

## Solution Review: Find Two Numbers that Add up to "k"

This review provides a detailed analysis of the different ways to solve the find two numbers that add up to k.

We'll cover the following ^

- Solution #1: Brute Force
  - Time Complexity
- Solution #2: Sorting the List
  - Time Complexity
- Solution #3: Moving indices
  - Time Complexity

### Solution #1: Brute Force #

```
1 def find_sum(lst, k):
2     # iterate lst with i
3     for i in range(len(lst)):
4         # iterate lst with j
5         for j in range(len(lst)):
6             # if sum of two iterators is k
7             # and i is not equal to j
8             # then we have our answer
9             if (lst[i]+lst[j] is k and i is not j):
10                return [lst[i], lst[j]]
11
12
13 print(find_sum([1, 2, 3, 4], 5))
```



This is the most time intensive but intuitive solution. Traverse the whole list of size, say  $s$ , for each element in the list and check if any of the two elements add up to the given number  $k$ . So, using two nested for-loops each iterating over the entire list will serve the purpose.

### Time Complexity #

Since we iterate over the entire list of  $k$  elements,  $n$  times in the worst case, therefore, the time complexity is  $O(n^2)$ .

### Solution #2: Sorting the List #

```
1 def binary_search(a, item):
2     first = 0
3     last = len(a) - 1
4     found = False
5     index = -1
6     while first <= last and not found:
7         mid = (first + last) // 2
```



```

8         if a[mid] == item:
9             index = mid
10            found = True
11        else:
12            if item < a[mid]:
13                last = mid - 1
14            else:
15                first = mid + 1
16    if found:
17        return index
18    else:
19        return -1
20
21
22    def find_sum(lst, k):
23        lst.sort()
24        for j in range(len(lst)):
25            # find the difference in list through binary search
26            # return the only if we find an index
27            index = binary_search(lst, k - lst[j])
28            if index is not -1 and index is not j:
29                return [lst[j], k - lst[j]]
30
31
32
33    print(find_sum([1, 5, 3], 2))
34    print(find_sum([1, 2, 3, 4], 5))

```



While solution #1 is very intuitive, it is not very time efficient. A better way to solve this challenge is by first sorting the list. Then for each element in the list, use a binary search to look for the difference between that element and the intended sum. In other words, if the intended sum is  $k$  and the first element of the sorted list is  $a_0$ , then we will do a binary search for  $k - a_0$ . The search is repeated for every  $a_i$  up to  $a_n$  until one is found.” You can implement the `binary_search()` function however you like, recursively or iteratively.

## Time Complexity #

Since most optimal comparison-based sorting functions take  $O(n \log n)$ , let’s assume that the Python `.sort()` function takes the same. Moreover, since binary search takes  $O(\log n)$  time for a finding a single element, therefore a binary search for all  $n$  elements will take  $O(n \log n)$  time.”

## Solution #3: Moving indices #

```

1    def find_sum(lst, k):
2        # sort the list
3        lst.sort()
4        index1 = 0
5        index2 = len(lst) - 1
6        result = []

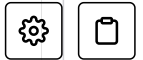
```



```

7     sum = 0
8     # iterate from front and back
9     # move accordingly to reach the sum to be equal to k
10    # returns false when the two indices meet
11    while (index1 != index2):
12        sum = lst[index1] + lst[index2]
13        if sum < k:
14            index1 += 1
15        elif sum > k:
16            index2 -= 1
17        else:
18            result.append(lst[index1])
19            result.append(lst[index2])
20            return result
21    return False
22
23
24 print(find_sum([1, 2, 3, 4], 5))
25 print(find_sum([1, 2, 3, 4], 2))

```



## Time Complexity #

The linear scan takes  $O(n)$  and sort takes  $O(n \log n)$ . The time complexity becomes  $O(n \log n) + O(n)$  because the sort and the linear scan are done one after the other. The overall would be  $O(n \log n)$  in the worst case.

**Note:** The solution provided above is not the optimal solution for this problem. We can write a more efficient solution using hashing. We will cover that approach in Hashing Chapter: Challenge 8 (<https://www.educative.io/courses/data-structures-in-python-an-interview-refresher/B13DmMq4YRJ>)

← Back

Next →

Challenge 3: Find Two Numbers that ...

Challenge 4: List of Products of all Ele...

✓ Completed



Report an  
Issue



Ask a Question

([https://discuss.educative.io/tag/solution-review-find-two-numbers-that-add-up-to-k\\_\\_introduction-to-lists\\_\\_data-structures-for-coding-interviews-in-python](https://discuss.educative.io/tag/solution-review-find-two-numbers-that-add-up-to-k__introduction-to-lists__data-structures-for-coding-interviews-in-python))