**Problem: Subarray with Given Sum**

**Question:** Given an unsorted array of nonnegative integers and an integer sum, find a continuous subarray that adds up to sum. If there are multiple subarrays with the same sum, return any one of them.

**Example with Detailed Iteration**

Given:

* arr = [1, 2, 3, 7, 5]
* sum = 12

We will check all possible subarrays to see if any of them sum up to the given sum.

Here's the brute force code again, followed by step-by-step iteration:

function subarrayWithGivenSumBruteForce(arr, sum) {

for (let start = 0; start < arr.length; start++) {

let currentSum = 0;

for (let end = start; end < arr.length; end++) {

currentSum += arr[end];

if (currentSum === sum) {

return arr.slice(start, end + 1);

}

}

}

return []; // Return an empty array if no subarray with given sum is found

}

// Test the function

console.log(subarrayWithGivenSumBruteForce([1, 2, 3, 7, 5], 12)); // Output: [2, 3, 7]

console.log(subarrayWithGivenSumBruteForce([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 15)); // Output: [1, 2, 3, 4, 5]

console.log(subarrayWithGivenSumBruteForce([1, 2, 3, 4], 10)); // Output: [1, 2, 3, 4]

console.log(subarrayWithGivenSumBruteForce([1, 2, 3, 4], 11)); // Output: []

**Detailed Iteration Steps**

1. **Initialization**:
   * start = 0
   * currentSum = 0
2. **First Outer Loop (start = 0)**:
   * **Inner Loop (end = 0)**:
     + currentSum += 1 (arr[0])
     + currentSum = 1 (not equal to sum)
   * **Inner Loop (end = 1)**:
     + currentSum += 2 (arr[1])
     + currentSum = 3 (not equal to sum)
   * **Inner Loop (end = 2)**:
     + currentSum += 3 (arr[2])
     + currentSum = 6 (not equal to sum)
   * **Inner Loop (end = 3)**:
     + currentSum += 7 (arr[3])
     + currentSum = 13 (greater than sum)
   * **Inner Loop (end = 4)**:
     + currentSum += 5 (arr[4])
     + currentSum = 18 (greater than sum)
3. **Second Outer Loop (start = 1)**:
   * **Reset currentSum**:
     + currentSum = 0
   * **Inner Loop (end = 1)**:
     + currentSum += 2 (arr[1])
     + currentSum = 2 (not equal to sum)
   * **Inner Loop (end = 2)**:
     + currentSum += 3 (arr[2])
     + currentSum = 5 (not equal to sum)
   * **Inner Loop (end = 3)**:
     + currentSum += 7 (arr[3])
     + currentSum = 12 (equal to sum)
     + Return subarray [2, 3, 7]

**Result**

* The function finds the subarray [2, 3, 7] and returns it.

**Complexity Analysis:**

* **Time Complexity**: O(n2)O(n^2)O(n2)
  + Outer loop runs n times.
  + Inner loop runs n - start times for each start, leading to a total of approximately n(n+1)/2n(n + 1)/2n(n+1)/2 iterations.
* **Space Complexity**: O(1)
  + Only a few variables are used, regardless of input size.

The brute force method is straightforward and works well for small input sizes. However, it is inefficient for larger arrays due to its quadratic time complexity.