Assignment 2 Copilot Log

Martin Zuluaga

Change target framework

Prompt: how can I change the target framework for this document into 9.0 0r 8.0?

Answer:

Use the **.NET: Select Launch Target Framework** command to change the target framework.

Show in Command Palette

Ensure your project file (.csproj) specifies the desired target framework version, such as <TargetFramework>net8.0</TargetFramework> or <TargetFramework>net9.0</TargetFramework>. If the desired .NET SDK is not installed, use the **.NET: Install New .NET SDK** command to install it.

Show ".NET: Install New .NET SDK"

After making changes, use the **.NET: Restore Project** command to restore dependencies.

Show ".NET: Restore Project"

Implementation: I followed the steps and updated the framework.

Question 1

Not a prompt, but copilot suggested this code when I initiated the loop:

int index = Math.Abs(nums[i]) - 1;

// Mark the number as negative to indicate it has been seen

if (index >= 0 && index < nums.Length)

{

nums[index] = -Math.Abs(nums[index]);

}

I think that it is a good suggestion, but I changed it for something that I believe is simpler and only computes once for each index.

Question 2

Promp: help me understand and solve the following errors: solve this error for the current code: Unreachable code detected

error CS0161: 'Program.SortArrayByParity(int[])': not all code paths return a value

Response: The issue with your [SortArrayByParity](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) method is that the [return evens.ToArray();](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) statement is inside the for loop, which causes two problems:

1. **Unreachable Code Detected**: The [return evens.ToArray();](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) inside the loop makes the rest of the loop unreachable after the first iteration.
2. **Not All Code Paths Return a Value**: If the loop doesn't execute (e.g., if the input array is empty), the method doesn't return anything, which violates the method's return type.

**Explanation of Fixes**

1. **Moved**[return evens.ToArray()](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html):
   * The return statement is moved outside the for loop to ensure the entire array is processed before returning the result.
2. **Combined**[evens](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**and**[odds](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**After the Loop**:
   * The [evens.AddRange(odds)](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) statement is placed after the loop to merge the two lists once all elements are categorized.
3. **Ensured All Code Paths Return a Value**:
   * The method now always returns a value, even if the input array is empty.

**Why This Fix Works**

* The loop processes all elements in the input array, categorizing them into [evens](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) and [odds](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
* After the loop, the two lists are combined, and the result is returned.
* This ensures no unreachable code and guarantees that all code paths return a value.

Implementation: moved the statement outside of the loop.

Question 3

Prompt: is this the more efficient way to find the indexes for 2 numbers from nums that added together equal target? // Question 3: Two Sum public static int[] TwoSum(int[] nums, int target) { try { var map = new Dictionary<int, int>();

Answer: Yes, the current implementation of the [TwoSum](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) method is efficient and widely used. It has a time complexity of **O(n)** and a space complexity of **O(n)**, which is optimal for this problem. Here's why:

**How It Works:**

1. **Dictionary for Lookup**:
   * The [map](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) dictionary stores numbers as keys and their indices as values.
   * This allows for constant-time lookup to check if the complement (i.e., [target - nums[i]](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)) exists in the array.
2. **Single Pass Through the Array**:
   * The method iterates through the array once, checking for the complement and updating the dictionary simultaneously.
3. **Return Early**:
   * As soon as a valid pair is found, the method returns the indices, avoiding unnecessary computation.

**Code Explanation:**

public static int[] TwoSum(int[] nums, int target)

{

    try

    {

        var map = new Dictionary<int, int>(); // Stores numbers and their indices

        for (int i = 0; i < nums.Length; i++)

        {

            int complement = target - nums[i]; // Calculate the complement

            if (map.ContainsKey(complement)) // Check if complement exists in the dictionary

            {

                return new int[] { map[complement], i }; // Return indices of the two numbers

            }

            map[nums[i]] = i; // Add the current number and its index to the dictionary

        }

        return new int[0]; // Return an empty array if no solution is found

    }

    catch (Exception)

    {

        throw;

    }

}

**Why This is Efficient:**

1. **Time Complexity**:
   * The loop runs once through the array (O(n)).
   * Dictionary operations (lookup and insertion) are O(1) on average.
   * Total time complexity: **O(n)**.
2. **Space Complexity**:
   * The dictionary stores up to [n](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) elements in the worst case.
   * Total space complexity: **O(n)**.

**Alternative Approaches:**

* **Brute Force**:
  + Nested loops to check all pairs of numbers.
  + Time complexity: **O(n²)**.
  + This is much slower for large arrays.
* **Sorting + Two Pointers**:
  + Sort the array and use two pointers to find the pair.
  + Time complexity: **O(n log n)** (due to sorting).
  + This approach doesn't work if you need the original indices, as sorting changes the order.

**Conclusion:**

The current implementation is the most efficient way to solve the problem while preserving the original indices. It is both time-efficient and space-efficient for this type of problem.

Implementation: left code as it was.

Question 4

Prompt: explain the following code: public static int MaximumProduct(int[] nums) { try { // Edge cases= input not sorted; negative numbers will be considered, as they could contribute to the highest possible product Array.Sort(nums); // Sort ascending by default

Answer: he [MaximumProduct](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) method calculates the maximum product of three numbers in an integer array. Here's a detailed explanation of how it works:

**Key Considerations:**

* **Negative Numbers**:
  + Negative numbers are considered because their product can be positive when multiplied by another negative number.
  + For example, in the array [-10, -10, 5, 2], the maximum product is (-10) \* (-10) \* 5 = 500.
* **Sorting**:
  + Sorting simplifies the logic by allowing direct access to the smallest and largest numbers.

Implementation: I left the code as it was. The intention of my prompt was understanding if my implemented code made sense, and what was the logic someone would assume from reading the code. I left the code as it is, as I am confident this is a very direct and reliable way to solve the problem.

Question 5

Prompt: what are some ways you can convert a positive number into binary in c#

Answer: Here are some common ways to convert a positive number into binary in C#:

**1. Using a Loop (Manual Conversion):**

* This is the approach used in your [DecimalToBinary](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) method. It repeatedly divides the number by 2 and collects the remainders.

public static string DecimalToBinary(int decimalNumber)

{

    if (decimalNumber < 0)

        throw new ArgumentException("Provide a positive number");

    if (decimalNumber == 0)

        return "0";

    string binary = "";

    while (decimalNumber > 0)

    {

        int remainder = decimalNumber % 2;

        binary = remainder + binary;

        decimalNumber /= 2;

    }

    return binary;

}

**2. Using Convert.ToString:**

* The Convert.ToString method can directly convert a number to its binary representation.

public static string DecimalToBinary(int decimalNumber)

{

    if (decimalNumber < 0)

        throw new ArgumentException("Provide a positive number");

    return Convert.ToString(decimalNumber, 2); // Base 2 for binary

}

**3. Using Bitwise Operations:**

* You can use bitwise operations to extract each bit of the number.

public static string DecimalToBinary(int decimalNumber)

{

    if (decimalNumber < 0)

        throw new ArgumentException("Provide a positive number");

    if (decimalNumber == 0)

        return "0";

    string binary = "";

    while (decimalNumber > 0)

    {

        binary = ((decimalNumber & 1) == 1 ? "1" : "0") + binary; // Check the least significant bit

        decimalNumber >>= 1; // Right shift the number

    }

    return binary;

}

**4. Using LINQ (For Learning Purposes):**

* You can use LINQ to create a binary representation, though it's less efficient.

public static string DecimalToBinary(int decimalNumber)

{

    if (decimalNumber < 0)

        throw new ArgumentException("Provide a positive number");

    if (decimalNumber == 0)

        return "0";

    return string.Concat(

        Enumerable.Range(0, 32) // Assume 32-bit integer

                  .Select(i => (decimalNumber & (1 << (31 - i))) != 0 ? "1" : "0")

                  .SkipWhile(c => c == "0") // Remove leading zeros

    );

}

**5. Using Convert.ToString with Padding:**

* If you want a fixed-width binary string (e.g., 8-bit or 32-bit), you can pad the result.

public static string DecimalToBinary(int decimalNumber, int bitWidth = 8)

{

    if (decimalNumber < 0)

        throw new ArgumentException("Provide a positive number");

    return Convert.ToString(decimalNumber, 2).PadLeft(bitWidth, '0'); // Pad with leading zeros

}

**Comparison of Methods:**

| **Method** | **Advantages** | **Disadvantages** |
| --- | --- | --- |
| Manual Loop | Simple, easy to understand | Slightly verbose |
| Convert.ToString | Short and efficient | Limited to base 2 conversion |
| Bitwise Operations | Efficient and low-level | Harder to read for beginners |
| LINQ | Creative and functional programming style | Less efficient, overkill for this task |
| Convert.ToString + Padding | Useful for fixed-width binary strings | Requires specifying bit width |

**Recommended Approach:**

For most use cases, **Convert.ToString(decimalNumber, 2)** is the simplest and most efficient way to convert a number to binary in C#.

Implementation: I decided to go with the manual loop, as I think it has the basic logic, which aligns on how I was initially thinking on solving it.

Question 6

Prompt: is the binary search applied correctly? Is there a betther way to apply in this context: int left = 0; int right = nums.Length - 1;

            while (left < right)

            {

                int mid = left + (right - left) / 2;

                if (nums[mid] > nums[right])

                {

                    left = mid + 1;

                }

                else

                {

                    right = mid;

                }

            }

            return nums[left];

Answer: Yes, the binary search is applied correctly in this context. The code is designed to find the minimum element in a **rotated sorted array**, which is a common problem. Here's why the implementation works and whether there are better alternatives:

**How the Code Works:**

1. **Initial Setup**:
   * [left](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) and [right](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) pointers are initialized to the start and end of the array, respectively.
2. **Binary Search Loop**:
   * The loop continues until [left](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) equals [right](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), which means the search has narrowed down to a single element (the minimum).
3. **Midpoint Calculation**:
   * [mid](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) is calculated as [left + (right - left) / 2](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) to avoid potential integer overflow.
4. **Comparison Logic**:
   * If [nums[mid] > nums[right]](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), it means the minimum element is in the right half of the array (excluding [mid](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)), so [left](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) is updated to [mid + 1](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
   * Otherwise, the minimum element is in the left half (including [mid](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)), so [right](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) is updated to [mid](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
5. **Return the Minimum**:
   * When the loop exits, [left](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) points to the minimum element, which is returned as [nums[left]](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).

**Why This Works:**

* In a rotated sorted array, the minimum element is the only element that is smaller than its previous element (or the smallest in the array if no rotation occurred).
* The binary search leverages the sorted nature of the array to efficiently narrow down the search space.

**Example Walkthrough:**

**Input:**

int[] nums = { 3, 4, 5, 1, 2 };

**Execution:**

1. Initial: [left = 0](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), [right = 4](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
2. First iteration:
   * [mid = 2](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) ([nums[mid] = 5](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)).
   * [nums[mid] > nums[right]](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (5 > 2), so [left = mid + 1 = 3](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
3. Second iteration:
   * [mid = 3](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) ([nums[mid] = 1](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)).
   * [nums[mid] <= nums[right]](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (1 <= 2), so [right = mid = 3](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
4. Exit loop: [left = 3](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), [nums[left] = 1](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).

**Output:**

1

**Is There a Better Way?**

This implementation is already optimal for this problem. The time complexity is **O(log n)**, which is the best you can achieve for this type of problem. However, here are a few considerations:

1. **Edge Cases**:
   * If the array is not rotated (e.g., {1, 2, 3, 4, 5}), the code still works because [nums[mid] <= nums[right]](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) will always hold, and the loop will narrow down to the smallest element.
2. **Simplification**:
   * The code is already concise and efficient. There isn't a significantly "better" way to solve this problem without sacrificing clarity or performance.
3. **Alternative Approaches**:
   * **Linear Search**:
     + You could iterate through the array to find the minimum, but this would have a time complexity of **O(n)**, which is worse than the current **O(log n)**.
   * **Recursive Binary Search**:
     + You could implement the binary search recursively, but it wouldn't improve performance and might increase stack usage.

**Final Thoughts:**

The current implementation is correct, efficient, and well-suited for the problem. There is no need to change it unless you want to explore alternative styles (e.g., recursion).

Implementation: I left the code as it was, it the feedback suggested. Using binary searched as proposed during class.

Question 7

Prompt: Explain this code: public static bool IsPalindrome(int x) { try { // Edge case: negative number if (x < 0) return false; string original = x.ToString(); char[] reversedArray = original.ToCharArray(); Array.Reverse(reversedArray); string reversed = new string(reversedArray); return original == reversed; } catch (Exception) { throw; } }

Answer: The [IsPalindrome](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) method checks whether a given integer [x](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) is a palindrome. A palindrome is a number that reads the same backward as forward (e.g., 121, 1221).

### Code Explanation:

public static bool IsPalindrome(int x)

{

    try

    {

        // Edge case: negative number

        if (x < 0)

            return false;

1. **Negative Numbers**:
   * Negative numbers are not palindromes because the negative sign (-) at the beginning makes them asymmetric.
   * For example, -121 is not the same when reversed (121-).

        string original = x.ToString();

1. **Convert the Number to a String**:
   * The integer [x](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) is converted to its string representation using [ToString()](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
   * This makes it easier to reverse the number and compare it.

        char[] reversedArray = original.ToCharArray();

        Array.Reverse(reversedArray);

        string reversed = new string(reversedArray);

1. **Reverse the String**:
   * The string is converted to a character array using [ToCharArray()](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
   * The array is reversed in place using [Array.Reverse()](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
   * A new string is created from the reversed character array using the string constructor.

        return original == reversed;

1. **Compare Original and Reversed Strings**:
   * The method checks if the original string is equal to the reversed string.
   * If they are equal, the number is a palindrome, and the method returns true.
   * Otherwise, it returns false.

    }

    catch (Exception)

    {

        throw;

    }

}

1. **Exception Handling**:
   * The try-catch block is included to handle any unexpected errors, though no exceptions are likely in this implementation.

### Example Walkthrough:

#### Input:

int x = 121;

#### Execution:

1. [x](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) is not negative, so the method proceeds.
2. [original = "121"](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
3. [reversedArray = ['1', '2', '1']](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (after [ToCharArray()](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)).
4. [reversedArray](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) becomes ['1', '2', '1'] (after [Array.Reverse()](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)).
5. [reversed = "121"](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
6. [original == reversed](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) evaluates to true.

#### Output:

true

### Edge Cases:

1. **Negative Numbers**:
   * Input: -121
   * Output: false (negative numbers are not palindromes).
2. **Single-Digit Numbers**:
   * Input: 7
   * Output: true (single-digit numbers are always palindromes).
3. **Zero**:
   * Input: 0
   * Output: true (zero is a palindrome).

### Complexity:

1. **Time Complexity**:
   * Converting the number to a string and reversing it both take **O(n)**, where [n](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) is the number of digits in [x](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
2. **Space Complexity**:
   * The character array used for reversing takes **O(n)** space.

### Alternative Approach (Without String Conversion):

If you want to avoid converting the number to a string, you can reverse the integer mathematically:

public static bool IsPalindrome(int x)

{

    if (x < 0)

        return false;

    int original = x;

    int reversed = 0;

    while (x > 0)

    {

        int digit = x % 10; // Extract the last digit

        reversed = reversed \* 10 + digit; // Build the reversed number

        x /= 10; // Remove the last digit

    }

    return original == reversed;

}

This approach avoids the overhead of string manipulation and has the same time complexity of **O(n)**.

Implementation: honestly, the alternative solution provided is more effcient, as it does not need to convert to string as noted in the feedback. I decided to stay with the code I initially had, as it aligns with my own thought process on how to solve it.

Question 8

Prompt: i made the initial set up to determine the fibonacci value of number n, please help me complete the code with relevant steps: // Edge case: negative number if (n < 0) throw new ArgumentException("n must be non-negative");

            if (n == 0)

                return 0;

            if (n == 1)

                return 1;

Program.cs:262-269

### GitHub Copilot

Answer: To complete the Fibonacci function, you can use either an **iterative approach** or a **recursive approach**. Below is the iterative approach, which is more efficient in terms of time and space complexity compared to recursion.

### Completed Code (Iterative Approach):

public static int Fibonacci(int n)

{

    try

    {

        // Edge case: negative number

        if (n < 0)

            throw new ArgumentException("n must be non-negative");

        // Base cases

        if (n == 0)

            return 0;

        if (n == 1)

            return 1;

        // Iterative calculation of Fibonacci numbers

        int prev1 = 0; // F(0)

        int prev2 = 1; // F(1)

        int current = 0;

        for (int i = 2; i <= n; i++)

        {

            current = prev1 + prev2; // F(n) = F(n-1) + F(n-2)

            prev1 = prev2;          // Update F(n-1)

            prev2 = current;        // Update F(n)

        }

        return current;

    }

    catch (Exception)

    {

        throw;

    }

}

### Explanation of the Code:

1. **Edge Cases**:
   * If [n < 0](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), an exception is thrown because Fibonacci is undefined for negative numbers.
   * If [n == 0](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), the result is 0.
   * If [n == 1](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), the result is 1.
2. **Iterative Calculation**:
   * Start with the first two Fibonacci numbers: F(0) = 0 and F(1) = 1.
   * Use a loop to calculate Fibonacci numbers from F(2) to F(n):
     + current = prev1 + prev2 calculates the current Fibonacci number.
     + Update prev1 and prev2 for the next iteration.
3. **Return the Result**:
   * After the loop, current holds the Fibonacci number for [n](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).

### Example Walkthrough:

#### Input:

int n = 5;

#### Execution:

1. Initialize: prev1 = 0, prev2 = 1.
2. Loop:
   * [i = 2](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): current = 0 + 1 = 1, update prev1 = 1, prev2 = 1.
   * [i = 3](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): current = 1 + 1 = 2, update prev1 = 1, prev2 = 2.
   * [i = 4](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): current = 1 + 2 = 3, update prev1 = 2, prev2 = 3.
   * [i = 5](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): current = 2 + 3 = 5, update prev1 = 3, prev2 = 5.
3. Return current = 5.

#### Output:

5

### Alternative: Recursive Approach

If you prefer recursion, here’s how it would look:

public static int Fibonacci(int n)

{

    if (n < 0)

        throw new ArgumentException("n must be non-negative");

    if (n == 0)

        return 0;

    if (n == 1)

        return 1;

    return Fibonacci(n - 1) + Fibonacci(n - 2);

}

#### Note:

* The recursive approach is less efficient due to repeated calculations, resulting in **O(2^n)** time complexity.
* For large [n](vscode-file://vscode-app/private/var/folders/1b/px5lkz957k14_j9qmt9tyfvh0000gn/T/AppTranslocation/F9978B71-EDA6-4519-ABA4-8CE212BE5A77/d/Visual%20Studio%20Code-3.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), this approach is not recommended unless optimized with **memoization** or **dynamic programming**.

### Final Recommendation:

The **iterative approach** is the most efficient and should be used for calculating Fibonacci numbers in this context.

Implementation: proceeded with the first approach, due to its efficiency, as suggested by copilot.