**Srikar Reddy Kamatham - U29573910**

**Assignment 2 - Computational problem-solving prompts Doc**

1. **Find Missing Numbers in Array**

Initial Idea: Find all numbers from 1 to n that are missing from the input array. Use input array as a hash by marking indices (value - 1) negative.

Prompt: Is there a better way to find missing numbers from an array instead of looping

Response: Using Contains() in a loop is inefficient (O(n²)). Instead, convert the array to a HashSet<int> for constant-time lookups. A more space-optimized solution uses in-place negation marking to track which numbers are present without additional space.

Implementation: Used the Math.Abs(nums[i]) - 1 strategy to mark visited indices as negative. Any index left positive corresponds to a missing number. This runs in O(n) time and O(1) space (excluding output).

1. **Sort Array by Parity**

Initial Idea: Should move even numbers to the front, odd to the back. Use two pointers, one from start (look for odd) and One from end (look for even)

Prompt: What’s the best way to move even numbers to the front and odd numbers to the end in an array, in-place?

Response: A two-pointer approach from start and end ensures that even numbers are placed in front and odd numbers at the back, with minimal swaps. This avoids extra space.

Implementation: Used in-place swapping using pointers i and j. If nums[i] is odd and nums[j] is even, swap them. Time complexity is O(n), space is O(1).

1. **Two Sum**

Initial Idea: To use a dictionary for storing the num -> index

Prompt: How can I reduce the time complexity of the classic Two Sum problem?

Response: Using a dictionary to store previously seen numbers allows checking for the complement (target - num) in O(1) time. This reduces time complexity from O(n²) to O(n).

Implementation: Used Dictionary<int, int> to store each number and its index while traversing. If complement exists in the map, return its index and current index.

1. **Maximum Product Of 3 Numbers Strategy**

Prompt: What is the best way to find the maximum product of three numbers in an integer array?

Response: Sort the array and compare Product of three largest numbers, Product of two smallest (negative) numbers with the largest

Implementation: Sorted array and compared nums[n-1] \* nums[n-2] \* nums[n-3] vs nums[0] \* nums[1] \* nums[n-1]. Returned the max of the two. Time complexity is O(n log n).

1. **DecimalToBinary Conversion**

Prompt: How do I convert a decimal number to binary in string format?

Response Summary:  
Use repeated division by 2 and append remainders to a string. Special case for input 0. Avoid negative inputs unless you want to return two’s complement.

Implementation Details:  
Handled 0 and negatives up front. Used a loop: binary = (n % 2) + binary and n /= 2 until n == 0.

1. **FindMinimumInRotatedSortedArray**

Prompt: How to efficiently find the minimum element in a rotated sorted array?

Response Summary: Use binary search, If mid > right, the min is in the right half. Else, the min is in the left half or at mid.

Implementation: Binary search loop with conditionals comparing nums[mid] and nums[right]. Runs in O(log n) time, space O(1).

1. **Is Palindrome or not a Palindrome**

Prompt: How to check if a number is a palindrome without converting it to a string?

Response Summary: Reverse half the number and compare it with the other half. Don’t allow negative numbers. Special care for numbers ending with 0 but not 0 itself (like 10).

Implementation: Used mathematical reverse technique and then compared either full or half depending on length.

1. **Fibonacci Number**

Prompt: What’s the most efficient way to calculate the nth Fibonacci number?

Response Summary: Use bottom-up iteration with two variables to store previous two values. Avoid recursion to reduce call stack overhead.

Implementation: Used variables and looped until n. Time O(n), space O(1).