Lesson 06 Demo 04

Analyzing and Improving Space Complexity of Code Using Generative AI

Objective: To analyze and optimize the utility function for space complexity in Expressgo's application code using GitHub Copilot

Tools required: Visual Studio, GitHub Copilot, and POSTMAN

Prerequisites: Refer to Lesson 06, Demo 03 for instructions on downloading, executing, and improving the time complexity of the Expressgo project

Steps to be followed:

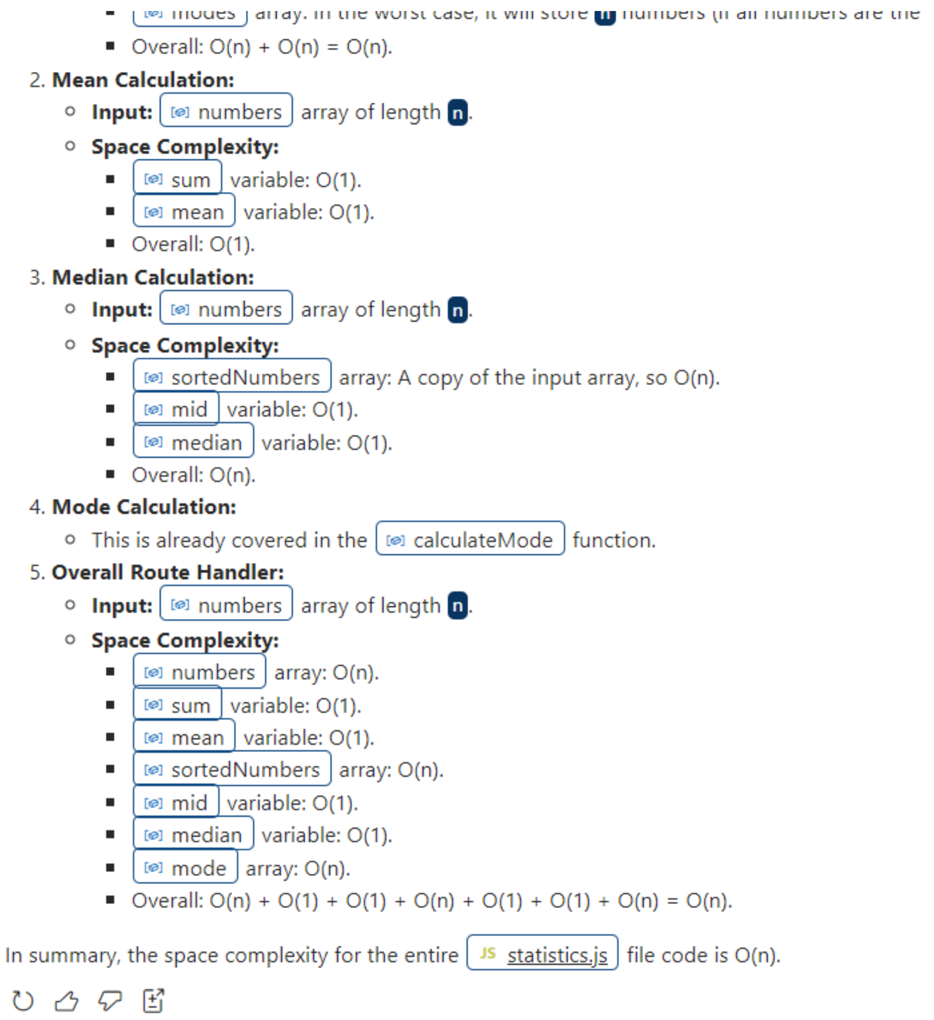
1. Analyze and optimize the space complexity of the existing code using GitHub Copilot

Note: The generative AI tool used in this exercise can produce varied outputs even when presented with similar prompts. Thus, you may get different outputs for the same prompt.

Step 1: Analyze and optimize the space complexity of the existing code using GitHub Copilot

1.1 Provide the following prompt in GitHub Copilot to analyze the space complexity for the existing code:
Now, let's calculate the space complexity for all these APIs.

@workspace /fix Now, let's calculate the space complexity of statistic.js file code

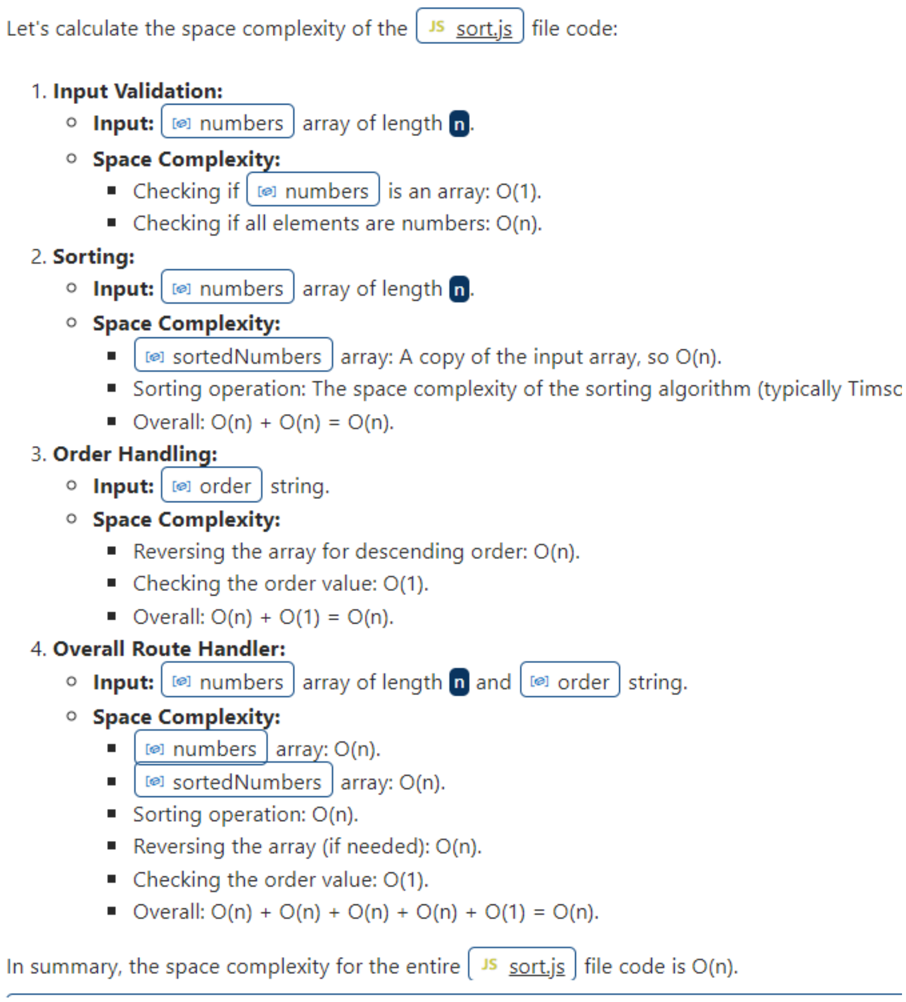


- modes array: In the worst case, it will store n numbers (if all numbers are the
- Overall: O(n) + O(n) = O(n).

2. **Mean Calculation:**
   - **Input:** numbers array of length n.
   - **Space Complexity:**
     - sum variable: O(1).
     - mean variable: O(1).
     - Overall: O(1).
3. **Median Calculation:**
   - **Input:** numbers array of length n.
   - **Space Complexity:**
     - sortedNumbers array: A copy of the input array, so O(n).
     - mid variable: O(1).
     - median variable: O(1).
     - Overall: O(n).
4. **Mode Calculation:**
   - This is already covered in the calculateMode function.
5. **Overall Route Handler:**
   - **Input:** numbers array of length n.
   - **Space Complexity:**
     - numbers array: O(n).
     - sum variable: O(1).
     - mean variable: O(1).
     - sortedNumbers array: O(n).
     - mid variable: O(1).
     - median variable: O(1).
     - mode array: O(n).
     - Overall: O(n) + O(1) + O(1) + O(n) + O(1) + O(1) + O(n) = O(n).

In summary, the space complexity for the entire JS statistics.js file code is O(n).

1.2 @workspace /fix Now, let's calculate the space complexity of statistic.js file code

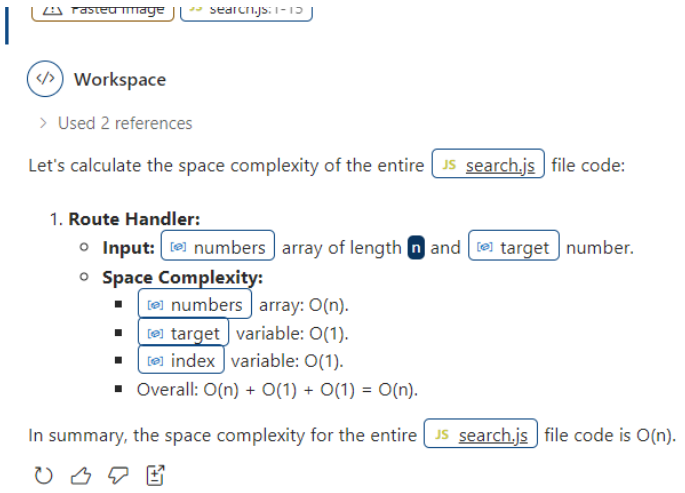


Let's calculate the space complexity of the sort.js file code:

1. **Input Validation:**
   - **Input:** numbers array of length n.
   - **Space Complexity:**
     - Checking if numbers is an array: O(1).
     - Checking if all elements are numbers: O(n).
2. **Sorting:**
   - **Input:** numbers array of length n.
   - **Space Complexity:**
     - sortedNumbers array: A copy of the input array, so O(n).
     - Sorting operation: The space complexity of the sorting algorithm (typically Timsc
     - Overall: O(n) + O(n) = O(n).
3. **Order Handling:**
   - **Input:** order string.
   - **Space Complexity:**
     - Reversing the array for descending order: O(n).
     - Checking the order value: O(1).
     - Overall: O(n) + O(1) = O(n).
4. **Overall Route Handler:**
   - **Input:** numbers array of length n and order string.
   - **Space Complexity:**
     - numbers array: O(n).
     - sortedNumbers array: O(n).
     - Sorting operation: O(n).
     - Reversing the array (if needed): O(n).
     - Checking the order value: O(1).
     - Overall: O(n) + O(n) + O(n) + O(n) + O(1) = O(n).

In summary, the space complexity for the entire sort.js file code is O(n).

1.3 @workspace /fix Now, let's calculate the space complexity of search.js file code

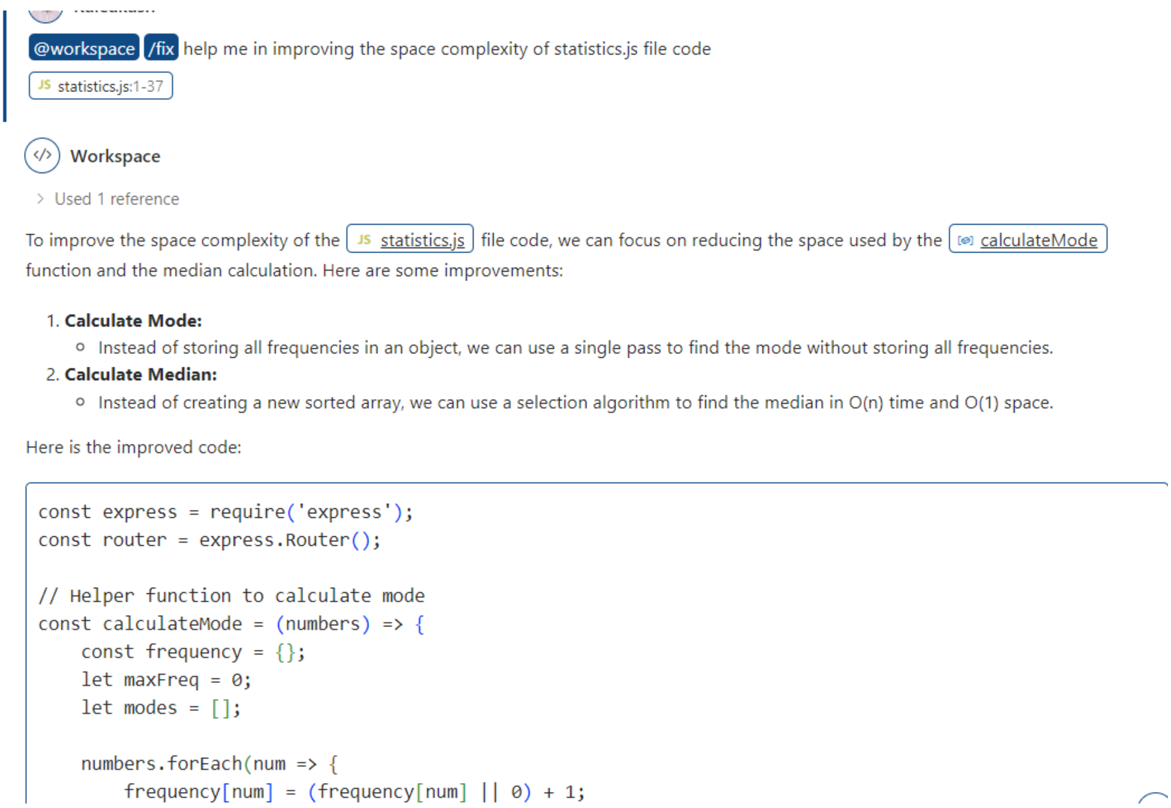


Let's calculate the space complexity of the entire search.js file code:

1. **Route Handler:**
   - **Input:** numbers array of length n and target number.
   - **Space Complexity:**
     - numbers array: O(n).
     - target variable: O(1).
     - index variable: O(1).
     - Overall: O(n) + O(1) + O(1) = O(n).

In summary, the space complexity for the entire search.js file code is O(n).

1.4 @workspace /fix help me in improving the space complexity of statistics.js file code



@workspace /fix help me in improving the space complexity of statistics.js file code

statistics.js:1-37

Workspace

Used 1 reference

To improve the space complexity of the statistics.js file code, we can focus on reducing the space used by the calculateMode function and the median calculation. Here are some improvements:

1. **Calculate Mode:**
   - Instead of storing all frequencies in an object, we can use a single pass to find the mode without storing all frequencies.
2. **Calculate Median:**
   - Instead of creating a new sorted array, we can use a selection algorithm to find the median in O(n) time and O(1) space.

Here is the improved code:

```
const express = require('express');
const router = express.Router();

// Helper function to calculate mode
const calculateMode = (numbers) => {
    const frequency = {};
    let maxFreq = 0;
    let modes = [];

    numbers.forEach(num => {
        frequency[num] = (frequency[num] || 0) + 1;
```

1.5 github copilot generated code replace with existing code and test it.

To improve the space complexity of the statistics.js file code, we can focus on reducing the space used by the calculateMode function and the median calculation. Here are some improvements:

1. **Calculate Mode:**
   - Instead of storing all frequencies in an object, we can use a single pass to find the mode without storing all frequencies.
2. **Calculate Median:**
   - Instead of creating a new sorted array, we can use a selection algorithm to find the median in O(n) time and O(1) space.

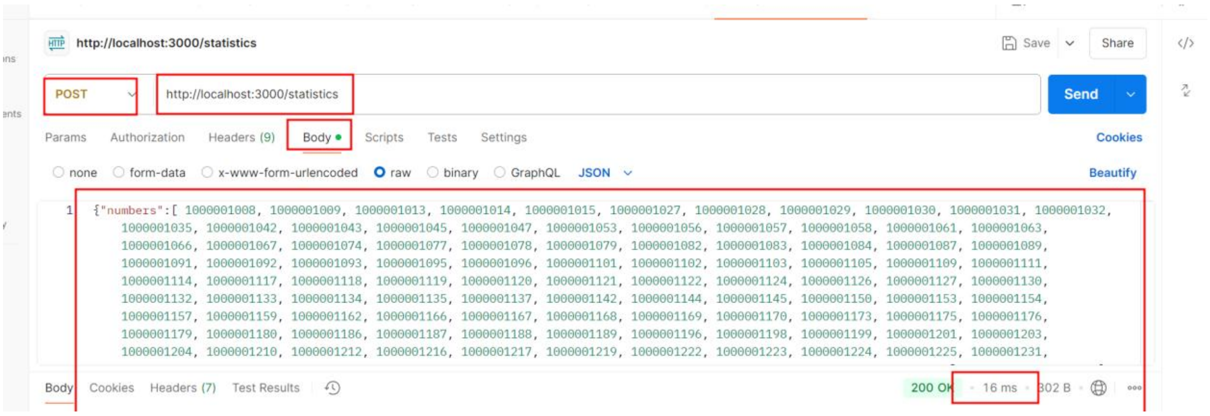
Here is the improved code:

statistics.js

```
const express = require('express');
const router = express.Router();

// Helper function to calculate mode
```

```
const calculateMode = (numbers) => {
  const frequency = {};
  let maxFreq = 0;
  let modes = [];

  numbers.forEach(num => {
    frequency[num] = (frequency[num] || 0) + 1;
    if (frequency[num] > maxFreq) {
      maxFreq = frequency[num];
      modes = [num];
    } else if (frequency[num] === maxFreq) {
      modes.push(num);
    }
  });

  return modes.length === numbers.length ? [] : modes; // If all values occur once, return
empty array
};

// Helper function to find the k-th smallest element (Quickselect algorithm)
const quickselect = (arr, k) => {
  const partition = (left, right, pivotIndex) => {
    const pivotValue = arr[pivotIndex];
    [arr[pivotIndex], arr[right]] = [arr[right], arr[pivotIndex]];
    let storeIndex = left;
    for (let i = left; i < right; i++) {
      if (arr[i] < pivotValue) {
        [arr[storeIndex], arr[i]] = [arr[i], arr[storeIndex]];
        storeIndex++;
      }
    }
    [arr[right], arr[storeIndex]] = [arr[storeIndex], arr[right]];
    return storeIndex;
  };

  let left = 0, right = arr.length - 1;
  while (left <= right) {
    const pivotIndex = Math.floor(Math.random() * (right - left + 1)) + left;
    const newPivotIndex = partition(left, right, pivotIndex);
    if (newPivotIndex === k) {
      return arr[newPivotIndex];
    } else if (newPivotIndex < k) {
      left = newPivotIndex + 1;
    } else {
      right = newPivotIndex - 1;
    }
  }
};

// Statistics Route
router.post('/', (req, res) => {
  const { numbers } = req.body;
  if (!Array.isArray(numbers) || numbers.length === 0) {
    return res.status(400).json({ error: 'Invalid input, expected a non-empty array' });
  }

  // Calculate Mean
  const sum = numbers.reduce((acc, num) => acc + num, 0);
  const mean = sum / numbers.length;

  // Calculate Median
  const mid = Math.floor(numbers.length / 2);
  const median = numbers.length % 2 === 0
    ? (quickselect(numbers, mid - 1) + quickselect(numbers, mid)) / 2
    : quickselect(numbers, mid);

  // Calculate Mode
  const mode = calculateMode(numbers);

  res.json({ mean, median, mode });
});

module.exports = router;
```

Tested with new code



```
{"numbers":[ 1000001008, 1000001009, 1000001013, 1000001014, 1000001015, 1000001027, 1000001028, 1000001029, 1000001030, 1000001031, 1000001032,
    1000001035, 1000001042, 1000001043, 1000001045, 1000001047, 1000001053, 1000001056, 1000001057, 1000001058, 1000001061, 1000001063,
    1000001066, 1000001067, 1000001074, 1000001077, 1000001078, 1000001079, 1000001082, 1000001083, 1000001084, 1000001087, 1000001089,
    1000001091, 1000001092, 1000001093, 1000001095, 1000001096, 1000001101, 1000001102, 1000001103, 1000001105, 1000001109, 1000001111,
    1000001114, 1000001117, 1000001118, 1000001119, 1000001120, 1000001121, 1000001122, 1000001124, 1000001126, 1000001127, 1000001130,
    1000001132, 1000001133, 1000001134, 1000001135, 1000001137, 1000001142, 1000001144, 1000001145, 1000001150, 1000001153, 1000001154,
    1000001157, 1000001159, 1000001162, 1000001166, 1000001167, 1000001168, 1000001169, 1000001170, 1000001173, 1000001175, 1000001176,
    1000001179, 1000001180, 1000001186, 1000001187, 1000001188, 1000001189, 1000001196, 1000001198, 1000001199, 1000001201, 1000001203,
    1000001204, 1000001210, 1000001212, 1000001216, 1000001217, 1000001219, 1000001222, 1000001223, 1000001224, 1000001225, 1000001231,
```

1.6 **@workspace /fix help me in improving the space complexity of sort.js file code**

To improve the space complexity of the sort.js file code, we can avoid creating a copy of the numbers array and sort the array in place. Additionally, we can avoid using the reverse method by directly sorting in descending order if needed.

Here is the improved code:

sort.js

```
const express = require('express');
const router = express.Router();

router.post('/', (req, res) => {
  const { numbers, order = 'asc' } = req.body;

  if (!Array.isArray(numbers) || numbers.some(num => typeof num !== 'number')) {
    return res.status(400).json({ error: 'Invalid input, expected an array of numbers' });
  }

  // Sort the array in place
  numbers.sort((a, b) => order === 'desc' ? b - a : a - b);

  res.json({ sortedNumbers: numbers });
});

module.exports = router;
```

1.7 @workspace /fix help me in improving the space complexity of search.js file code

@workspace /fix help me in improving the space complexity of search.js file code

To improve the space complexity of the sort.js file code, we can avoid creating a copy of the numbers array for sorting. Instead, we can sort the array in place. This will reduce the space complexity from O(n) to O(1).
Here is the improved code:

search.js

```
const express = require('express');
const router = express.Router();
```

```
router.post('/', (req, res) => {
  const { numbers, order = 'asc' } = req.body;

  if (!Array.isArray(numbers) || numbers.some(num => typeof num !== 'number')) {
    return res.status(400).json({ error: 'Invalid input, expected an array of numbers' });
  }

  // Sort the array in place
  numbers.sort((a, b) => a - b);

  if (order === 'desc') {
    numbers.reverse(); // Reverse for descending order
  } else if (order !== 'asc') {
    return res.status(400).json({ error: 'Invalid order, use "asc" or "desc"' });
  }

  res.json({ sortedNumbers: numbers });
});

module.exports = router;
```

By following these steps, you have successfully utilized generative to analyze and optimize the utility function for space complexity in Expressgo's application code using GitHub Copilot, thereby improving the efficiency and scalability of memory usage.