PROBLEM SOLVING TRICKS - ARRAYS

1-Dimensional Arrays Approaches

1. Two-Pointer Technique

Use Case: When you need to process pairs, remove duplicates, or find subarrays.

Approach: Use two pointers (start & end, or slow & fast) to traverse the array efficiently.

Examples:

1) **Reverse an array:** Swap elements from start and end pointers.

2) **Search an element:**

- Use Linear Search for unsorted arrays (simpler and equally efficient).
- Use Two-Pointer (Binary Search) only if the array is sorted (O(log n) time).

3) **Remove duplicates:** Use a slow pointer to track unique elements.

4) **Two-sum problem:** Sort the array and use two pointers to find pairs.

2. Sliding Window Technique

The Sliding Window is an algorithmic technique used to efficiently solve problems involving arrays, strings, or sequences by maintaining a "window" of elements that satisfies certain conditions. It reduces time complexity from $O(n^2)$ to $O(n)$ for many problems.

Use Case: For problems involving subarrays with a fixed size or a condition (e.g., maximum sum, smallest subarray).

Approach: Maintain a window (subarray) and adjust its size dynamically.

Examples:

1) Maximum sum of k consecutive elements.

2) Longest subarray with sum ≤ K.

3. Binary Search on Arrays

Use Case: When the array is sorted (or can be sorted) and you need O(log n) search.

Approach: Divide the search space in half repeatedly.

Examples:

1) Find an element in a sorted array.

2) Find the first/last occurrence of an element.

4. Cyclic Rotations & Reversals

Use Case: Rotating an array or reversing parts of it.

Approach: Use reversal tricks to rotate in O(1) space.

Example: Rotate array right by k steps

5. Frequency Counting (Hashing)

Use Case: When you need to count occurrences or find duplicates.

Approach: Use a hash map (or an array if elements are bounded) to store frequencies.

Examples:

1) Find duplicates in an array.

2) First non-repeating element.

6. Swap & Reorder Techniques

Use Case: When you need to modify the array in-place (e.g., move zeros to end, segregate even-odd numbers).

Approach: Use a pointer to track the correct position and swap elements.

Examples:

1) Move all zeros to the end.

2) Segregate even and odd numbers.

7. Prefix Sum (Cumulative Sum)

Use Case: When you need to compute range sums or differences efficiently.

Approach: Precompute cumulative sums to answer range queries in O(1) time.

Examples:

1) Find equilibrium index (where left sum = right sum).

2) Subarray sum equals K.

8. Kadane's Algorithm (Maximum Subarray Sum)

Use Case: Finding the maximum sum of a contiguous subarray.

Approach: Keep track of the current sum and reset it if it becomes negative.

1. Finding Min & Max in One Pass

```
int min = arr[0], max = arr[0];
for(int i = 1; i < n; i++) {
    if(arr[i] < min) min = arr[i];
    if(arr[i] > max) max = arr[i];
}
printf("Min: %d, Max: %d", min, max);
```

2. Moving Zeros to End (In-Place)

```
int nonZero = 0;
for(int i = 0; i < n; i++) {
    if(arr[i] != 0) {
        arr[nonZero++] = arr[i];
    }
}
while(nonZero < n) arr[nonZero++] = 0;
```

3. Swapping Without Temp Variable

```
arr[i] = arr[i] + arr[j];
arr[j] = arr[i] - arr[j];
arr[i] = arr[i] - arr[j];
```

4. Reverse an Array In-Place

```
int i = 0, j = n - 1;
while(i < j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    i++; j--;
}
```

5. Check for Duplicate Elements

```
for(int i = 0; i < n; i++)
    for(int j = i + 1; j < n; j++)
        if(arr[i] == arr[j])
            printf("Duplicate: %d\n", arr[i]);
```

For better performance, use hash or sort + adjacent check.

6. Move Zeros to End

```
int index = 0;
for (int i = 0; i < n; i++)
    if (arr[i] != 0) arr[index++] = arr[i];
while (index < n)
    arr[index++] = 0;
```

7. Frequency Count of Elements

```
for (int i = 0; i < n; i++) {
    int count = 1;
    if (arr[i] == -1) continue;
    for (int j = i + 1; j < n; j++) {
        if (arr[i] == arr[j]) {
            count++;
            arr[j] = -1; // mark as visited
        }
    }
    printf("%d appears %d times\n", arr[i], count);
}
```

TOPIC WISE QUESTIONS

TOPIC	S.No	Problem Statement	LeetCode	Geeksfor Geeks
1. Two-Pointer Technique	1	Two Sum (Sorted)	Link	Link
	2	Remove Duplicates	Link	Link
	3	Container With Most Water	Link	Link
	4	Move Zeroes	Link	Link
	5	Valid Palindrome	Link	Link
	6	Merge Sorted Arrays	Link	Link
	7	3Sum	Link	Link
	8	Trapping Rain Water	Link	Link
	9	3Sum Closest	Link	Link

	10	Sort Colors	Link	Link
	11	Minimum Window Substring	Link	Link
	12	Count Triplets (Sum < K)	-	Link
	13	Equal Sum Partition	-	Link
2. Sliding Window Technique	1	Max Sum Subarray (Size K)	Link	Link
	2	Smallest Subarray (Sum ≥ Target)	Link	Link
	3	Longest Substring (Unique Chars)	Link	Link
	4	Fruit Into Baskets	Link	Link
	5	Permutation in String	Link	Link
	6	Longest Subarray (Sum = K)	Link	Link
	7	Max Element in K-Sized Window	Link	Link
	8	Longest Substring (≤ K Unique Chars)	Link	Link
	9	Subarrays with Product < K	Link	Link
	10	Max Consecutive Ones III	Link	Link
3. Binary Search on Arrays	1	Binary Search	Link	Link
	2	Search in Rotated Sorted Array	Link	Link
	3	First/Last Position of Element	Link	Link
	4	Find Peak Element	Link	Link

	5	Square Root of X	Link	Link
	6	Find K Closest Elements	Link	Link
	7	Split Array Largest Sum	Link	Link
4. Cyclic Rotations & Reversals	1	Reverse Array	-	Link
	2	Find Minimum in Rotated Sorted Array	Link	Link
	3	Reverse Words in String	Link	Link
	4	Circular Array Loop	Link	Link
	5	Rotate Array (Reversal Method)	Link	Link
	6	Check Rotation of Arrays	-	Link
5. Frequency Counting (Hashing)	1	Two Sum (Unsorted)	Link	-
	2	Majority Element	Link	-
	3	All Duplicates	Link	-
	4	First Missing Positive	Link	-
	5	Anagram Grouping	Link	-
	6	Count Frequency	-	Link
	7	Non-repeating Element	Link	-
	8	Top K Frequent Elements	Link	-
	9	Longest Harmonious Subsequence	Link	-
	10	Contiguous Array	Link	-

	11	Subarray Sums Divisible by K	Link	-
	12	Majority Element (> n/2)	Link	-
	13	Freq > n/3	Link	-
	14	Group Frequency	-	Link
6. Swap & Reorder Techniques	1	Dutch National Flag	Link	-
	2	Move Negative Numbers to Left	-	Link
	3	Segregate Even and Odd	-	Link
	4	Cyclic Sort	-	Link
	5	Minimum Swaps to Sort Array	-	Link
	6	Move Zeros	Link	-
	7	Rearrange Array (Alternate +ve & -ve)	-	Link
	8	Kth Largest in Array	Link	-
	9	Reorganize String	Link	-
	10	Max Even Sum	-	Link
	11	Parity Sort	Link	-
	12	Min Swap (Elements < K Together)	-	Link
	13	Wave Form	-	Link
7. Prefix Sum (Cumulative Sum)	1	Range Sum Query	Link	-
	2	Equilibrium Index	-	Link

	3	Subarray Sum Equals K	Link	-
	4	Maximum Subarray (Kadane's)	Link	-
	5	Product Except Self	Link	-
	6	Subarray Sum = 0	-	Link
	7	Subarray Count (Given Sum)	-	Link
	8	Random Pick with Weight	Link	-
	9	Subarray Sum Equals K (Negatives)	Link	-
	10	Maximum Size Subarray (Sum = K)	-	Link
	11	Count Number of Nice Subarrays	Link	-
	12	Subarray Division (Divisible by K)	Link	-
	13	Subarray 1s & 0s (Equal Count)	Link	-
	14	Max Sum (Even Length Subarray)	-	Link
	15	Sum = Target (Count Subarrays)	Link	-
8. In-Place Problems	1	Remove Element	Link	-
	2	Remove Duplicates (Sorted)	Link	-
	3	Merge Sorted Arrays	Link	-
	4	Replace with Next Greatest	-	Link
	5	Missing Positive	Link	-

	6	Rotate Array	Link	-
	7	Reverse an Array	-	Link
	8	Set Matrix Zeroes	Link	-
	9	Wiggle Sort	Link	-
	10	Equal Sum (3-Partition)	Link	-
	11	Sort Colors	Link	-
	12	Next Greater (Replace Elements)	-	Link
9. Palindrome-Related Problems	1	Palindrome String	Link	-
	2	Palindrome Array	-	Link
	3	Palindrome Number	Link	-
	4	Palindrome Subarray	Link	-
	5	Minimum String Palindrome	Link	-
	6	Longest Palindromic Subsequence	Link	-
	7	Palindrome Partitioning	Link	-
	8	Shortest Palindrome	Link	-
	9	Palindrome Pairs	Link	-
	10	Palindrome Digit	-	Link
	11	Palindromic Form (Min Changes)	-	Link

2D Arrays

2D Array Problem-Solving Tricks:

#	Trick	Description	Use Case
1	**Row-wise & Column-wise Traversal**	Use nested loops: for(i) outer for rows, for(j) inner for columns	Printing, summing, input
2	**Diagonal Traversal**	For main diagonal: i == j, for secondary: i + j == n – 1	Pattern, symmetry
3	**Transpose Matrix**	Swap elements: mat[i][j] = mat[j][i]	Rotation, matrix algebra
4	**Matrix Rotation**	Rotate 90° clockwise: Transpose + reverse each row	Games, image rotation
5	**Search in Sorted 2D Matrix**	Start from top-right, go left/down accordingly	Matrix binary search
6	**Spiral Traversal**	Use 4 bounds: top, bottom, left, right and loop inward	Spiral print
7	**Boundary Traversal**	Print only outermost elements	Shell printing
8	**Count Zeros or Ones Efficiently**	If sorted, use binary search per row	Optimized counting
9	**Sum of Each Row/Col**	Use accumulator per loop	Reports, histogram
10	**Prefix Sum Matrix (2D)**	prefix[i][j] = above + left - diagonal + current	Submatrix sum queries
11	**In-place Element Swap**	Use a temp variable or arithmetic swaps	Memory-efficient update
12	**Max Submatrix Sum (Kadane 2D)**	Apply Kadane's algorithm on columns over rows	Advanced DP
13	**Flip Image (Horizontal/Vertical)**	Reverse rows or columns	Pattern problems
14	**Flood Fill / DFS on Grid**	Use recursion on grid[i][j] and valid neighbors	Maze, islands
15	**Pattern Matching in Matrix**	Search a small matrix pattern in large one	2D pattern search

2D Array Problem Solving Questions

Category	S. No	Problem Statement	LeetCode	Geeksfor Geeks
1. Row-wise & Column-wise Traversal	1	Print matrix row-wise and column-wise	-	Link
	2	Find row with maximum sum	-	Link
	3	Print elements greater than X	-	Link
	4	Find row with smallest sum	-	Link
	5	Print alternate rows/columns	-	Link
	6	Check if any column is entirely zero	-	Link
2. Diagonal Traversal	1	Print primary diagonal	-	Link
	2	Print secondary diagonal	-	Link
	3	Sum of both diagonals	Link	Link
	4	Replace diagonal elements with squares	-	Link
	5	Check identical diagonals	-	Link
3. Transpose Matrix	1	Transpose square matrix	Link	Link
	2	Transpose non-square matrix	-	Link
	3	Check symmetric matrix	-	Link
	4	Check skew-symmetric matrix	-	Link
	5	Transpose sparse matrix	-	Link
	6	Verify orthogonal matrix	-	Link

Matrix Rotation	1	Rotate 90° clockwise	Link	Link
	2	Rotate 180°	-	Link
	3	Rotate anti-clockwise	-	Link
	4	Rotate submatrix	-	Link
	5	Rotate matrix layers	-	Link
	6	Rotate a ring	-	Link
4. Search in Sorted 2D Matrix	1	Search in sorted matrix	Link	Link
	2	Count occurrences of K	-	Link
	3	Find position of K	Link	Link
	4	Find median	Link	Link
	5	Find smallest > K	-	Link
	6	Search with duplicates	-	Link
5. Spiral Traversal	1	Print spiral order	Link	Link
	2	Find Kth in spiral	-	Link
	3	Spiral reverse print	-	Link
	4	Store in 1D array	-	Link
	5	Replace with primes	-	-
	6	Jagged matrix spiral	-	Link
6. Boundary Traversal	1	Print boundary	-	Link
	2	Sum boundary	-	Link

	3	Count corners	-	-
	4	Product boundary	-	-
	5	ASCII boundary	-	-
	6	Palindrome boundary	-	-
7. Count Zeros/Ones	1	Count 0s	-	Link
	2	Count 1s per row	-	Link
	3	Row with max 1s	Link	Link
	4	Count 0s islands	-	Link
	5	Distance from 0	Link	Link
8. Sum of Rows/Columns	1	Print row/column sums	-	Link
	2	Row with highest sum	-	Link
	3	Check equal row sums	-	Link
	4	Normalize by row sum	-	Link
	5	Longest even sequence column	-	Link
	6	Check magic square	-	Link
9. Prefix Sum Matrix	1	Compute prefix sum	-	Link
	2	Submatrix sum query	Link	Link
	3	Submatrix average	-	Link
	4	Count submatrices sum K	Link	Link
	5	Max sum rhombus	-	Link

10. In-place Swaps	1	Diagonal swap	-	Link
	2	Corner swap	-	Link
	3	Transpose in-place	Link	Link
	4	Reverse rows/columns	-	Link
	5	Swap alternate rows	-	Link
	6	Random shuffle	-	Link
11. Max Submatrix Sum	1	Max sum submatrix	-	Link
	2	Max sum rectangle	Link	Link
	3	Largest all-1s area	Link	Link
	4	K×K max sum submatrix	-	Link
	5	Uniform max sum rectangle	-	Link
12. Flip Operations	1	Horizontal flip	Link	Link
	2	Vertical flip	-	Link
	3	Mirror image	-	Link
	4	Quadrant flip	-	-
	5	Conditional flip	-	-
	6	Diagonal flip	-	Link
13. Flood Fill/DFS	1	Number of islands	Link	Link
	2	Connected components	-	Link
	3	Color fill	Link	Link

	4	Enclosed regions	Link	Link
	5	Shortest path in maze	Link	Link
	6	Increasing path	Link	Link
14. Pattern Matching	1	Find submatrix pattern	-	Link
	2	Square pattern	-	Link
	3	L/T/+ patterns	-	Link
	4	Word search	Link	Link
	5	Plus sign pattern	Link	Link
	6	Checkerboard pattern	-	Link

Note:

1. **Pattern Matching**:
   - Use nested loops for small patterns (e.g., 2x2).
   - For larger patterns, DFS/BFS or Rabin-Karp hashing (for character grids).
2. **2D Kadane's Algorithm**:
   - Convert to 1D by fixing rows and applying Kadane's vertically.
3. **In-Place Tricks**:
   - Use bit manipulation or matrix cells themselves to store temporary states (e.g., matrix[i][j] |= (new_val << 16)).
4. **Flood Fill Optimizations**:
   - For large matrices, use BFS with queue instead of recursive DFS to avoid stack overflow.

STRINGS Problem-Solving Tricks:

#	Trick	Description
1	Two-pointer on strings	Use two indices (start and end) to compare or manipulate string efficiently
2	Character frequency counting	Use an int freq[256] = {0} to count each character using ASCII values
3	In-place string reversal	Swap characters using two pointers, no extra string needed
4	Removing duplicates	Mark visited chars with a boolean array or use slow-fast pointers
5	String tokenization	Use strtok() to split string based on delimiters like spaces, commas
6	String rotation check	Concatenate string with itself and check if other string is a substring
7	Substring search (Brute/Optimized)	Use nested loops for brute, or implement KMP/Rabin-Karp
8	String to number / number to string	Use atoi(), itoa() or manual conversion via loop
9	Prefix/Suffix matching	Compare beginning or end characters using strncmp() or manual loop
10	Reversing words in string	First reverse entire string, then reverse each word
11	Palindrome subsequence / substring	Check longest palindromic subsequence or substring
12	Sliding window on strings	Maintain frequency window to find smallest/largest substrings
13	Lexicographic operations	Use strcmp(), sorting, etc., to compare alphabetically

14	Edit distance / DP on strings	Use 2D DP table to compute min operations to convert strings
15	Bit masking for lowercase letters	Use 26-bit integer to track lowercase letters efficiently

String Problem-Solving Questions by Technique

Category	S. No	Problem Statement	Leet Code	Geeksfor Geeks
1. Two-Pointer on Strings	1	Palindrome Check (case-insensitive, ignore non-alphanumeric)	Link	-
	2	Reverse String (in-place)	Link	Link
	3	Valid Palindrome II (delete ≤1 char)	Link	-
	4	Check if two strings are mirrors	-	Link
2. Character Frequency Counting	1	Count frequency of each character	-	Link
	2	Anagram Check	Link	Link
	3	First Unique Character	Link	-
	4	Group Anagrams	Link	-
3. In-Place String Reversal	1	Reverse a String (in-place)	Link	Link
	2	Reverse Words in a Sentence	Link	Link
	3	Reverse Vowels	Link	-
	4	Reverse String (except special chars)	-	Link
4. Removing Duplicates	1	Remove Adjacent Duplicates (in-place)	Link	Link

	2	Remove All Duplicates (keep first occurrence)	-	Link
5. String Tokenization	1	Word Count (CSV tokens)	-	Link
	2	Validate IP Address	Link	Link
6. String Rotation Check	1	Rotate String (check rotation)	Link	Link
	2	Minimum Rotations to Original	-	Link
	3	Circular String Match	-	Link
7. Substring Search	1	Count Length of All Substrings	-	Link
	2	Implement strStr() (substring search)	Link	Link
	3	Repeated Substring Pattern	Link	-
	4	Rabin-Karp / KMP Pattern Matching	-	Link
8. String ↔ Number Conversion	1	String to Integer (atoi)	Link	Link
	2	Integer to Roman	Link	Link
	3	Validate Numeric String	Link	-
9. Prefix/Suffix Matching	1	Longest Common Prefix	Link	Link
	2	Suffix Check (ends with given suffix)	-	Link
	3	Auto-complete Feature	-	Link
10. Reversing Words	1	Reverse Words II (preserve whitespace)	Link	-
	2	Reverse Order of Words (not letters)	Link	-
	3	Reverse Characters Word-by-Word	-	Link

11. Palindrome Subsequence /Substring	1	Count Length of All Palindromic Subsequences	-	Link
	2	Longest Palindromic Substring	Link	Link
	3	Longest Palindromic Subsequence	Link	Link
	4	Count All Palindromic Substrings	Link	-
12. Sliding Window on Strings	1	Minimum Window Substring	Link	Link
	2	Longest Substring Without Repeating Characters	Link	Link
	3	Longest Repeating Character Replacement	Link	-
13. Lexicographic Operations	1	Lexicographical Order (sort strings)	-	Link
	2	Largest Number (arrange numbers as string)	Link	Link
14. Edit Distance (DP on Strings)	1	Edit Distance (min operations)	Link	Link
	2	One Edit Distance (insert/delete/replace)	Link	-
	3	Word Break Problem	Link	Link
15. Bitmasking for Lowercase Letters	1	Unique Characters (case-insensitive)	Link	Link
	2	Find the Difference (added char)	Link	-
	3	Find Duplicate Chars (bitmask)	-	Link
	4	Subsets of String (bitmasking)	-	Link

Pointers Problem-Solving Tricks-1 in C

#	Concept	Description	Example
1	**Pointer Basics**	A pointer holds the address of a variable. Use * to dereference and & to get address.	int x = 10;int *p = &x;printf("%d", *p); // 10
2	**Pointer to Pointer**	A pointer to another pointer; used in 2D arrays or to modify pointers.	int x = 5;int *p = &x;int **pp = &p;printf("%d", **pp); // 5
3	**Function with Pointer Args**	Pointers allow a function to modify original values (pass by address).	void swap(int *a, int *b) { int t=*a; *a=*b; *b=t;}
4	**Array and Pointer Relationship**	Arrays and pointers are tightly related. arr[i] == *(arr+i).	int arr[] = {1, 2};printf("%d", *(arr + 1)); // 2
5	**Dynamic Memory Allocation**	Allocates memory during runtime using malloc, free, etc.	int *p = malloc(5 * sizeof(int));free(p);
6	**Pointer to Array Elements**	Traverse an array using a pointer instead of indexing.	int arr[] = {10, 20};int *p = arr;printf("%d", *(p + 1)); // 20
7	**Pointer to String**	Strings are char arrays; can be traversed using a char*.	char *s = "hello";printf("%c", *(s + 1)); // e
8	**Pointer with Structs**	Use -> to access struct members via pointers.	struct S { int x; };struct S s = {5};struct S *p = &s;printf("%d", p->x); // 5

9	**Array of Pointers**	Useful to store strings or multiple arrays.	char *names[] = {"A", "B"};printf("%s", names[1]); // B
10	**Function Pointer**	Stores address of a function to call dynamically.	int add(int a, int b) { return a + b; }int (*fptr)(int, int) = add;printf("%d", fptr(2, 3)); // 5
11	**Void Pointer**	A generic pointer that can point to any data type.	int x = 10;void *vp = &x;printf("%d", *(int*)vp); // 10
12	**Const with Pointers**	Controls mutability of value or pointer: const int *p, int *const p	const int *p; // can't change *pint *const p; // can't change p
13	**Dangling/Wild Pointer**	Pointer that refers to freed or uninitialized memory.	int *p;// no initialization = wildfree(p); // now dangling
14	**Pointer in Recursion**	Pass pointers in recursion to keep track of results.	void rec(int *sum, int n) { if(n) { *sum += n; rec(sum, n-1); }}
15	**Pointer-based Swapping**	Swap values without return using pointers.	void swap(int *a, int *b) { int t=*a; *a=*b; *b=t;}

Pointer Problem-Solving Tricks-2 in C

#	Trick Category	Description	Code Example	Use Case
1	Pointer Arithmetic	Move pointers to traverse memory	int *ptr = arr; printf("%d", *(ptr + 2));	Array traversal, offset calculations
2	Double Pointers	Pointers pointing to other pointers	int **pp = &p; printf("%d", **pp);	Dynamic 2D arrays, modifying pointers in functions
3	Dynamic Allocation	Allocate/free memory at runtime	int *arr = malloc(5 * sizeof(int)); free(arr);	Resizable arrays, strings
4	Pointer Swapping	Swap values using pointers (no temp variable needed)	void swap(int *a, int *b) { int t = *a; *a = *b; *b = t; }	In-place algorithms (e.g., sorting)
5	Function Pointers	Store functions in variables or pass as arguments	int (*func)(int, int) = add; func(3, 5);	Callbacks, event-driven programming
6	String Manipulation	Use pointers for efficient string operations	while (*str != '\0') { str++; }	Custom strlen, strcpy implementations
7	Struct Pointers	Access struct members using -> operator	Point *p = malloc(sizeof(Point)); p->x = 10;	Linked lists, trees, dynamic data structures
8	Void Pointers	Type-agnostic pointers for generic functions	void print(void *data, char type) { ... }	Generic data handling (e.g., qsort)

9	Pointer to Array	Treat arrays as pointers for traversal	int *ptr = arr; for (int i = 0; i < 5; i++) printf("%d", ptr[i]);	Efficient array processing
10	Debugging Pitfalls	Avoid dangling pointers and out-of-bounds access	int *p = NULL; if (p) printf("%d", *p);	Preventing crashes/undefined behavior
11	Const Pointers	Restrict pointer modifications for safety	const int *p = &x; // Can't modify *p	Read-only data protection
12	Pointer Subtraction	Calculate distances between pointers (e.g., string length)	int len = end_ptr - start_ptr;	Custom strlen, subarray calculations
13	Array of Pointers	Store multiple pointers in an array	char *words[] = {"hello", "world"};	Command-line arguments (argv)
14	Pointer Casting	Convert between pointer types (e.g., int* to char*)	int x = 65; char *c = (char*)&x; // 'A' (ASCII 65)	Binary data manipulation
15	Null Pointer Checks	Validate pointers before dereferencing	if (ptr != NULL) { *ptr = 10; }	Robust error handling

Structures - Problem Solving Tricks:

No.	Trick	Description	Example	Use Case
1	**Basic struct declaration & access**	Use dot to access members of a structure.	struct Student { int id; float marks; }; struct Student s = {1, 89.5}; printf("%d %.1f", s.id, s.marks);	Represent a single entity like a student or product.
2	**Struct with arrays**	Include arrays inside a struct to store related data.	struct Student { char name[30]; int marks[5]; };	Store student name and marks for 5 subjects.
3	**Array of structs**	Create multiple records with the same structure.	struct Emp { int id; char name[20]; }; struct Emp e[2] = {{1,"A"}, {2,"B"}};	Manage employee records, product list, etc.
4	**Pass struct to function**	Pass entire struct by value or address.	void print(struct Emp e) { printf("%s", e.name); }	Separate logic into functions like display data.
5	**Return struct from function**	Return a complete struct from a function.	struct Point getOrigin() { struct Point p = {0, 0}; return p; }	Return best match, min/max record, etc.
6	**Pointer to struct**	Access struct members via pointer using arrow operator.	struct Book {char title[20]; int price;}; struct Book *b; b = malloc(sizeof(*b)); strcpy(b->title, "C");	Dynamic structures like linked lists, trees.

7	**Dynamic memory for struct**	Allocate struct objects during runtime.	struct Node *n = malloc(sizeof(struct Node));	Create flexible data structures at runtime.
8	**Nested structures**	One struct inside another.	struct Address { char city[20]; }; struct Person { char name[20]; struct Address addr; };	Real-world data modeling like user profiles.
9	**Self-referential structure**	Struct contains pointer to itself.	struct Node { int data; struct Node *next; };	Essential for Linked Lists, Trees, Stacks.
10	**Function pointer inside struct**	Simulate behavior via function pointers.	struct Operation { int (*add)(int, int); };	Callback or menu-driven programs.
11	**Sorting array of structs**	Sort structs based on field values.	qsort(emp, n, sizeof(emp[0]), cmp);	Sort students by marks, products by price.
12	**Compare two structs**	Compare all fields manually.	if (s1.id == s2.id && strcmp(s1.name, s2.name)==0)	Check for record equality.
13	**Structure padding**	Understand memory layout with gaps.	struct A { char c; int i; }; size = 8	Optimize memory layout in embedded apps.

14	**typedef with structs**	Create a shortcut name using typedef.	typedef struct Book Book; Book b;	Cleaner code, especially in large projects.
15	**File I/O using struct**	Read or write struct directly to files.	fwrite(&s, sizeof(s), 1, f); fread(&s, sizeof(s), 1, f);	Store and retrieve structured records from disk.

Pointer Mistakes:

Pitfall	Fix
Dangling pointers	Always free memory or set to NULL after free(ptr)
Memory leaks	Pair every malloc with free
Buffer overflows	Check bounds: if (index < size) arr[index] = val;
Type mismatches	Use explicit casting: int *p = (int*)malloc(n * sizeof(int));

FUNCTIONS & RECURSION

Aspect	Iterative Approach	Recursive Approach
Time Complexity	Usually same as recursive	Usually same as iterative
Space Complexity	O(1) (no call stack)	O(n) (call stack overhead)
Readability	Less intuitive sometimes	More elegant for some problems
Performance	Faster (no function calls)	Slower (call stack ops)