AI Coding Assistant

**a. Prompt:**  
For "Question 1," how would I create an if statement that adds loop values of i to a list if i is not contained in a HashSet?

**b. Response:**  
The AI provided the condition if (!numSet.Contains(i)). Here’s the context:

* A for loop iterates from 1 to nums.Length.
* The if statement checks if the HashSet numSet does **not** contain the current value of i.
* If i is missing from the HashSet, it is added to the missingNumbers list, which is returned after the loop.

**c. Implementation Details:**  
The AI regenerated my entire code block, but I only needed the if statement. I extracted the relevant logic (!numSet.Contains(i)) and integrated it into my existing code within the for loop.

**d. Adjustments:**

* No modifications were needed, the solution worked as expected.

**a. Prompt:**

For "Question 2," create a line of code to handle two edge cases: checking if the input array is null. And checking if the array is empty.

**b. Response:**

The AI provided:

if (nums == null || nums.Length == 0)

return nums ?? Array.Empty<int>();

* The if statement checks if nums is **null** **or** if its length is **0**.
* If either condition is true, it returns Array.Empty<int>() (a cached empty array instance).
* The nums ?? is a null-coalescing operator, but it’s redundant here since we already confirmed nums is either null or empty.

**c. Implementation Details:**

* The AI regenerated my entire code block, but I only needed the edge-case check.
* I placed this snippet **at the top** of my method, **before** initializing lists or starting loops, to exit early if the input is invalid.

**d. Adjustments:**

* **Removed nums ??** because:
  + If nums is null, the condition is already true, and Array.Empty<int>() is the correct return.
  + If nums is empty (but not null), nums.Length == 0 is true, and nums itself could technically be returned, but returning Array.Empty<int>() is cleaner and more consistent.

**a. Prompt:**

For "Question 4," create a line of code to handle three edge cases: check if the input array is null, has fewer than 3 elements, or if the array has exactly 3 elements.

**b. Response:**

The AI provided:

if (nums == null || nums.Length < 3) return 0;

if (nums.Length == 3) return nums[0] \* nums[1] \* nums[2];

* **First check**: Exits early if nums is null or has **<3 elements**, returning 0 (or another default; clarify requirements).
* **Second check**: Optimizes for the case where nums.Length == 3 by directly returning the product of all elements.

**c. Implementation Details:**

* The AI regenerated the full code block, but I extracted **only the edge-case checks**.
* Placed these checks **at the top** of the method (before sorting or further logic) for efficiency.

**d. Adjustments:**

* **None needed**. The logic aligns with the requirements:
  + Early exit for invalid/undersized inputs.
  + Short-circuit for the trivial case (exactly 3 elements).

**a. Prompt:**

For "Question 5," provide the logic to convert a decimal (integer) into its binary representation.

**b. Response:**

The AI outlined the following steps:

1. **Initialize an empty string** (e.g., binaryStr) to store the binary result.
2. **While loop**: Repeatedly divide the decimal number by 2 and prepend the remainder (0 or 1) to binaryStr.
3. **Return** the final binaryStr after the loop completes.

**c. Implementation Details:**

* **Used the AI’s logic as a blueprint** but implemented it independently.
* **Added edge cases**:
  + Handled decimalNum == 0 (returns "0").
  + Added input validation (e.g., negative numbers).
* **Optimization**: Used StringBuilder instead of string concatenation for efficiency (optional).

**d. Adjustments:**

* **Extended the logic**: The AI’s steps were foundational but lacked edge-case handling.
  + Added checks for 0 and negative inputs.
  + Ensured correct output format (e.g., leading zeros if needed).

**a. Prompt:**

For "Question 5," create code to handle edge cases where: the decimal number is zero and the decimal number is negative

**b. Response:**

The AI provided:

if (decimalNumber == 0)

return "0";

if (decimalNumber < 0)

return string.Empty;

* **First check**: Returns "0" immediately if input is zero (base case)
* **Second check**: Returns empty string for negative numbers (may vary by requirements)

**c. Implementation Details:**

* Extracted just the edge-case handling from AI's full solution
* Placed this logic at the **start** of the method:
  + Before initializing the binary string
  + Before the conversion while-loop
* Maintained clean separation between validation and core logic

**d. Adjustments:**

* **None required**: The solution:
  + Matched the problem requirements
  + Handled cases explicitly
  + Used appropriate return values

**a. Prompt:**

For "Question 6," write code to handle null/empty array input by returning 0.

**b. Response:**

The AI provided:

if (nums == null || nums.Length == 0)

return 0;

* Checks if nums is null OR empty (Length == 0)
* Returns 0 immediately if either condition is true (edge case handling)

**c. Implementation Details:**

* Extracted only this guard clause from AI's full solution
* Placed it **before any processing logic**:
  1. At method start (before variable declarations)
  2. Preceding the main for loop
* Ensures invalid inputs exit early without unnecessary computations

**d. Adjustments:**

* **Verified correctness**: Confirmed 0 is the required default for invalid inputs
* **Maintained consistency**:
  + Kept original return type (int)
  + Preserved all functional requirements
* **No changes needed**: The snippet worked as-is for all test cases

**a. Prompt:**

For "Question 7," provide the definition of a palindrome number.

**b. Response:**

The AI defined a palindrome number as:  
*A number that reads the same backward as forward (e.g., 121 or 1331).*

**Key Characteristics:**

* Symmetrical digit sequence
* Identical when reversed
* Single-digit numbers (0-9) are trivially palindromic

**c. Implementation Details:**

While no code was provided, this definition directly informed:

1. **Algorithm Approach**:
   * Convert number to string → Compare with its reverse  
     OR
   * Mathematically reconstruct the reversed number
2. **Edge Case Handling**:
   * Negative numbers (automatically non-palindromic due to '-' sign)
   * Numbers ending with 0 (cannot be palindromic unless 0 itself)

**d. Adjustments:**

* No modifications were needed, this was a theoretical question.

**a. Prompt:**

For "Question 7," explain the logic needed to determine if a number is a palindrome.

**b. Response:**

The AI outlined this step-by-step approach:

1. **Negative Check**: Immediately return false for negative numbers (can't be palindromes)
2. **Variable Setup**:
   * Store original number (original) for later comparison
   * Initialize reversed (0) to build the reversed number
3. **Digit Reversal**:
   * Use a while loop to process each digit
   * Extract last digit (x % 10)
   * Append digit to reversed (reversed \* 10 + digit)
   * Remove processed digit (x /= 10)
4. **Comparison**: Check if original == reversed

**c. Implementation Details:**

* **Used the AI’s logic as a blueprint** but implemented it independently.

**Key Implementation Decisions:**

* Optimized by returning early for single-digit numbers
* Used integer operations instead of string conversion for efficiency

**d. Adjustments:**

* **Logic → Code Translation**:
  + Implemented the AI's described logic verbatim
  + Verified edge case handling (negatives, single-digit, zero)
* **Structural Optimizations**:
  + Eliminated unnecessary exception handling
  + Streamlined variable usage
* **Validation**: Confirmed correct behavior for:
  + Palindromes (121, 1331)
  + Non-palindromes (123, -454)
  + Edge cases (0, 1-9)

**a. Prompt:**

For "Question 8," provide the mathematical definition of a Fibonacci number.

**b. Response:**

The AI described the Fibonacci sequence as:

* A series where each number (after the first two) is the sum of the two preceding numbers
* Standard initialization: F₀ = 0, F₁ = 1
* Sequence progression: 0, 1, 1, 2, 3, 5, 8, 13, 21,..

**Mathematical Definition:**

Copy

Fₙ = Fₙ₋₁ + Fₙ₋₂

with base cases:

F₀ = 0,

F₁ = 1

**c. Implementation Considerations:**

While no code was requested, this definition informs potential implementations:

1. **Recursive Approach**
2. **Iterative Approach** (Preferred)
3. **Memorization** (Optimized recursion)

**d. Adjustments:**

No adjustments were needed as this was purely a theoretical question. The AI's definition:

* Matches standard mathematical conventions
* Provides sufficient foundation for implementation
* Correctly identifies the sequence initialization

**a. Prompt:**

For "Question 8," describe the logic for an iterative Fibonacci sequence solution.

**b. Response:**

The AI outlined the following approach:

1. **Handle Edge Cases**:
   * If **n** is negative, return -1.
   * If **n** is 0, return 0.
   * If **n** is 1, return 1.
   * If **n** is greater than 30, return -1 (to avoid excessive computation).
2. **Initialize Variables**:
   * Initialize two variables **a** and **b** to 0 and 1, respectively. These represent the first two Fibonacci numbers.
3. **Iterate to Calculate Fibonacci**:
   * Use a loop to iterate from 2 to **n**.
   * In each iteration, calculate the next Fibonacci number by summing **a** and **b**.
   * Update **a** to the value of **b**, and **b** to the newly calculated Fibonacci number.
4. **Return the Result**:
   * After the loop, **b** will contain the nth Fibonacci number

**c. Implementation:**

**Implementation Details**

**1. Edge Case Handling**

**if (n < 0) return -1; // Reject negative inputs**

**if (n == 0) return 0; // Base case: F(0) = 0**

**if (n == 1) return 1; // Base case: F(1) = 1**

**if (n > 30) return -1; // Prevent potential overflow**

**2. Core Algorithm**

**int a = 0; // Tracks F(n-2)**

**int b = 1; // Tracks F(n-1)**

**int result = 0;**

**for (int i = 2; i <= n; i++)**

**{**

**result = a + b; // Compute current term**

**a = b; // Shift F(n-2) forward**

**b = result; // Shift F(n-1) forward**

**}**

**return result;**

**d. Adjustments:**

1. **Overflow Prevention Guard**
   * **Original Code: No explicit overflow protection**
   * **Adjusted Code:**

**if (n > 30) return -1; // Strict limit per assignment requirements**

* + **Reason:**
    - **Per assignment specifications, input n must not exceed 30**
    - **Prevents integer overflow while meeting project constraints**

1. **Edge Case Consolidation**

* **Simplified base cases:**

**if (n <= 1) return n;**