**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.

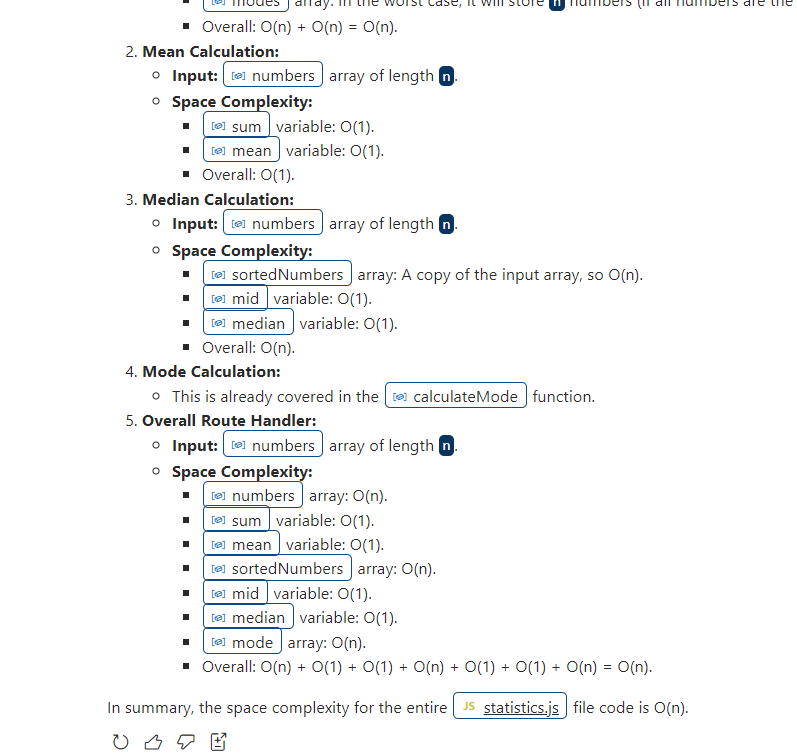
**Step 1: Analyze and optimize the space complexity of the existing code using**

**GitHub Copilot**

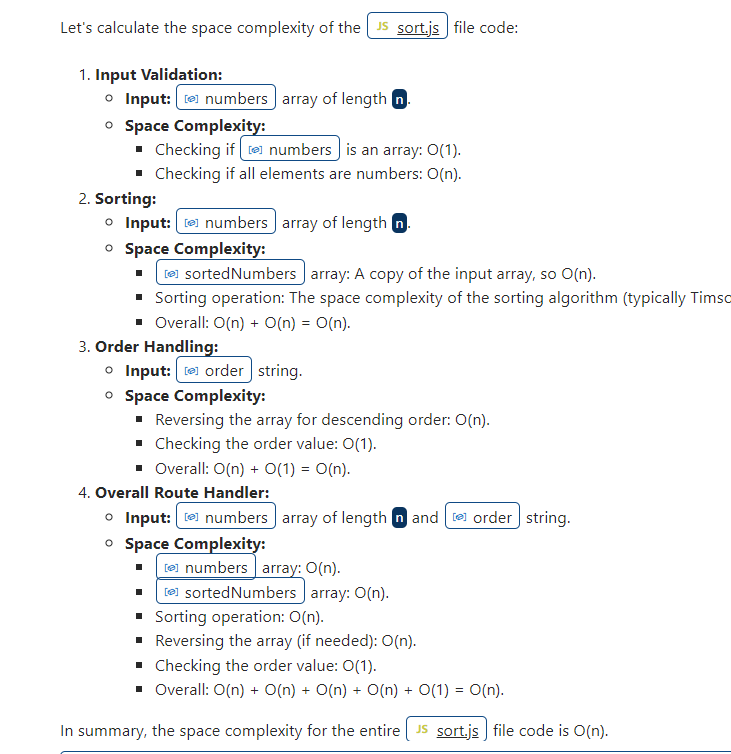
* 1. Provide the following prompt in GitHub Copilot to analyze the space complexity for the existing code:

**Now, let's calculate the space complexity for all these APIs.**

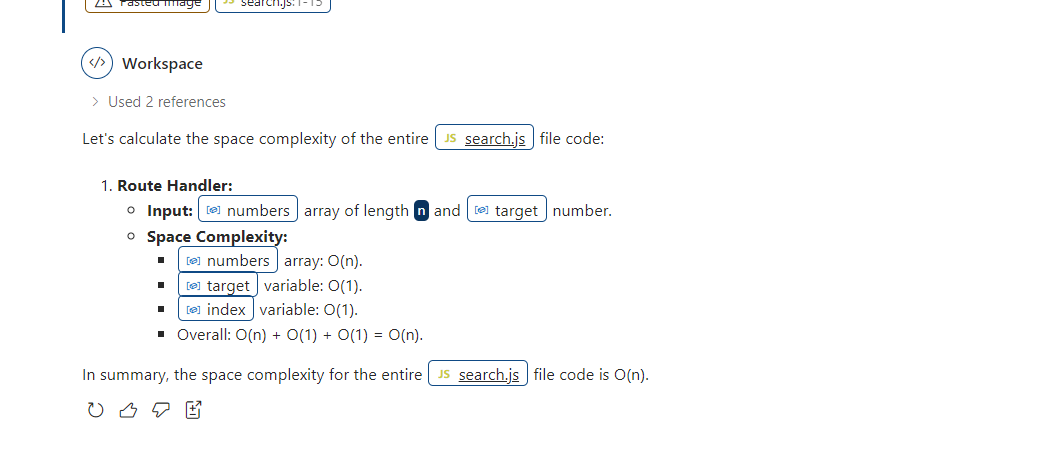
@workspace /fix Now, let's calculate the space complexity of statistic.js file code



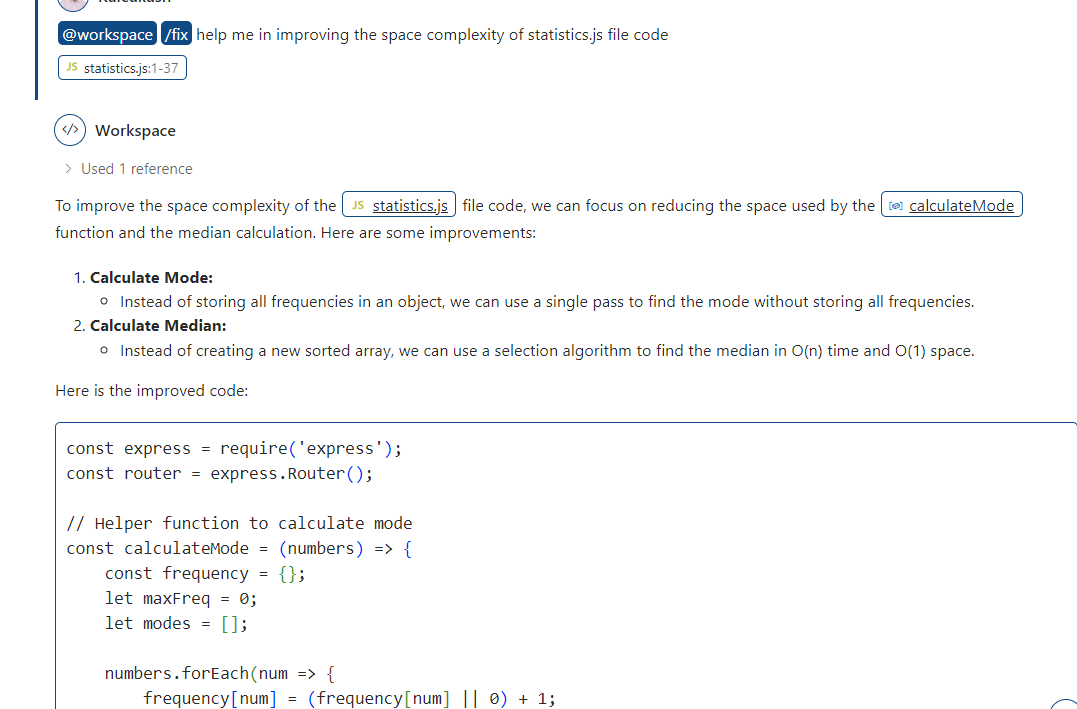
* 1. @workspace /fix Now, let's calculate the space complexity of statistic.js file code



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



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



* 1. github copilot generated code replace with existing code and test it.

To improve the space complexity of the [statistics.js](vscode-file://vscode-app/c:/Users/akash/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) file code, we can focus on reducing the space used by the [calculateMode](vscode-file://vscode-app/c:/Users/akash/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) 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

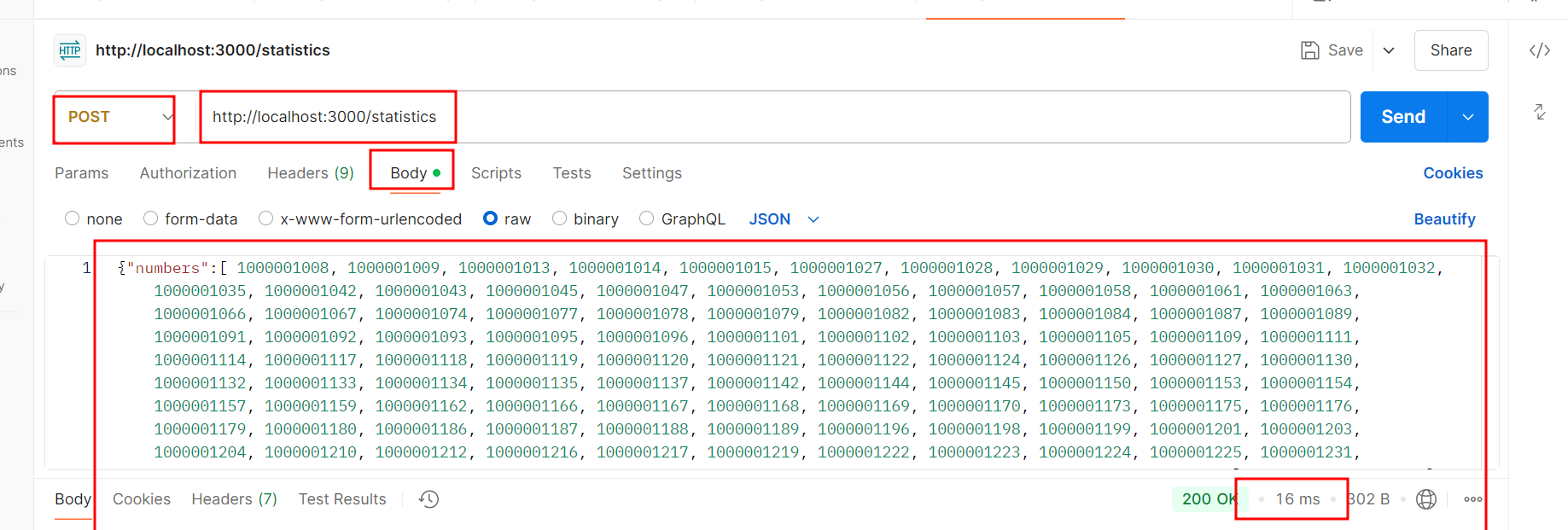
    const mode = calculateMode(numbers);

    res.json({ mean, median, mode });

});

module.exports = router;

Tested with new code



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](vscode-file://vscode-app/c:/Users/akash/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) file code, we can avoid creating a copy of the [numbers](vscode-file://vscode-app/c:/Users/akash/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) array and sort the array in place. Additionally, we can avoid using the [reverse](vscode-file://vscode-app/c:/Users/akash/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) 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](vscode-file://vscode-app/c:/Users/akash/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) file code, we can avoid creating a copy of the [numbers](vscode-file://vscode-app/c:/Users/akash/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) 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.