Solution Review: Find Two Pairs Such That a+b = c+d

This review provides a detailed analysis of the solution to the Find Two Pairs Such That a+b = c+d Challenge.

We'll cover the following

Solution: Sums Stored as Hash Keys

```
def find_pair(my_list):
    result = []
    # Create a dictionary my_dict with the key being the sum
    # and the value being a pair, i.e key = 3 , value = {1,2}
    # Traverse all possible pairs in my_list and store sums in my_dict
    # If sum already exists then print out the two pairs.
    my_dict = dict()
    for i in range(len(my_list)):
        for j in range(i+1, len(my_list)):
            added = my_list[i] + my_list[j]  # calculate sum
            # the 'in' operator on dict() item has a complexity of O(1)
            # This is due to hashing
            # On a list, the 'in' operator would have the complexity of O(n)
            if added not in my_dict:
                # If added is not present in dict then insert it with pair
                my_dict[added] = [my_list[i], my_list[j]]
            else:
                # added already present in the dictionay
                prev_pair = my_dict.get(added)
                # Since list elements are distinct, we don't
```

```
21                  # need to check if any element is common among pairs
22                  second_pair = [my_list[i], my_list[j]]
23                  result.append(prev_pair)
24                  result.append(second_pair)
25                  return result
26      return result
27
28
```

Each element in `my_list` is summed with all other elements one by one and the pair is stored. The sum becomes the key in the `my_dict` dictionary. At every key, we store the integer pair whose sum generated that key.

Whenever a sum is found such that its key in the dictionary already has an integer pair stored in it, we can conclude that this sum can be made by two different pairs in the list. These two pairs are then returned in the `result` list.

Time Complexity

The algorithm contains a nested loop. However, the inner loop always starts one step ahead of the outer loop:

```
for i in range(len(my_list)):
  for j in range(i+1,len(my_list)):
```

As the outer loop grows, the inner loop gets smaller. First, the inner loop runs `n - 1` times, then `n - 2`, and so on...

This is an arithmetic series.

After evaluating the series for these values, the time complexity of this algorithm is $O(n^2)$.

Back

Challenge 5: Find Two Pairs in List suc...

Next

Challenge 6: A Sublist with a Sum of 0

Mark as Completed

Report an Issue