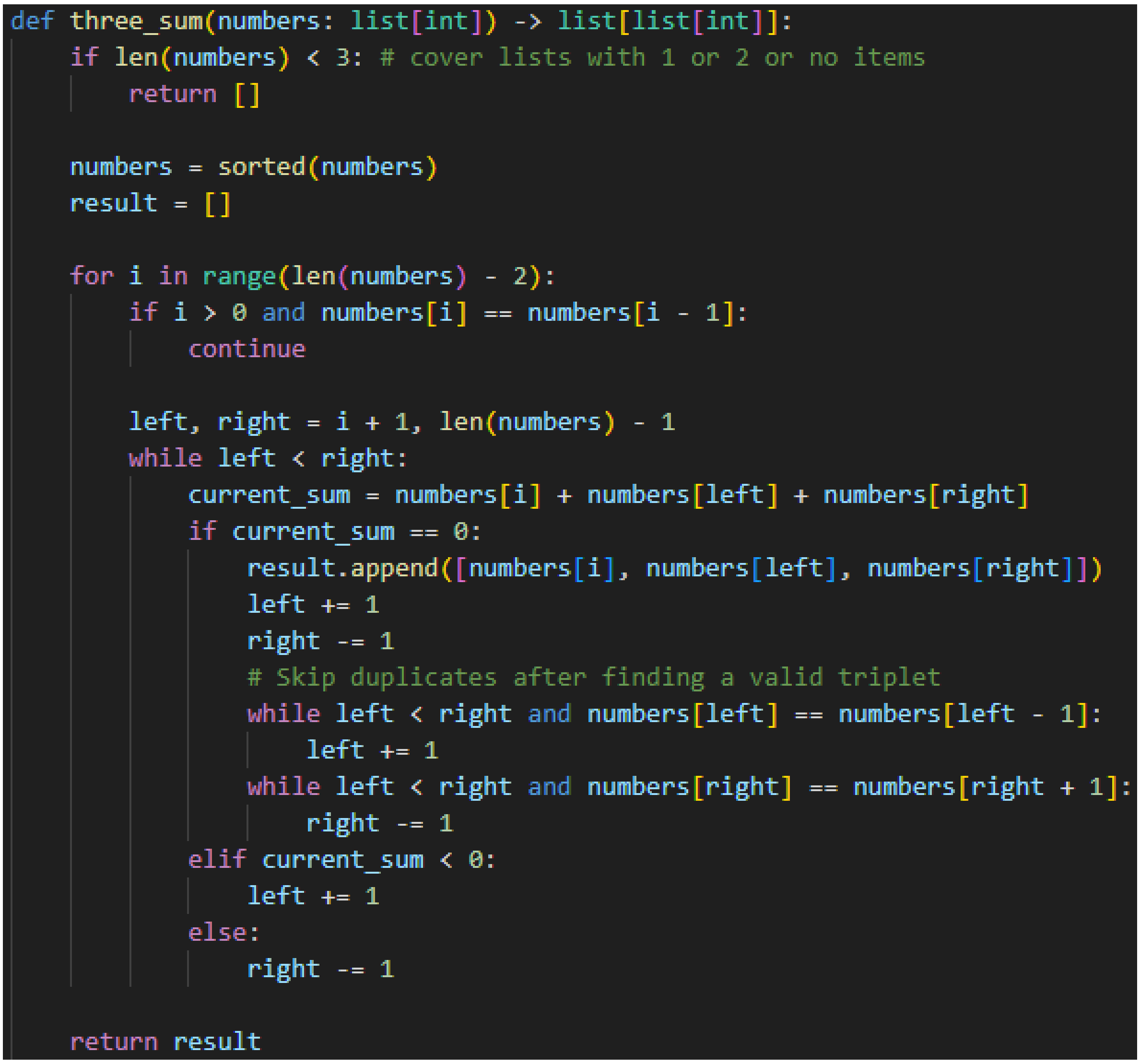
Problem Statement:

Given an integer array numbers, return all unique triplets [numbers[i], numbers[j], numbers[k]] such that the sum of the three numbers is zero (i.e., numbers[i] + numbers[j] + numbers[k] == 0). Each triplet must consist of distinct indices, and the output should not contain any duplicate triplets. The solution must return the triplets in any order.

Solution 1

Approach uses the two-pointer method for each number in the list. The pointers are also used to help reduce runtime by identifying whether the same integers are adjacent to them.



```python
def three_sum(numbers: list[int]) -> list[list[int]]:
    if len(numbers) < 3: # cover lists with 1 or 2 or no items
        return []

    numbers = sorted(numbers)
    result = []

    for i in range(len(numbers) - 2):
        if i > 0 and numbers[i] == numbers[i - 1]:
            continue

        left, right = i + 1, len(numbers) - 1
        while left < right:
            current_sum = numbers[i] + numbers[left] + numbers[right]
            if current_sum == 0:
                result.append([numbers[i], numbers[left], numbers[right]])
                left += 1
                right -= 1
                # Skip duplicates after finding a valid triplet
                while left < right and numbers[left] == numbers[left - 1]:
                    left += 1
                while left < right and numbers[right] == numbers[right + 1]:
                    right -= 1
            elif current_sum < 0:
                left += 1
            else:
                right -= 1

    return result
```

Step-by-Step Breakdown

1. **Input:**
    - numbers: A list of integers [-1, 0, 1, 2, -1, -4].

2. **Intermittent step 1:**
   - Edge Case Check:
     - If the length of numbers is less than 3, immediately return an empty list []
       - Fewer than three numbers cannot form a valid triplet.
3. **Intermittent step 2:**
   - **Sort the Input:**
     - Sort numbers to simplify duplicate management and to allow the use of a two-pointer approach.
   - **Initialize Result List:**
     - Create an empty list result to store the valid triplets.
   - **Iterate Through the Array each index i from 0 to len(numbers) - 3:**
     - **Skip Duplicates for i:**
       - If i > 0 and numbers[i] equals numbers[i - 1], skip the iteration to avoid duplicate triplets.
     - **Set Up Two Pointers:**
       - Initialize left to i + 1 and right to len(numbers) - 1.
     - **Two-Pointer Search:**
         - While left < right, calculate:

```
current_sum = numbers[i] + numbers[left] + numbers[right]
```

     - **Check Sum Against Target (0):**
       - **If current_sum is 0:**
         - Append [numbers[i], numbers[left], numbers[right]] to result.
         - Increment left and decrement right to search for any other potential triplets.
       - **Skip Duplicates:**
         - Continue moving left forward while the new numbers[left] equals the previous value.
         - Similarly, continue moving right backward while the new numbers[right] equals the previous value.
     - **If current_sum is less than 0:**
       - Increment left to try a larger sum.
     - **If current_sum is greater than 0:**
       - Decrement right to try a smaller sum.

4. **Output:**
   - Return the list result containing all unique triplets that sum to zero.
5. **Efficiency:**
   - **Time Complexity: $O(n^2)$**
     - Sorting takes $O(n \log n)$, and the two-pointer approach inside a loop results in $O(n^2)$ in the worst case.
   - **Space Complexity: $O(1)$ (excluding the space required for the output)**
     - Only a few pointers and loop variables are used, with no additional data structures for processing.

Visual Flow Diagram

```
                Input: numbers = [-1, 0, 1, 2, -1, -4]
                                │
                                ▼
             ┌─────────────────────────────────┐
             │ Check: len(numbers) < 3?        │
             └─────────────────────────────────┘
                                │
                     ┌──────────┴───────────┐
                    Yes                     No
                     │                      │
                     ▼                      ▼
              Return []            Sort the numbers
                                         │
                                         ▼
                            Sorted numbers: [-4, -1, -1, 0, 1, 2]
                                         │
                                         ▼
                          For i in range(0, len(numbers)-2)
                                         │
                                         ▼
                       ┌────────────────────────────────────────┐
                       │ If i > 0 and numbers[i]==numbers[i-1]  │
                       └────────────────────────────────────────┘
                                         │
                         (Skip duplicates for i if needed)
                                         │
                                         ▼
                         Set left = i+1, right = len(numbers)-1
                                         │
                                         ▼
                         ┌─────────────────────────────┐
                         │ While left < right:         │
                         └─────────────────────────────┘
                                         │
                                         ▼
           Calculate current_sum = numbers[i] + numbers[left] + numbers[right]
                                         │
                                         ▼
                 ┌──────────────────────────────────────────────┐
                 │ Is current_sum == 0?                         │
                 └──────────────────────────────────────────────┘
                           │                          │
                          Yes                         No
                           │                          │
                           ▼                          ▼
                    Append triplet
           [num[i], num[left], num[right]]    Is current_sum < 0?
                           │                          │
                           ▼                          ▼
        Increment left, decrement right     If Yes: Increment left
                           │                          │
         Skip duplicates for left/right       Else: Decrement right
                           │                          │
                           ▼                          ▼
                        Loop until left >= right
                                         │
                                         ▼
                         Continue loop for next i value
                                         │
                                         ▼
                            End of loop: Return result
```