Session - 5

Two Pointers

CONTENTS

01

Introduction to Two Pointers

Introduction to Two Pointers

What :-

- An algorithmic approach that uses two pointers to traverse data structures efficiently.
- Helps in reducing time complexity compared to brute-force methods.
- Commonly used for problems involving arrays, linked lists, and searching algorithms.

Why :-

- Reduces time complexity in many scenarios.
- Avoids nested loops in certain problems.
- Works well with sorted data or specific constraints.
- Common in problems like:
  - Pair sum
  - Reverse arrays
  - Palindrome check
  - Merging sorted arrays

Introduction to Two Pointers

When to Use :-

- When dealing with sorted arrays or linked lists.
- When searching for pairs, subarrays, or specific conditions.
- When a nested loop ($O(N^2)$) solution can be optimized to $O(N)$.

How It Works?

- Two pointers are initialized at different positions in the data structure.
- They move according to specific conditions to optimize problem-solving.
- Helps in minimizing redundant computations by avoiding nested loops.

02

Types of Two Pointer Approach

Types of Two Pointer Approach

1. Opposite Direction

- Start one pointer at the **beginning** (index 0).
- Start the other at the **end** (index input.length - 1).
- Move them **towards each other** until they meet.

Use Cases :-

- Checking for palindromes
- Finding a pair with a given sum in a sorted array
- Reversing arrays

Optimization Tips:

- Avoid unnecessary iterations
- Can often replace nested loops with linear time

Types of Two Pointer Approach

2. Same Direction

- Both pointers **start at the same point** (or nearby) and **move forward** together.
- Useful for maintaining **a sliding window**, **range**, or **subsequence**

Use Cases:

- Longest substring with unique characters
- Merging two sorted arrays
- Finding subarrays with a specific sum

Patterns:

- Expand `right` to grow window
- Contract `left` when constraints are broken
- Track max/min/length/etc. on the go

Dual Array Traversal

- Use two pointers on **two sorted arrays**
- Advance based on element comparison

When to Use :-

- Arrays are **sorted** or can be sorted.
- You need to **compare** or **combine** elements from both arrays.
- Avoid **$O(n^2)$** time by replacing nested iterations.

Benefits :-

- Linear time: **$O(n + m)$**
- Memory-efficient: no extra nested iterations
- Works great with pre-sorted data
- Can be combined with **binary search**, **greedy**, or **hashing**

03

Advantages and Challenges

Advantages & Challenges of Two Pointer Technique

Advantages :-

- ✔ Optimized performance – Reduces unnecessary iterations.
- ✔ Works well with sorting – Often combined with sorting for efficiency.
- ✔ Easy to implement – Simple logic with significant performance gain.
- ✔ Used in multiple problem types – Searching, merging, subarrays, and more.

Challenges :-

- ⚠ Handling duplicates while processing pairs or subarrays.
- ⚠ Deciding which pointer to move based on problem conditions.
- ⚠ Ensuring edge case handling (e.g., empty lists, negative numbers).

Real-World Applications

- Data Processing – Finding patterns in large datasets.
- Memory Management – Managing allocation and deallocation.
- Network Routing – Optimization of shortest paths in graphs.
- Game Development – Collision detection and movement mechanics.

04

Activity

Activity

- 3 Sum
- Trapping Rain Water
- Longest Palindromic Substring
- Remove Duplicates from sorted Array
- Container with most water