Step 1 By Hand: 6 minutes
Step 2 Approach: 10 minutes
Step 3 Pseudocode: 15 minutes
Step 4 Copilot: 3 minutes
Step 5 Compare and Contrast: 11 minutes
Step 6 Update: 10 minutes
Step 7 Trace: 15 minutes
Step 8 Efficiency: 3 minutes

Step 1.

[12, 25, 37, 18, 42, 55, 29, 61, 34, 45, 50, 41, 39, 33]

The sums/averages of the groups of ten:
358/35.8
396/39.6
412/41.2
414/41.4
429/42.9

Step 2.

To find the highest average sub-array of size 10, I will use a sliding window technique. First, I will compute the sum of the first 10 elements. Then, I will slide the window by removing the first element of the previous sub-array and adding the next element in the list. I will keep track of the maximum sum encountered and calculate the highest average by dividing the max sum by 10.

Step 3.

```
FUNCTION find_max_average(array, k):
    Initialize max_sum and current_sum to 0
    Compute the sum of the first k elements and store it in current_sum
    Set max_sum to current_sum

    FOR i FROM k TO length of array - 1:
        Subtract array[i - k] from current_sum (remove the leftmost element)
        Add array[i] to current_sum (add the new element)
        Update max_sum if current_sum is greater

    RETURN max_sum / k
```

Step 4.

```
// Function to find the highest average sub-array of size 10
function findHighestAverageSubArray(arr):
if length(arr) < 10:
    return "Array is too small"

// Compute the sum of the first 10 elements
currentSum = 0
for i = 0 to 9:
    currentSum = currentSum + arr[i]
// Initialize maxSum with the sum of the first sub-array
maxSum = currentSum

// Slide the window through the array
for i = 10 to length(arr) - 1:
    currentSum = currentSum - arr[i - 10] + arr[i]
    maxSum = max(maxSum, currentSum)

// Calculate and return the highest average
highestAverage = maxSum / 10
return highestAverage
```

Step 5:

My pseudocode:

Pros:

My pseudocode is clear, compact, and concise. It minimizes the number of computations and focuses solely on the core logic.

Cons:

It doesn't account for any edge cases or arrays that are too small to form a valid sub-array.

Copilot's pseudocode:

Pros:

It has more descriptive variable names and handles errors better, checking whether the array is the right size or not.

Cons:

It's more detailed than necessary and the checks for length could be seen as redundant.

My Improvement

Mine can double check and validate the length of the array.

Copilot's Improvement

Instead of using a loop for the sum, you can just use the built-in sum operator in Python to improve efficiency.

Step 4 match Step 1

Yes it matches

Step 6.

```
    FUNCTION find_max_average(array, k):
        IF length(array) < k:
            RETURN "Array is too small"

        Initialize current_sum to sum of first k elements
        Initialize max_sum to current_sum

        FOR i FROM k TO length of array - 1:
            Subtract array[i - k] from current_sum (remove the leftmost element)
            Add array[i] to current_sum (add the new element)
            Update max_sum if current_sum is greater than max_sum

        RETURN max_sum / k
```

Step 7.

Label	current_sum	max_sum	i	sub-array (window)
A	165	165	N/A	[41, 45, 47, 32]
B	165	165	N/A	[41, 45, 47, 32]
C	165	165	4	[45, 47, 32, 49]
D	173	173	4	[45, 47, 32, 49]
E	168	173	5	[47, 32, 49, 40]
F	168	173	5	[47, 32, 49, 40]
G	153	173	6	[32, 49, 40, 32]
H	153	173	6	[32, 49, 40, 32]
I	173	173	N/A	N/A

Step 8.

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