Assignment 2 - Computational problem solving

Author: Raghu Ram Komara

Step 1: Clone the Starter Repository

Action Taken: I used the GitHub repository link provided for the assignment and created a fork of that repository under my own GitHub account. This allowed me to retain the original template and commit my changes independently.

Repository Setup Process:

1. Navigated to the provided GitHub repository link.
2. Clicked on the "Fork" button to create a copy of the repository under my GitHub account.
3. Opened the solution in Visual Studio for development.

Step 2: Implement Algorithms

I completed the method implementations as defined in the Program.cs file. For each function, I ensured the logic adheres to time and space complexity expectations while addressing all possible edge cases.

Step 3: Use AI for Coding Assistance

Throughout this assignment, I utilized GitHub Copilot within Visual Studio for coding suggestions, along with external resources such as YouTube tutorials and GeeksforGeeks articles to strengthen my conceptual understanding.

GitHub Copilot Assistance

For each question, I prompted Copilot with direct C#-related queries, and Copilot suggested base logic for solving the problem. I reviewed these suggestions, applied the ones that matched the problem constraints, and adjusted where needed for edge cases and optimal performance.

Prompts Used:

Examples of exact Copilot queries include:

- "find missing numbers in array c#"
- "two sum c# brute force"
- "sort array by parity using two pointers"
- "fibonacci number iterative c#"
- "check if number is a palindrome without converting to string"

Responses Received:

- For most queries, Copilot suggested basic control-flow or data structure approaches (e.g., loops, sorting, index comparisons).
- For logic like reversing digits, binary conversion, or checking for gaps in sorted arrays, the output was syntactically correct but required review and modifications to handle edge cases.

Implementation Details:

- I selectively used Copilot's suggestions as a starting point, then added custom validations (like array length checks, duplicate handling, or overflow protection).
- For example, in the Sort Array by Parity problem, Copilot suggested a basic two-pointer implementation which I refined to avoid unnecessary swaps.

Adjustments Made:

- Handled edge cases Copilot did not suggest (e.g., empty arrays, single-element arrays, out-of-range indices).
- Enhanced mathematical logic using runtime exception handling (like detecting overflow using checked blocks).
- Added explanatory comments to clarify logic.

External Resources Used: -

Not just the copilot cleared the concepts in addition to Copilot, I referred to the following to strengthen core algorithmic concepts:

- YouTube Tutorials:
    - *"Two Pointer Technique Explained"* – helped clarify in-place swapping logic.
    - *"Binary Search on Rotated Sorted Array"* – aided in implementing FindMin() logic efficiently.
- GeeksforGeeks Articles:
    - Two Pointer Technique
    - Sliding Window Technique

These resources provided theoretical context to Copilot's practical suggestions, enabling me to build well-structured, edge-case-safe solutions.

Question 1: Find Missing Numbers in Array

Problem Description:
Given an unsorted integer array of size n with elements from 1 to n, return all missing numbers.

Approach:

- Sort the array.
- Loop through and identify gaps between consecutive elements.
- Skip duplicates and collect missing values in a list.

Copilot Prompt:
find missing numbers in array c#

Response Received:
Suggested sorting the array and checking differences between adjacent elements to detect gaps.

Adjustments Made:

- Skipped repeated values using current != next.
- Checked edge case where the largest number is less than n.
- Checked for empty input.

Time Complexity: O(n log n)
Space Complexity: O(m), where m is the number of missing values

Concepts Covered: Sorting, Arrays, Gap Detection

Edge Cases Handled:

- Empty array
- Repeated numbers (e.g., multiple 2s)
- Missing numbers at the end
- Out-of-order input

Question 2: Sort Array by Parity

Problem Description:
Move all even numbers to the front of the array, followed by odd numbers.

Approach:

- Use a two-pointer approach.
- Swap odd/even values in-place.

Copilot Prompt:
sort array by parity using two pointers in c#

Response Received:
Suggested maintaining two pointers and swapping elements based on parity.

Adjustments Made:

- Prevented unnecessary swaps.
- Preserved array size and indices.

Time Complexity: O(n)
Space Complexity: O(1)

Concepts Covered: Two-Pointer Technique, In-Place Array Manipulation

Edge Cases Handled:

- All even or all odd elements
- Single element array
- Array already sorted by parity
- Mixed parity values and duplicates

Question 3: Two Sum

Problem Description:
Return indices of two numbers in an array that sum to a given target.

Approach:

- Used brute-force nested loop to compare every pair.

Copilot Prompt:
brute force two sum c#

Response Received:
Suggested double loop comparing sums of unique pairs.

Adjustments Made:

- Checked i != j to avoid using the same element twice.
- Returned empty array if no match is found.

Time Complexity: $O(n^2)$
Space Complexity: O(1)

Concepts Covered: Brute Force Search, Index Comparison

Edge Cases Handled:

- No valid pair exists
- Multiple pairs (returns the first one found)
- Same number used twice is avoided
- Target formed by identical values

Question 4: Find Maximum Product of Three Numbers

Problem Description:
Return the maximum product of any three numbers in the array.

Approach:

- Sort the array.
- Compare the product of the top three numbers and the product of the two smallest (possibly negative) and the largest.

Copilot Prompt:
find maximum product of three numbers in array c#

Response Received:
Suggested sorting and comparing edge value combinations.

Adjustments Made:

- Checked for array length < 3.
- Discussed effect of negative numbers.
- Added explanation of math logic.

Time Complexity: O(n log n)
Space Complexity: O(1)

Concepts Covered: Sorting, Negative Product Logic

Edge Cases Handled:

- Less than 3 elements (exception thrown)
- All negative numbers
- Mix of large positive and small negative numbers
- Duplicate elements
- Integer overflow (handled via checked and Overflow Exception)

Question 5: Decimal to Binary Conversion

Problem Description:
Convert a decimal number to a binary string.

Approach:

- Used .NET built-in Convert.ToString(n, 2).

Copilot Prompt:
convert decimal to binary string in c#

Response Received:
Recommended built-in method for base conversion.

Adjustments Made:

- Added error handling using try-catch.
- Verified correct conversion of 0.

Time Complexity: O(log n)
Space Complexity: O(log n) — based on length of binary string

Concepts Covered: Number Systems, Built-in Functions

Edge Cases Handled:

- Zero input (0 → "0")
- Powers of 2 (e.g., 8 → 1000)
- Negative values not expected per spec

Question 6: Find Minimum in Rotated Sorted Array

Problem Description:
Return the minimum value in a sorted array that has been rotated.

Approach:

- Applied binary search using pivot detection logic (nums[mid] > nums[right]).

Copilot Prompt:
find minimum in rotated sorted array using binary search c#

Response Received:
Recommended binary search logic using left/mid/right pointers.

Adjustments Made:

- Used l + (r - l) / 2 to avoid overflow.
- Documented step-by-step decisions.

Time Complexity: O(log n)
Space Complexity: O(1)

Concepts Covered: Binary Search, Rotation Pivot

Edge Cases Handled:

- Array not rotated (already sorted)
- Rotated at first or last index
- Single-element array
- Duplicates not allowed (based on original spec)

Question 7: Palindrome Number

Problem Description:
Check if an integer is a palindrome (reads same forward and backward).

Approach:

- Reverse digits using math, then compare with original number.

Copilot Prompt:
check if integer is palindrome without converting to string c#

Response Received:
Suggested mathematical reversal using % and /.

Adjustments Made:

- Handled negatives (return false).
- Explained digit-by-digit reversal.
- Avoided string conversion.

Time Complexity: O(log n)
Space Complexity: O(1)

Concepts Covered: Math Reversal, Control Flow

Edge Cases Handled:

- Negative numbers → false
- Numbers ending in 0 (e.g., 10 → false)
- Single-digit numbers → true
- Palindromes with even and odd digits

Question 8: Fibonacci Number

Problem Description:
Return the nth Fibonacci number using an iterative approach.

Approach:

- Iteratively compute values using a + b.

Copilot Prompt:
fibonacci number iterative c#

Response Received:
Suggested using loop with three variables: a, b, and result.

Adjustments Made:

- Covered base cases n = 0 and n = 1.
- Optimized with O(1) space by avoiding arrays or recursion.

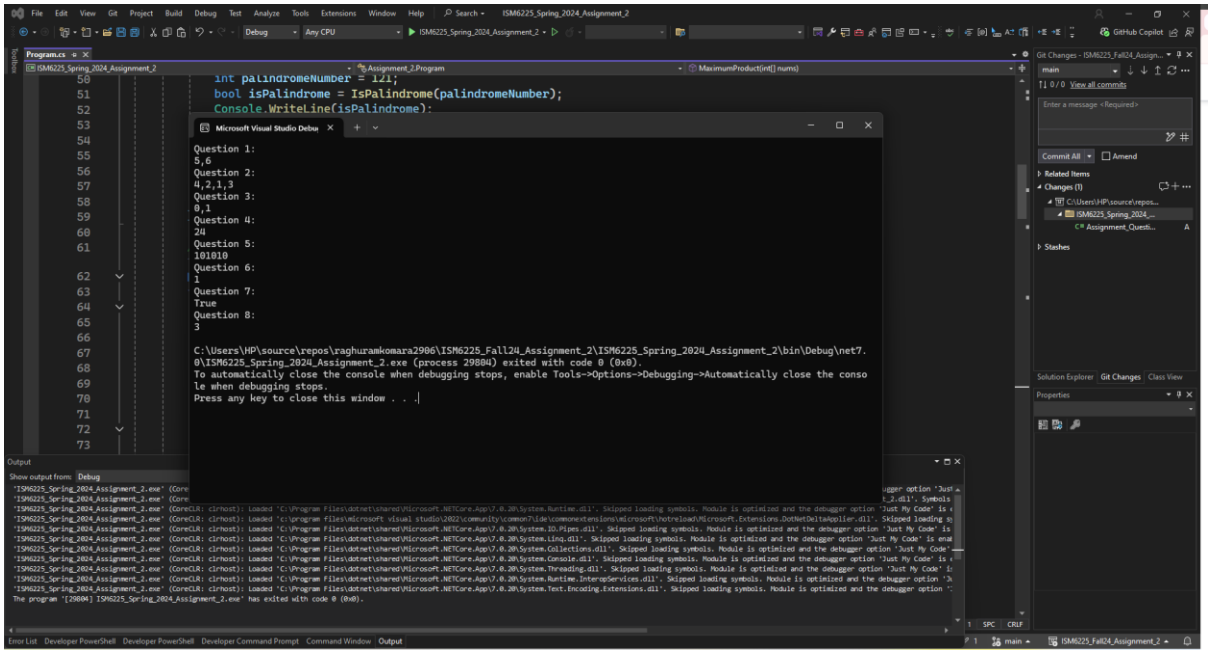
Time Complexity: O(n)
Space Complexity: O(1)

Concepts Covered: Iteration, Dynamic Programming (Bottom-Up)

Edge Cases Handled:

- n = 0 and n = 1
- Only positive integers allowed (per constraint)
- No recursion used (avoids stack overflow for large n)

Microsoft Visual Studio Debug Console: -



Git hub repository link: -

https://github.com/raghuramkomara2906/ISM6225_Fall24_Assignment_2.git