBLEM 4. TWO SUM II – Sorted Array (LeetCode 167)

📌 Problem Summary
Sorted array diya hua hai.
Find two numbers such that:

nums[i] + nums[j] = target
Return 1-based indices.

⭐ 1. Pointer Initialization

```ini
i = 0
j = nums.size() - 1
```

⭐ 2. Why i < j (not <=)?

- Because **we need two distinct elements**.
- Agar i == j hua → same element ko use kar loge → invalid.
- Isliye strictly:

```scss
while(i < j)
```

⭐ 3. Logic (Sorting ke property se):

👉 If nums[i] + nums[j] > target

→ Right wala element bada hai → kabhi pair nahi banayega.
→ Move j--.

👉 If nums[i] + nums[j] < target

→ Left chhota hai → increase sum.
→ Move i++.

👉 If nums[i] + nums[j] == target

→ Answer mil gaya
→ return {i+1, j+1}

⭐ 4. C++ Code

```cpp
vector<int> twoSumII(vector<int>& numbers, int target) {
    int i = 0, j = numbers.size() - 1;

    while(i < j) {
        int sum = numbers[i] + numbers[j];

        if(sum == target) {
            return {i+1, j+1};
        }
        else if(sum > target) {
            j--;
        }
        else {
            i++;
        }
    }
    return {-1,-1}; // just in case
}
```

⭐ 5. Time Complexity

```scss
O(n)
Only two pointers move at most n steps total.
```

⭐ Space Complexity

```scss
O(1)
```

⭐⭐⭐ PROBLEM 5: 3Sum — Brute force → Optimized

(LeetCode 15)

⭐ 1 Problem Summary

Find all **unique triplets** such that:

nums[i] + nums[j] + nums[k] = 0
i, j, k must be distinct
Return list of triplets (values, not indices).
Duplicate triplets NOT allowed.

⭐ 2 Brute Force (Only for understanding)

Three loops:

```
for(i)
 for(j=i+1)
  for(k=j+1)
```

Check sum.

❌ Time = O(n³) too slow.

Optimal Two-Pointer Approach (Very Important)

STEP 1: Sort the array

Sorted array → two pointer logic apply hoga.

```
sort(nums.begin(), nums.end());
```

STEP 2: Fix first element f

```
for(int f = 0; f < n; f++)
```

But **skip duplicates**:

```
if(f > 0 && nums[f] == nums[f-1])
    continue;
```

STEP 3: Now apply Two Pointer between f+1 and n-1

```
i = f+1
j = n-1
```

⭐ Two Pointer Condition inside 3Sum

Calculate:

```
sum = nums[f] + nums[i] + nums[j]
```

Case A → If sum < 0

→ Need bigger value → i++

Case B → If sum > 0

→ Need smaller value → j--

Case C → If sum == 0

→ Triplet found

→ Push {nums[f], nums[i], nums[j]}
→ Now skip duplicates:

while(i < j && nums[i] == nums[i-1]) i++;
while(i < j && nums[j] == nums[j+1]) j--;
Then do:

i++;
j--;

⭐⭐⭐ (MOST IMPORTANT EXPLANATION)
Why duplicate skip required?
Example:

-1, 0, 1, 1
If we don't skip duplicate 1,
we will again form:

-1 0 1
Duplicate triplet → wrong.

⭐ 4 C++ Code (Final Clean Version)

cpp

Copy code

```cpp
vector<vector<int>> threeSum(vector<int>& nums) {
    vector<vector<int>> ans;
    sort(nums.begin(), nums.end());
    int n = nums.size();

    for(int f = 0; f < n; f++) {

        // Stop early: if nums[f] > 0 → sum can never be 0
        if(nums[f] > 0) break;

        // Skip duplicates of first element
        if(f > 0 && nums[f] == nums[f-1]) continue;

        int i = f + 1;
        int j = n - 1;

        while(i < j) {
            long long sum = (long long)nums[f] + nums[i] + nums[j];

            if(sum == 0) {
                ans.push_back({nums[f], nums[i], nums[j]});

                i++;
                j--;

                // skip duplicates of i
                while(i < j && nums[i] == nums[i-1]) i++;

                // skip duplicates of j
                while(i < j && nums[j] == nums[j+1]) j--;
            }
            else if(sum < 0) {
                i++;
            }
            else {
                j--;
            }
        }
    }
    return ans;
```

⭐ 5 Time & Space Complexity

Time

```mathematica
Sorting = O(n log n)
Outer loop = O(n)
Inner two pointer = O(n)

Total = O(n²)
```

Space

```scss
O(1) (ignoring output)
```