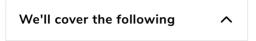




## Solution Review: A Sublist with a Sum of 0

This review provides a detailed analysis of the solution to the A Sublist with a Sum of 0 Challenge.



- Solution: Iterative Hashing
  - Time Complexity

## Solution: Iterative Hashing #

```
def find_sub_zero(my_list):
 2
        # Use hash table to store the cumulative sum as key
 3
        # and the element as value till which sum has been calculated
        # Traverse the list and return true if either
        # elem == 0 or sum == 0 or hash table already contains the sum
 6
        # If you completely traverse the list
 7
        # and haven't found any of the above three
 8
        # conditions then simply return false
 9
        ht = dict()
10
        total_sum = 0
11
        # Traverse through the given list
12
        for elem in my_list:
             total_sum += elem
13
            if elem is 0 or total_sum is 0 or ht.get(total_sum) is not None:
14
15
                 return True
            ht[total sum] = elem
16
17
        return False
18
19
20
   my_list = [6, 4, -7, 3, 12, 9]
21
   print(find_sub_zero(my_list))
22
23
\triangleright
                                                                                              []
```

The naive solution would be to iterate the list in a nested loop, summing each element with all the elements succeeding it.

A hash table makes things much simpler.

We basically have to check for 3 conditions:

- If 0 exists in the list
- If the sum becomes zero in the iteration

• If the sum reverts back to a value which was already a key in the hash table. This means that there was a sublist that has a sum of zero making the overall sum to go back to previous value.

Any of these three conditions confirms the existence of a sublist that sums up to be zero.

## Time Complexity #

As always, a linear iteration over  $\mathbf{n}$  elements means that the algorithm's time complexity is O(n).

