Leetcode 15. 2 Sum & 3 Sum

2024.01.28

원루빈

Problem Description - 2 Sum

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Example 1:
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation:
Because nums[0] + nums[1] == 9, we return [0, 1].

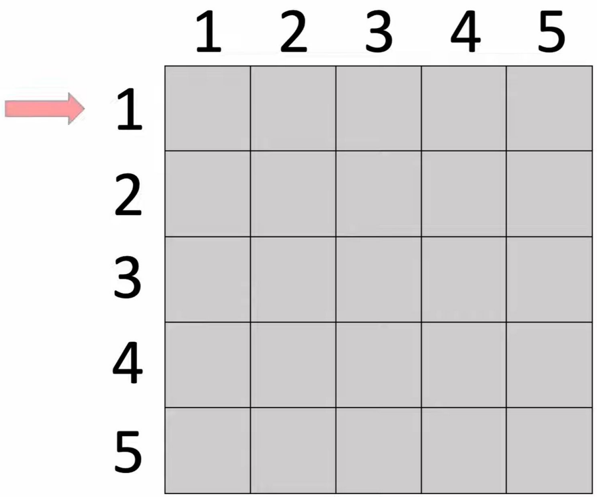
Example 2:
Input: nums = [3,2,4], target = 6
Output: [1,2]

Example 3:
Input: nums = [3,3], target = 6
Output: [0,1]

1 2 Sum (Brute Force)

[Approach 1] Brute Force Solution:

Simply calculate every single combination until you find the right answer.

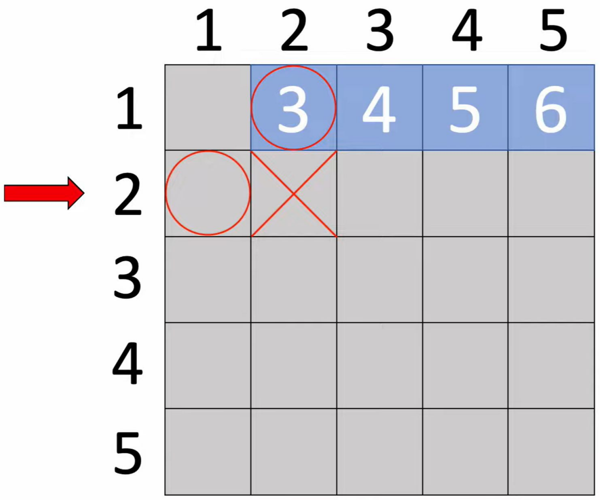


1 2 Sum (Brute Force)

[Approach 1] Brute Force Solution:

Simply calculate every single combination until you find the right answer.

*We cannot use the same number twice.



	1	2	3	4	5
1		3	4	5	6
2					
3					
4					
5					

1 2 Sum (Brute Force)

[Approach 1] Brute Force Solution:

It’s simple and it does work, but this solution runs in O(n^2) time.

Following the same logic….

	1	2	3	4	5
1		3	4	5	6
2			5	6	7
3				7	8
4					9
5					

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$$

Time complexity: O(n^2) times

2 2 Sum (Using Stack & Hash Table)

[Approach 2] Using Stack & Hash Table:

Instead of thinking as “**x + y = target**”,

Since we already know what the target is, think as…

$$\mathbf{y\ = target - x}$$

Time complexity: O(n) times,

since we now only have to traverse the array once.

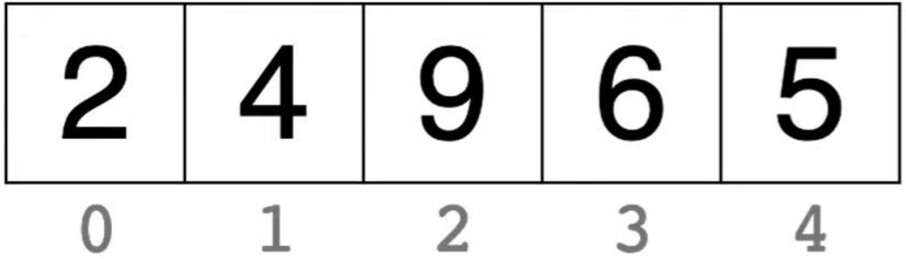
2

2 Sum (Using Stack & Hash Table)

[Approach 2] Using Stack & Hash Table:

Input:

nums = [2,4,9,6, 5]
target = 10



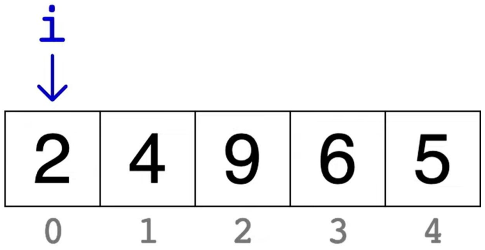
2	4	9	6	5
0	1	2	3	4

2

2 Sum (Using Stack & Hash Table)

Input:

nums = [2,4,9,6, 5]
target = 10



2	4	9	6	5
0	1	2	3	4

diff

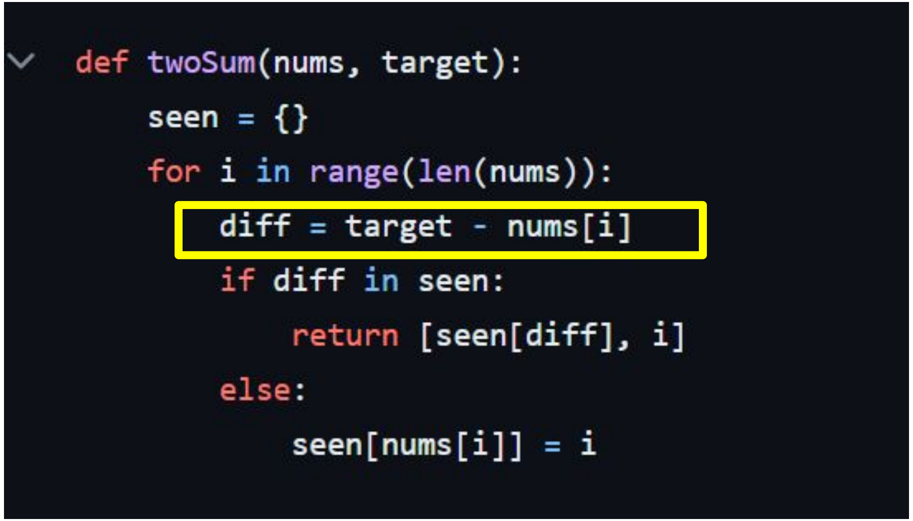
8

seen (num: index)

{

}

Key: num
Value: index



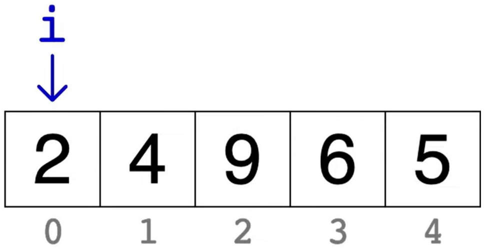
```
def twoSum(nums, target):
    seen = {}
    for i in range(len(nums)):
        diff = target - nums[i]
        if diff in seen:
            return [seen[diff], i]
        else:
            seen[nums[i]] = i
```

2

2 Sum (Using Stack & Hash Table)

Input:

nums = [2,4,9,6, 5]
target = 10



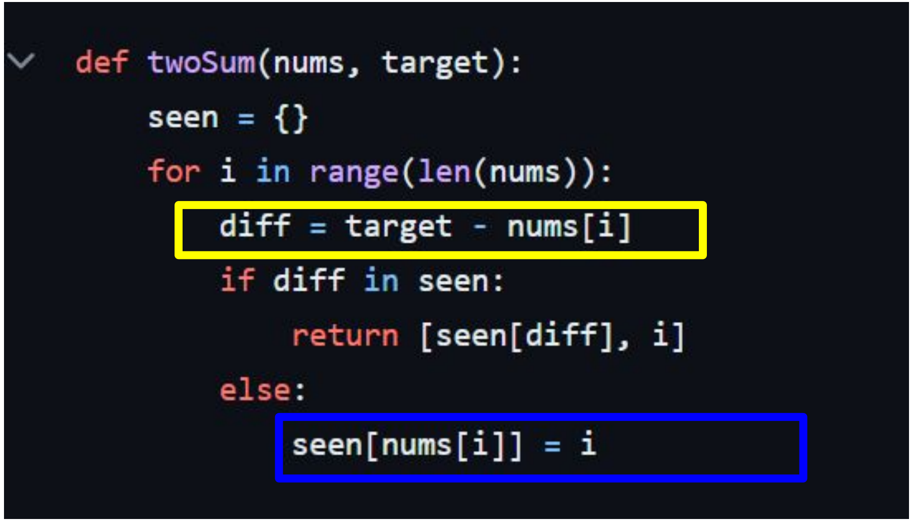
diff

8

seen (num: index)

{
2 : 0
}

Key: num
Value: index

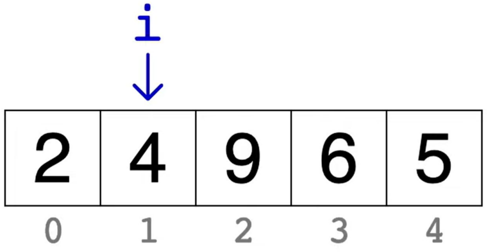


```python
def twoSum(nums, target):
    seen = {}
    for i in range(len(nums)):
        diff = target - nums[i]
        if diff in seen:
            return [seen[diff], i]
        else:
            seen[nums[i]] = i
```

2 | 2 Sum (Using Stack & Hash Table)

Input:

nums = [2,4,9,6, 5]
target = 10



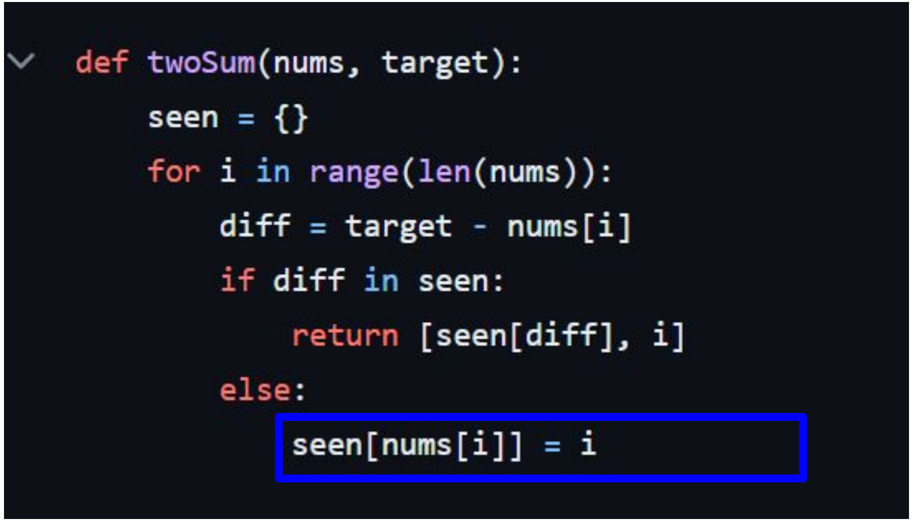
2	4	9	6	5
0	1	2	3	4

diff

6

seen (num: index)

```
{
    2 : 0
    4 : 1
}
```



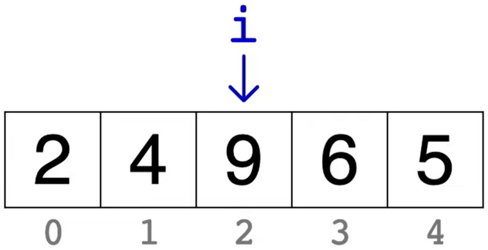
```
def twoSum(nums, target):
    seen = {}
    for i in range(len(nums)):
        diff = target - nums[i]
        if diff in seen:
            return [seen[diff], i]
        else:
            seen[nums[i]] = i
```

Key: num
Value: index

2 2 Sum (Using Stack & Hash Table)

Input:

nums = [2,4,9,6, 5]
target = 10



2	4	9	6	5
0	1	2	3	4

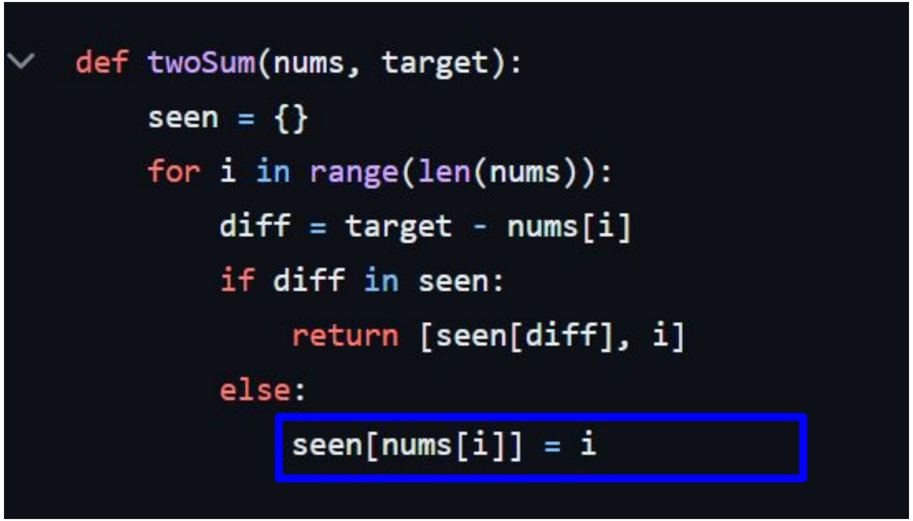
diff

1

seen (num: index)

{
2 : 0
4 : 1
9 : 2
}

Key: num
Value: index



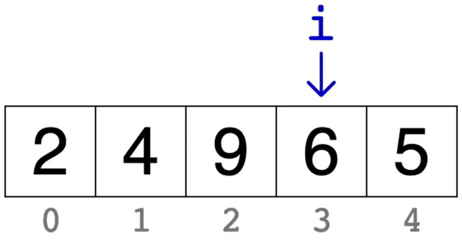
```
def twoSum(nums, target):
    seen = {}
    for i in range(len(nums)):
        diff = target - nums[i]
        if diff in seen:
            return [seen[diff], i]
        else:
            seen[nums[i]] = i
```

2

2 Sum (Using Stack & Hash Table)

Input:

nums = [2,4,9,6, 5]
target = 10



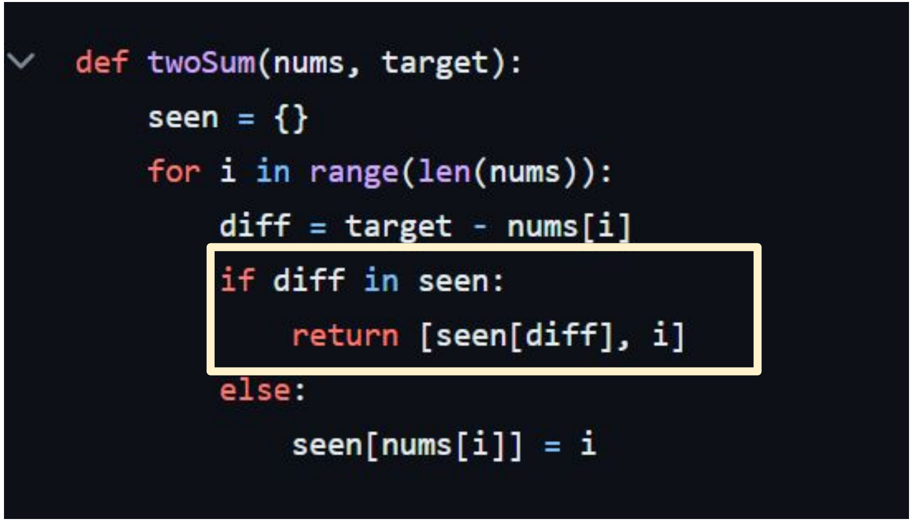
diff

4

seen (num: index)

```
{
    2 : 0
    4 : 1
    9 : 2
}
```

Key: num
Value: index



```
def twoSum(nums, target):
    seen = {}
    for i in range(len(nums)):
        diff = target - nums[i]
        if diff in seen:
            return [seen[diff], i]
        else:
            seen[nums[i]] = i
```

→ return [1, 3]

Problem Description - 3 Sum

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that **i != j, i != k, and j != k,** and **nums[i] + nums[j] + nums[k] == 0.**

Notice that the solution set must not contain duplicate triplets.

Example 1:

Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not matter.

Example 2:

Input: nums = [0,1,1]
Output: []
Explanation:
The only possible triplet does not sum up to 0.

Example 3:

Input: nums = [0,0,0]
Output: [[0,0,0]]
Explanation:
The only possible triplet sums up to 0.

1

3 Sum (Differences compared to 2 Sum)

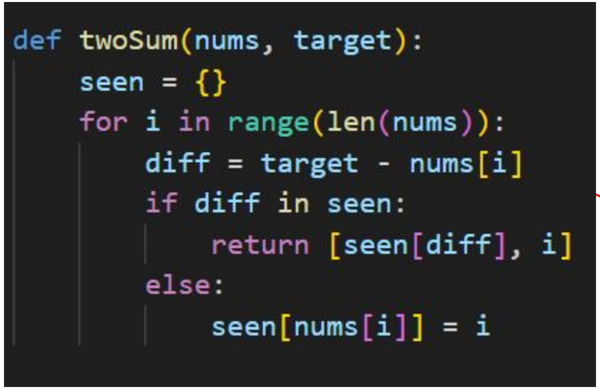
Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that **i != j, i != k, and j != k,** and **nums[i] + nums[j] + nums[k] == 0.**

Notice that the solution set must not contain duplicate triplets.

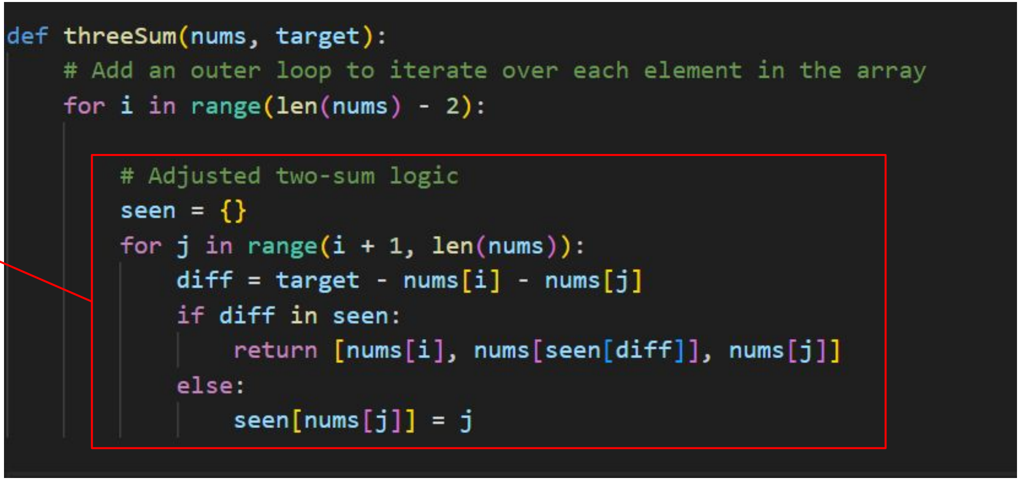
- Brute Force Solution: O(n^3)
- Sum up three values to 0
- Return array values to 0
- There can be multiple solutions, and we must return all of them.

1

3 Sum (Differences compared to 2 Sum)



```python
def twoSum(nums, target):
    seen = {}
    for i in range(len(nums)):
        diff = target - nums[i]
        if diff in seen:
            return [seen[diff], i]
        else:
            seen[nums[i]] = i
```



```python
def threeSum(nums, target):
    # Add an outer loop to iterate over each element in the array
    for i in range(len(nums) - 2):

        # Adjusted two-sum logic
        seen = {}
        for j in range(i + 1, len(nums)):
            diff = target - nums[i] - nums[j]
            if diff in seen:
                return [nums[i], nums[seen[diff]], nums[j]]
            else:
                seen[nums[j]] = j
```

→ Could simply modify twoSum function by adding an outer loop

Problem: It doesn’t account for multiple solution

1

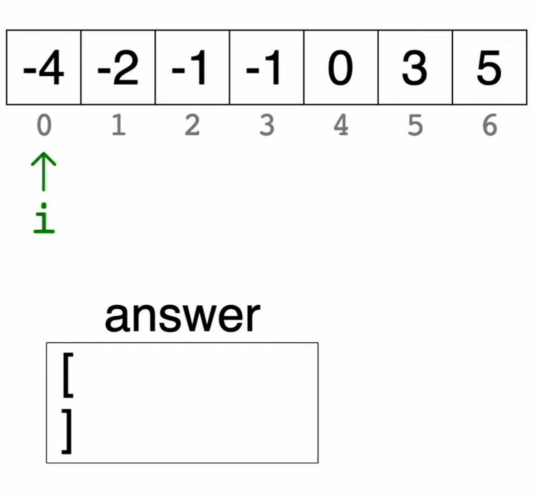
3 Sum (New Approach)

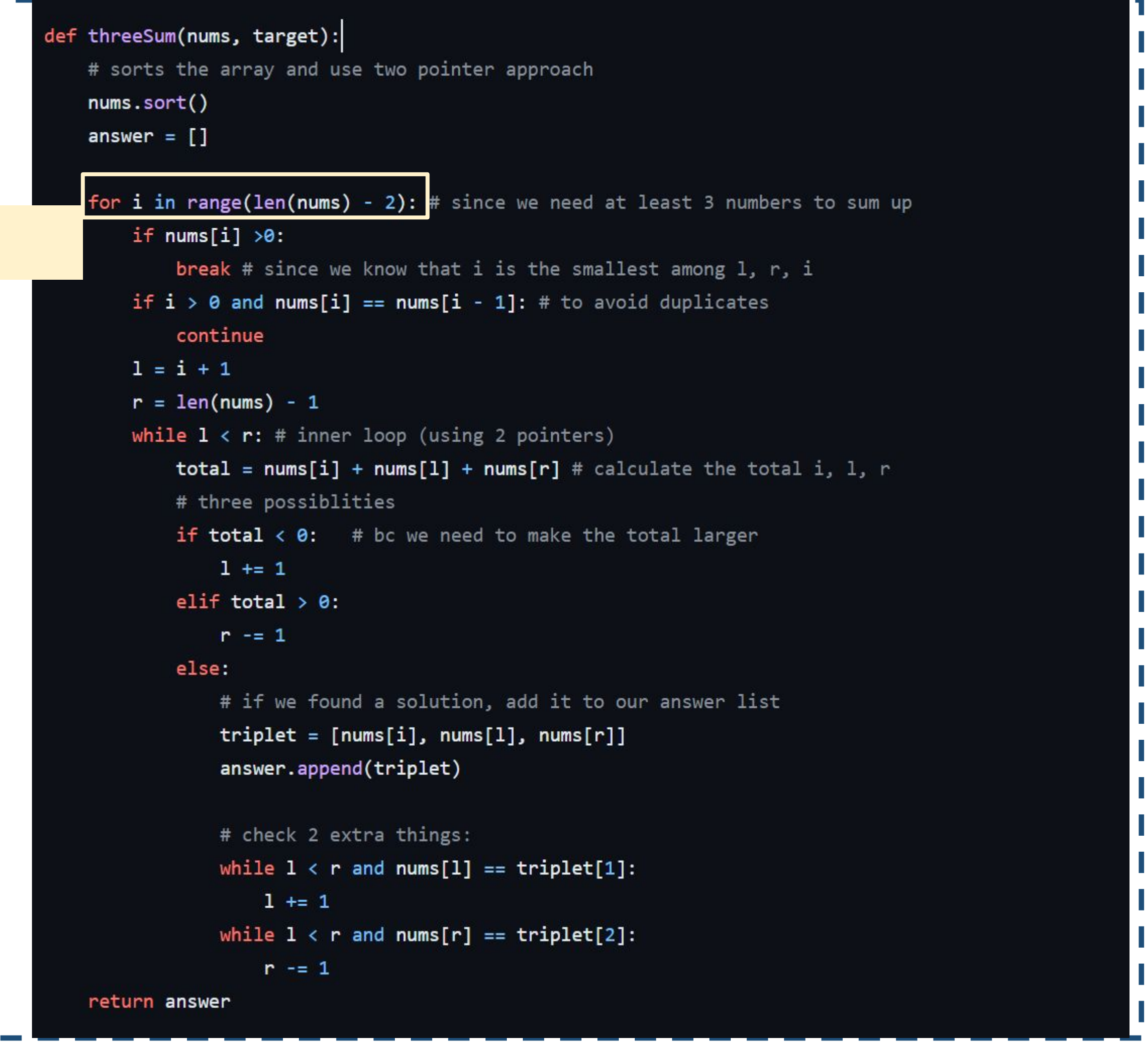
We need to avoid duplicates

→ First **sort** the array and perform **two pointers approach**

1 3 Sum (New Approach)

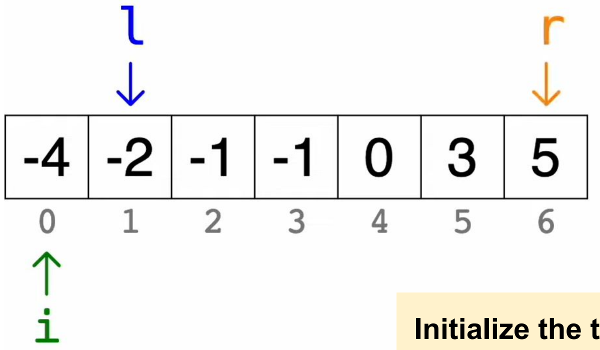
Start iterating i





```python
def threeSum(nums, target):
    # sorts the array and use two pointer approach
    nums.sort()
    answer = []

    for i in range(len(nums) - 2): # since we need at least 3 numbers to sum up
        if nums[i] >0:
            break # since we know that i is the smallest among l, r, i
        if i > 0 and nums[i] == nums[i - 1]: # to avoid duplicates
            continue
        l = i + 1
        r = len(nums) - 1
        while l < r: # inner loop (using 2 pointers)
            total = nums[i] + nums[l] + nums[r] # calculate the total i, l, r
            # three possiblities
            if total < 0:   # bc we need to make the total larger
                l += 1
            elif total > 0:
                r -= 1
            else:
                # if we found a solution, add it to our answer list
                triplet = [nums[i], nums[l], nums[r]]
                answer.append(triplet)

                # check 2 extra things:
                while l < r and nums[l] == triplet[1]:
                    l += 1
                while l < r and nums[r] == triplet[2]:
                    r -= 1
    return answer
```

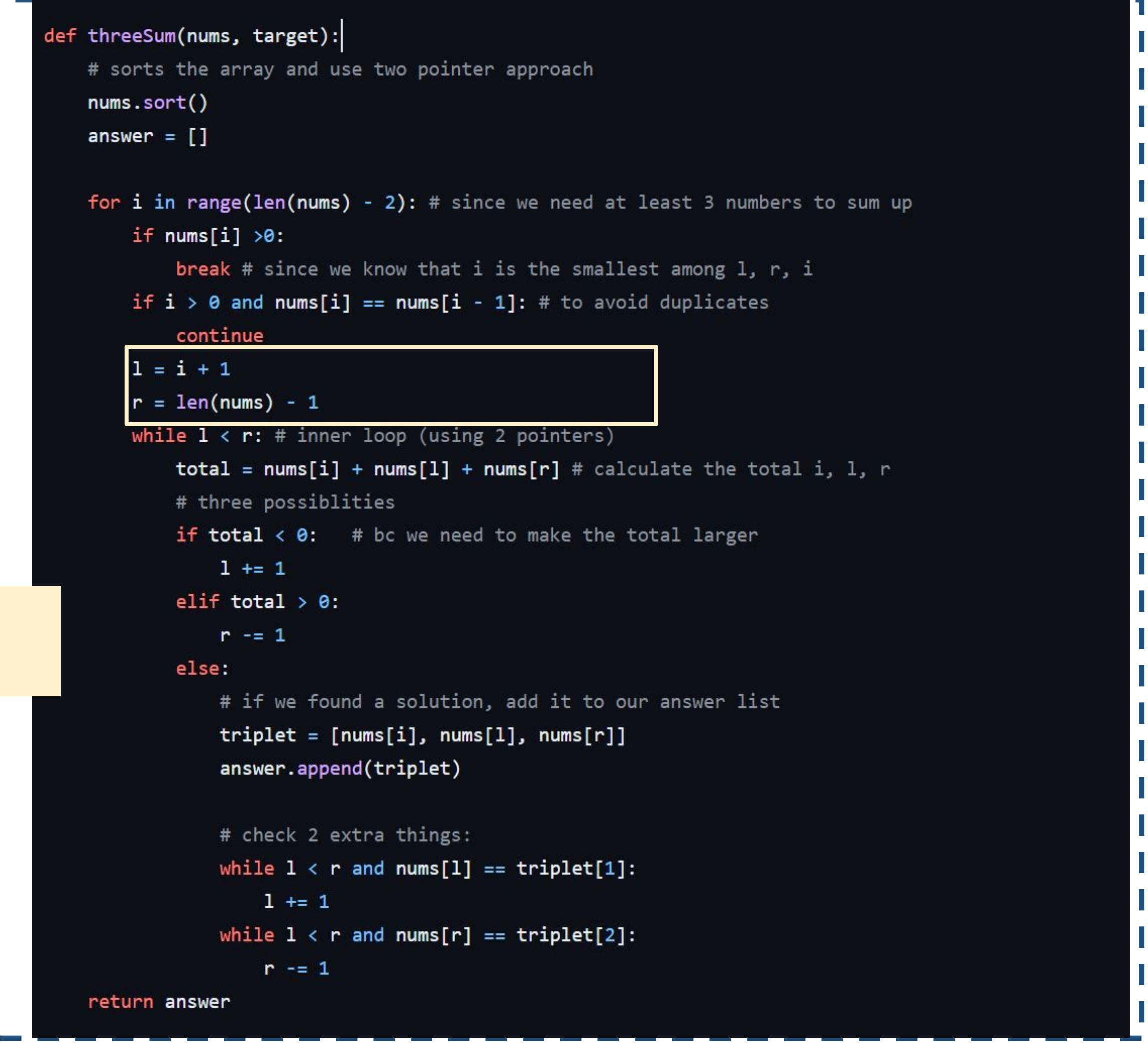
1 3 Sum (New Approach)



Initialize the two pointers

answer

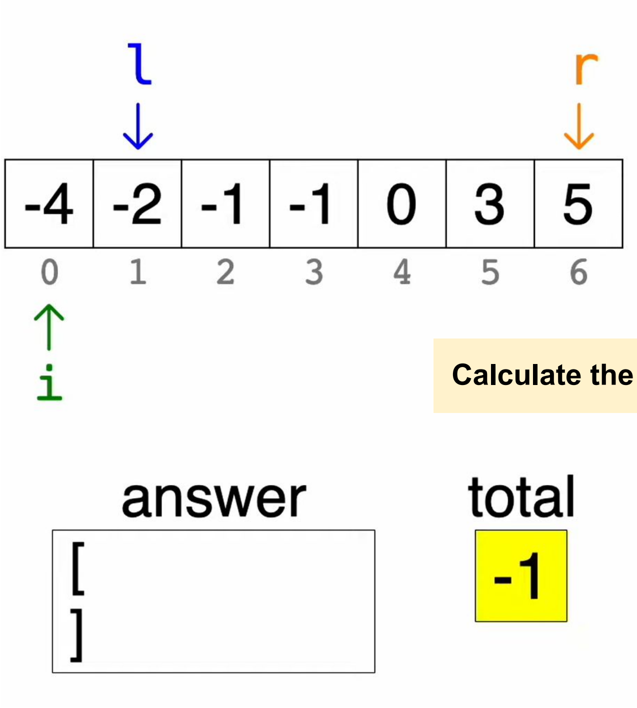
[
]



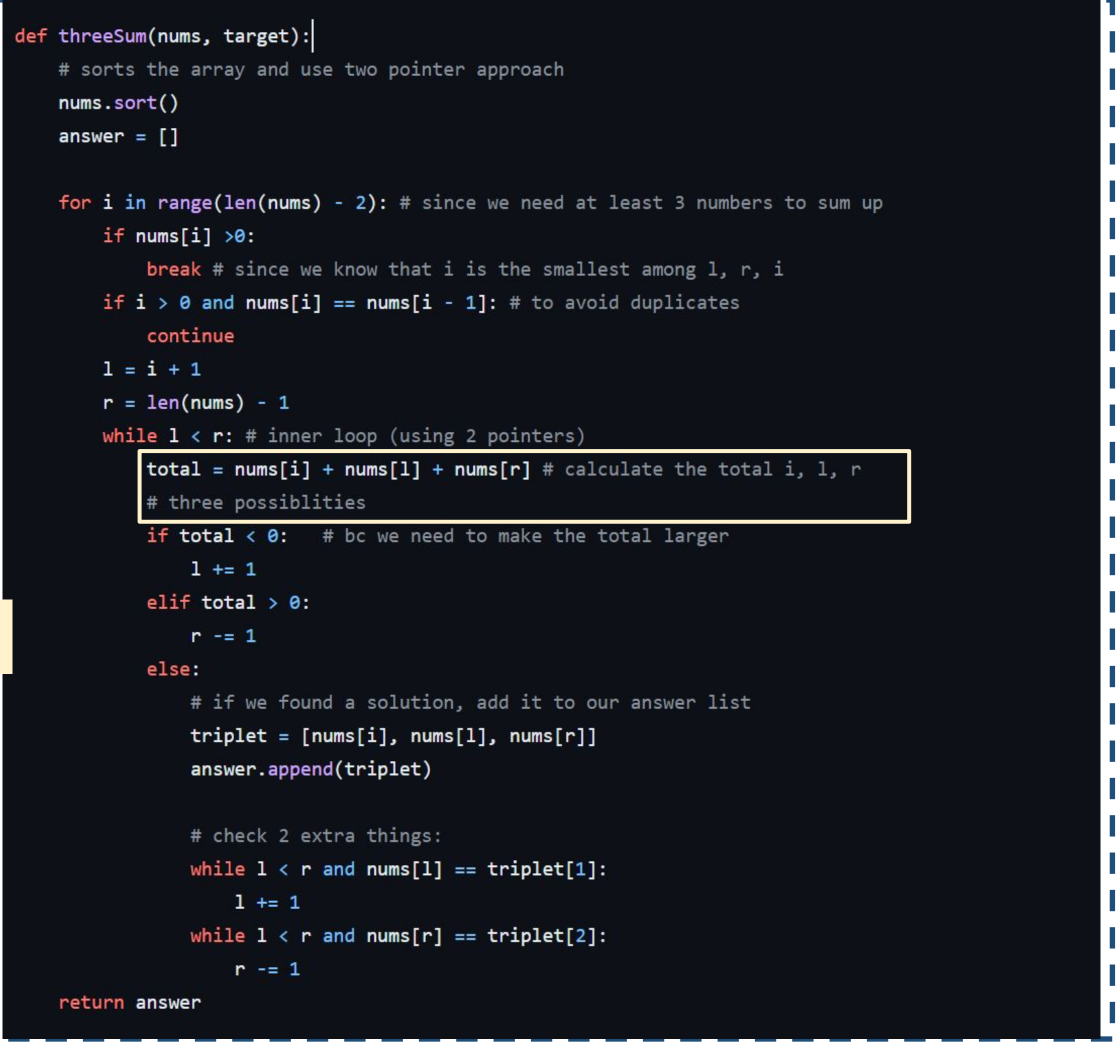
```
def threeSum(nums, target):
    # sorts the array and use two pointer approach
    nums.sort()
    answer = []

    for i in range(len(nums) - 2): # since we need at least 3 numbers to sum up
        if nums[i] >0:
            break # since we know that i is the smallest among l, r, i
        if i > 0 and nums[i] == nums[i - 1]: # to avoid duplicates
            continue
        l = i + 1
        r = len(nums) - 1
        while l < r: # inner loop (using 2 pointers)
            total = nums[i] + nums[l] + nums[r] # calculate the total i, l, r
            # three possiblities
            if total < 0:   # bc we need to make the total larger
                l += 1
            elif total > 0:
                r -= 1
            else:
                # if we found a solution, add it to our answer list
                triplet = [nums[i], nums[l], nums[r]]
                answer.append(triplet)

                # check 2 extra things:
                while l < r and nums[l] == triplet[1]:
                    l += 1
                while l < r and nums[r] == triplet[2]:
                    r -= 1
    return answer
```

1

3 Sum (New Approach)

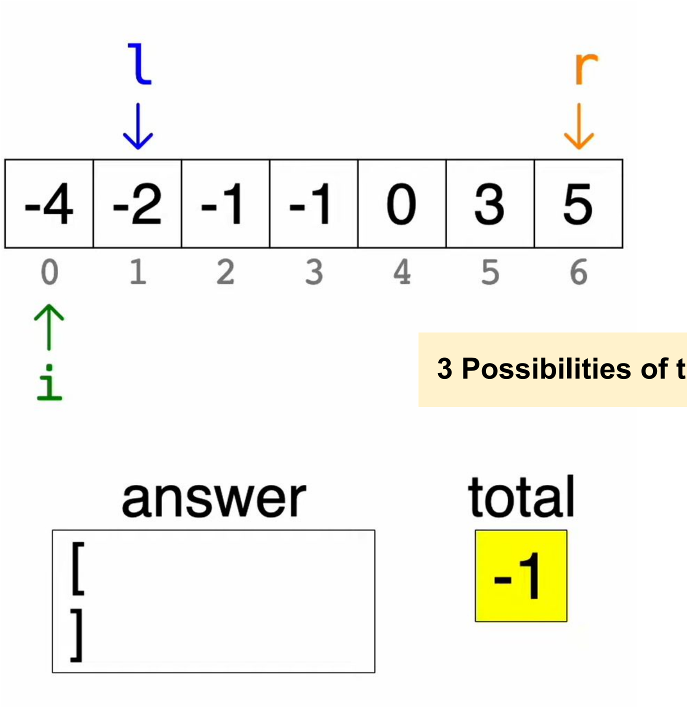


Calculate the total

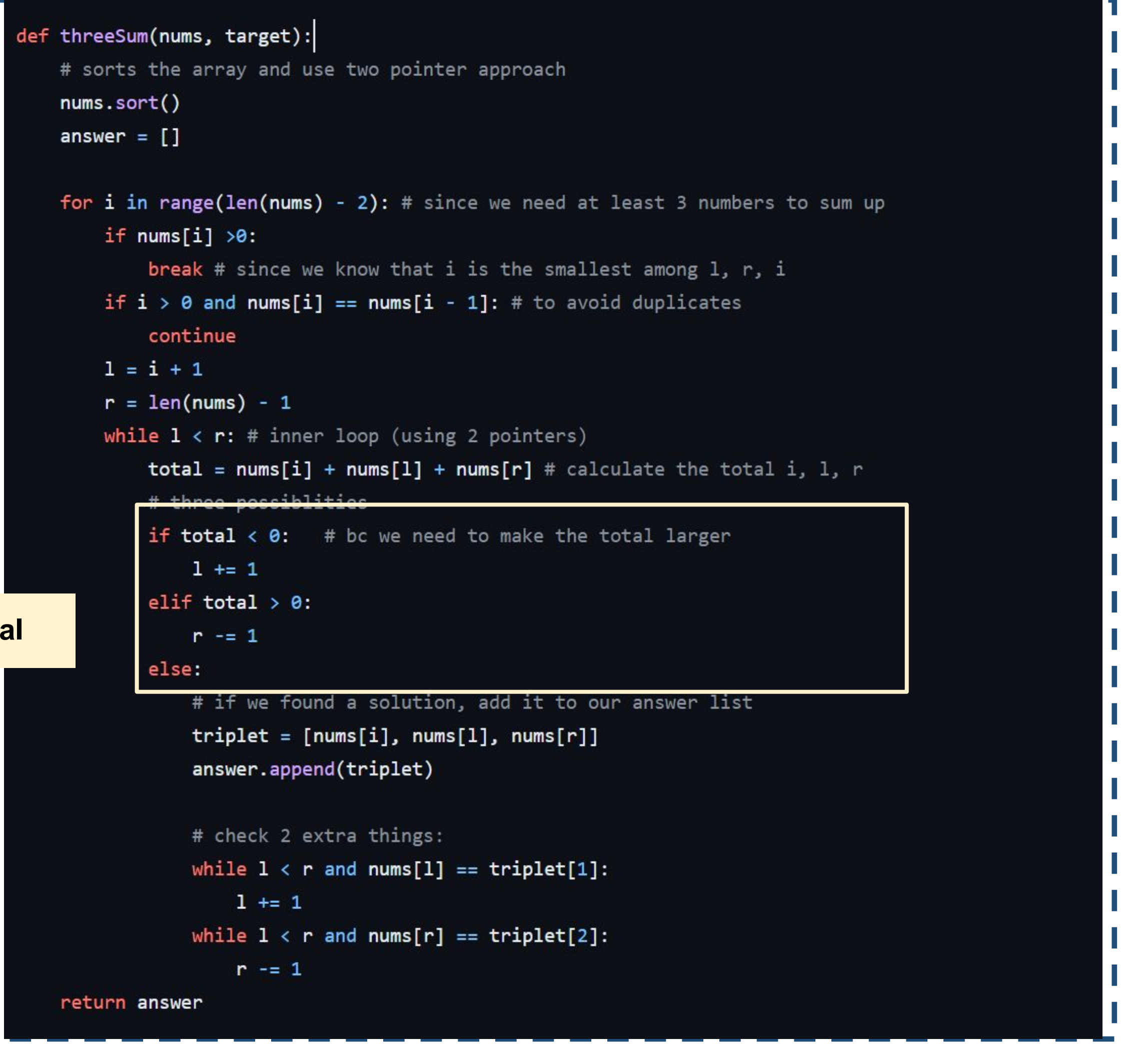


```python
def threeSum(nums, target):
    # sorts the array and use two pointer approach
    nums.sort()
    answer = []

    for i in range(len(nums) - 2): # since we need at least 3 numbers to sum up
        if nums[i] >0:
            break # since we know that i is the smallest among l, r, i
        if i > 0 and nums[i] == nums[i - 1]: # to avoid duplicates
            continue
        l = i + 1
        r = len(nums) - 1
        while l < r: # inner loop (using 2 pointers)
            total = nums[i] + nums[l] + nums[r] # calculate the total i, l, r
            # three possiblities
            if total < 0:   # bc we need to make the total larger
                l += 1
            elif total > 0:
                r -= 1
            else:
                # if we found a solution, add it to our answer list
                triplet = [nums[i], nums[l], nums[r]]
                answer.append(triplet)

                # check 2 extra things:
                while l < r and nums[l] == triplet[1]:
                    l += 1
                while l < r and nums[r] == triplet[2]:
                    r -= 1
    return answer
```

1 3 Sum (New Approach)

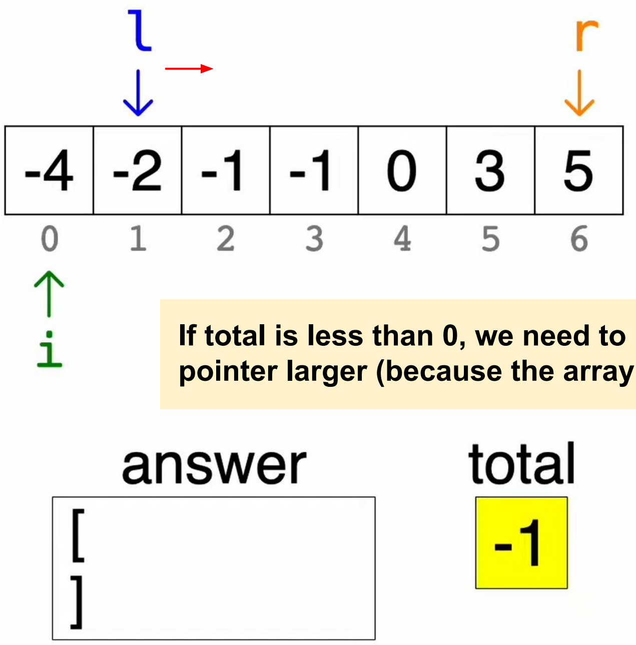


3 Possibilities of total

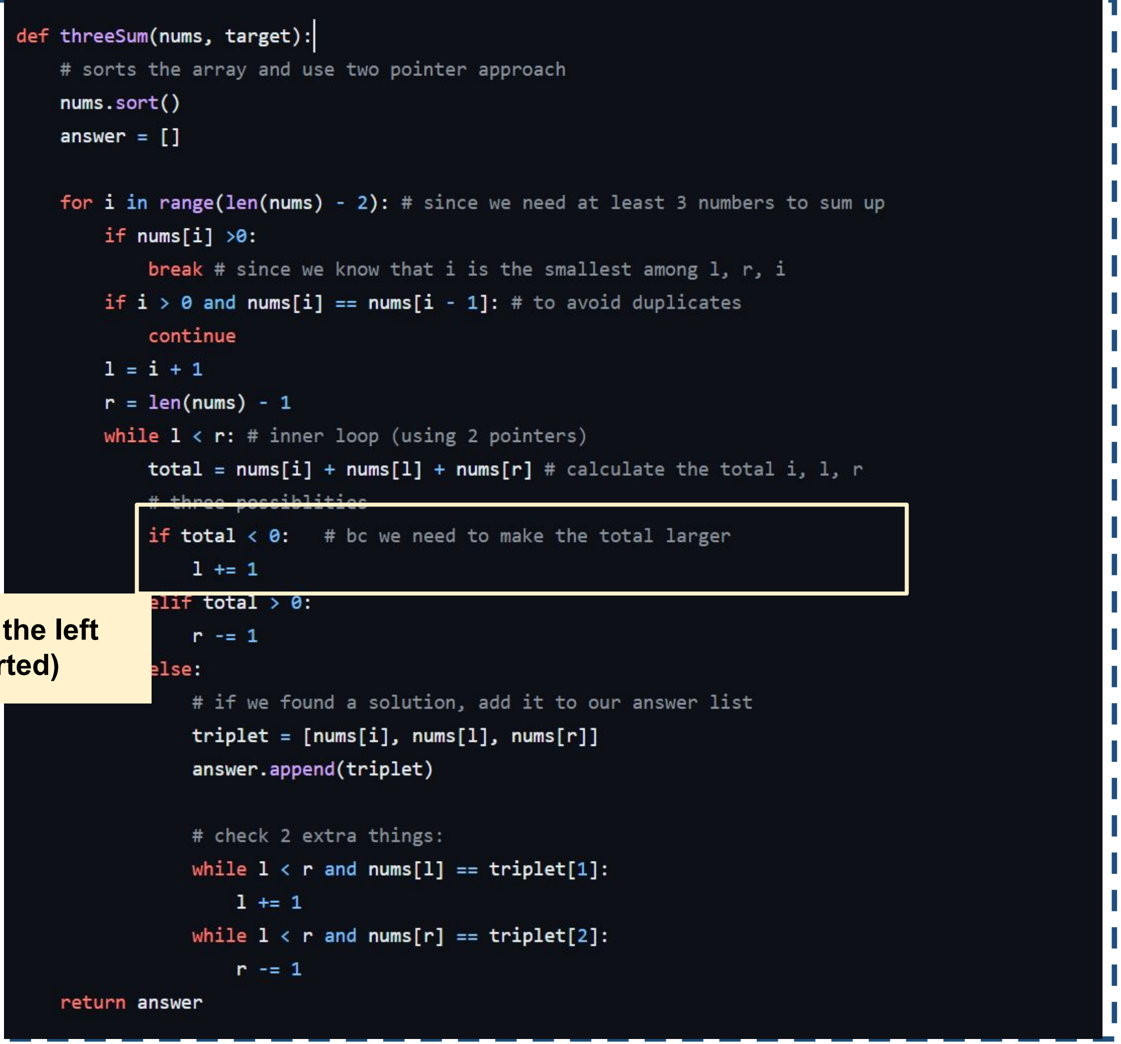


```
def threeSum(nums, target):
    # sorts the array and use two pointer approach
    nums.sort()
    answer = []

    for i in range(len(nums) - 2): # since we need at least 3 numbers to sum up
        if nums[i] >0:
            break # since we know that i is the smallest among l, r, i
        if i > 0 and nums[i] == nums[i - 1]: # to avoid duplicates
            continue
        l = i + 1
        r = len(nums) - 1
        while l < r: # inner loop (using 2 pointers)
            total = nums[i] + nums[l] + nums[r] # calculate the total i, l, r
            # three possiblities
            if total < 0:   # bc we need to make the total larger
                l += 1
            elif total > 0:
                r -= 1
            else:
                # if we found a solution, add it to our answer list
                triplet = [nums[i], nums[l], nums[r]]
                answer.append(triplet)

                # check 2 extra things:
                while l < r and nums[l] == triplet[1]:
                    l += 1
                while l < r and nums[r] == triplet[2]:
                    r -= 1
    return answer
```

1 3 Sum (New Approach)



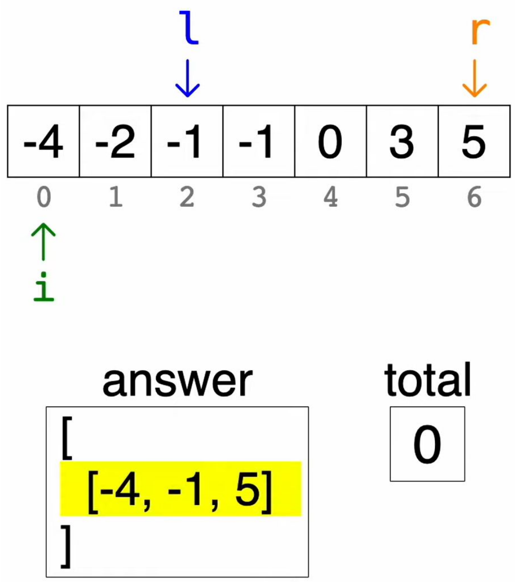
If total is less than 0, we need to make the left pointer larger (because the array is sorted)

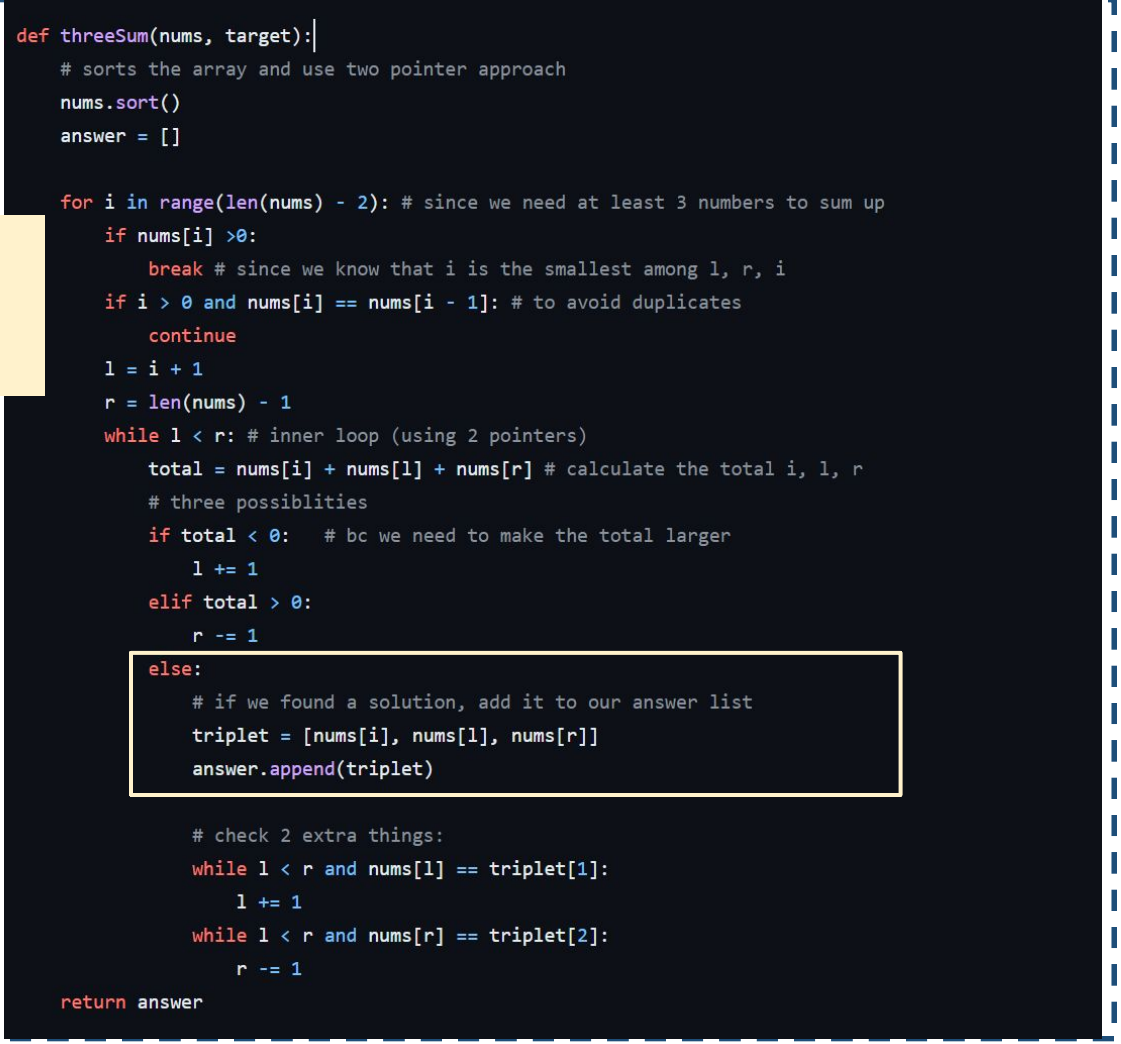


```
def threeSum(nums, target):
    # sorts the array and use two pointer approach
    nums.sort()
    answer = []

    for i in range(len(nums) - 2): # since we need at least 3 numbers to sum up
        if nums[i] >0:
            break # since we know that i is the smallest among l, r, i
        if i > 0 and nums[i] == nums[i - 1]: # to avoid duplicates
            continue
        l = i + 1
        r = len(nums) - 1
        while l < r: # inner loop (using 2 pointers)
            total = nums[i] + nums[l] + nums[r] # calculate the total i, l, r
            # three possiblities
            if total < 0:   # bc we need to make the total larger
                l += 1
            elif total > 0:
                r -= 1
            else:
                # if we found a solution, add it to our answer list
                triplet = [nums[i], nums[l], nums[r]]
                answer.append(triplet)

                # check 2 extra things:
                while l < r and nums[l] == triplet[1]:
                    l += 1
                while l < r and nums[r] == triplet[2]:
                    r -= 1
    return answer
```

1 3 Sum (New Approach)

After moving the left pointer, Calculate the total again, And since total == 0, we append it to our answer stack

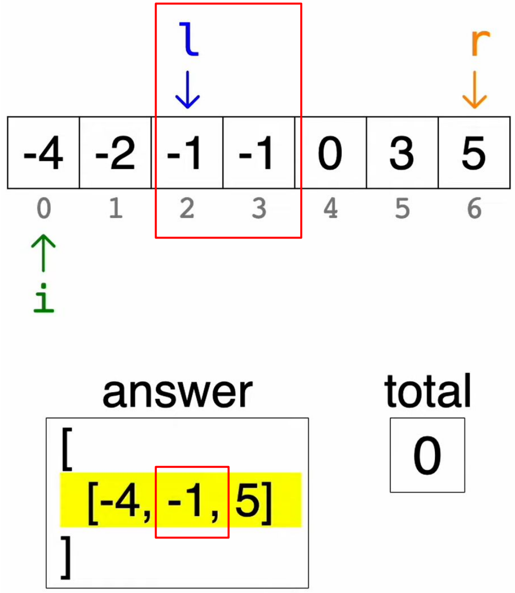


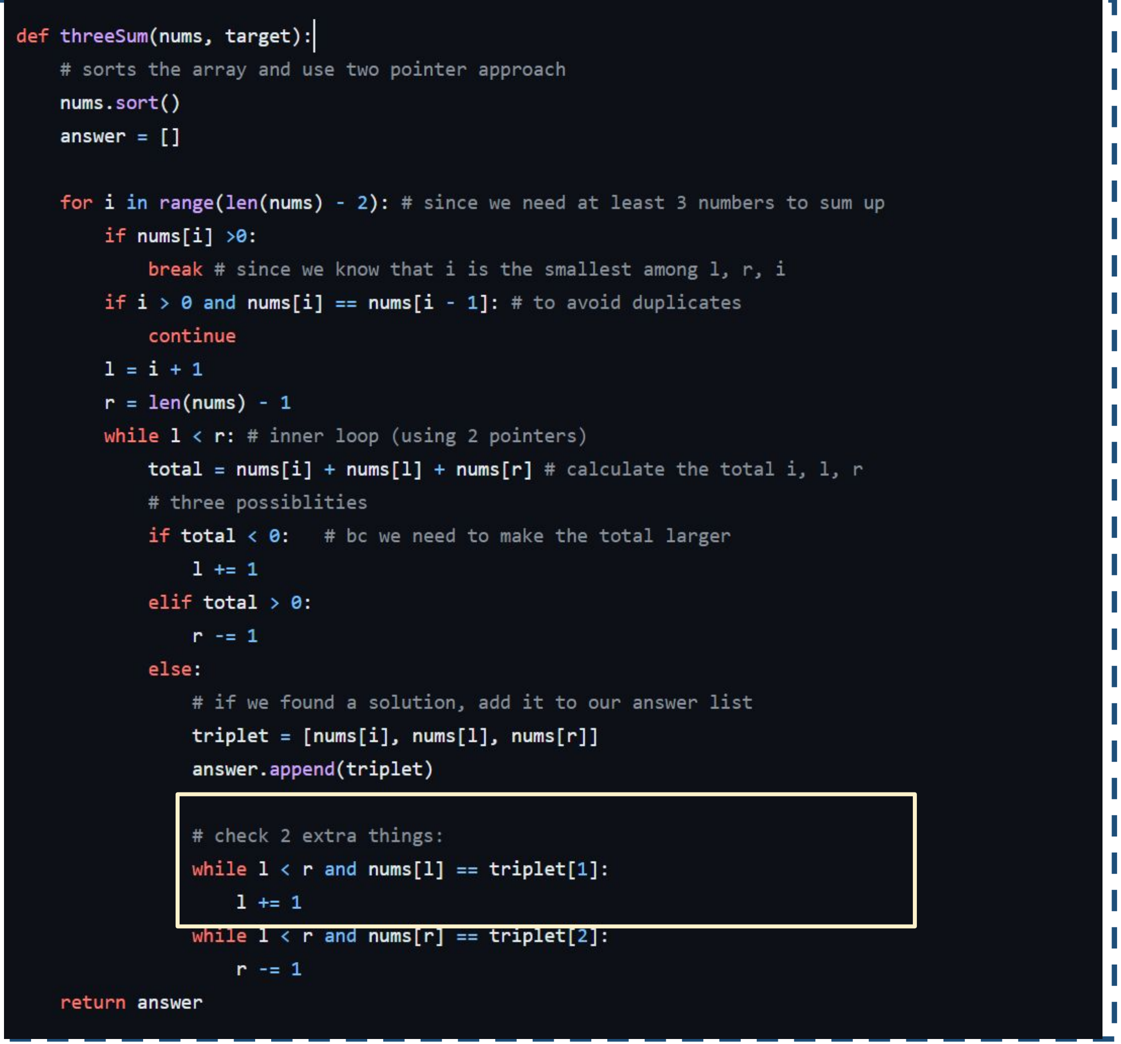


```
def threeSum(nums, target):
    # sorts the array and use two pointer approach
    nums.sort()
    answer = []

    for i in range(len(nums) - 2): # since we need at least 3 numbers to sum up
        if nums[i] >0:
            break # since we know that i is the smallest among l, r, i
        if i > 0 and nums[i] == nums[i - 1]: # to avoid duplicates
            continue
        l = i + 1
        r = len(nums) - 1
        while l < r: # inner loop (using 2 pointers)
            total = nums[i] + nums[l] + nums[r] # calculate the total i, l, r
            # three possiblities
            if total < 0:   # bc we need to make the total larger
                l += 1
            elif total > 0:
                r -= 1
            else:
                # if we found a solution, add it to our answer list
                triplet = [nums[i], nums[l], nums[r]]
                answer.append(triplet)

                # check 2 extra things:
                while l < r and nums[l] == triplet[1]:
                    l += 1
                while l < r and nums[r] == triplet[2]:
                    r -= 1
    return answer
```

1 3 Sum (New Approach)

Check for duplicate solution.
→ increment l pointer until it points to the different number

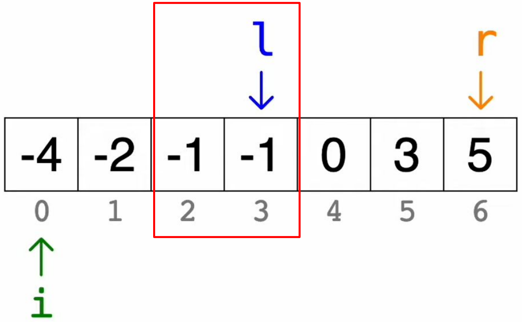




```python
def threeSum(nums, target):
    # sorts the array and use two pointer approach
    nums.sort()
    answer = []

    for i in range(len(nums) - 2): # since we need at least 3 numbers to sum up
        if nums[i] >0:
            break # since we know that i is the smallest among l, r, i
        if i > 0 and nums[i] == nums[i - 1]: # to avoid duplicates
            continue
        l = i + 1
        r = len(nums) - 1
        while l < r: # inner loop (using 2 pointers)
            total = nums[i] + nums[l] + nums[r] # calculate the total i, l, r
            # three possiblities
            if total < 0:   # bc we need to make the total larger
                l += 1
            elif total > 0:
                r -= 1
            else:
                # if we found a solution, add it to our answer list
                triplet = [nums[i], nums[l], nums[r]]
                answer.append(triplet)

                # check 2 extra things:
                while l < r and nums[l] == triplet[1]:
                    l += 1
                while l < r and nums[r] == triplet[2]:
                    r -= 1
    return answer
```

1 3 Sum (New Approach)

Check for duplicate solution.
→ increment l pointer until it points to the different number

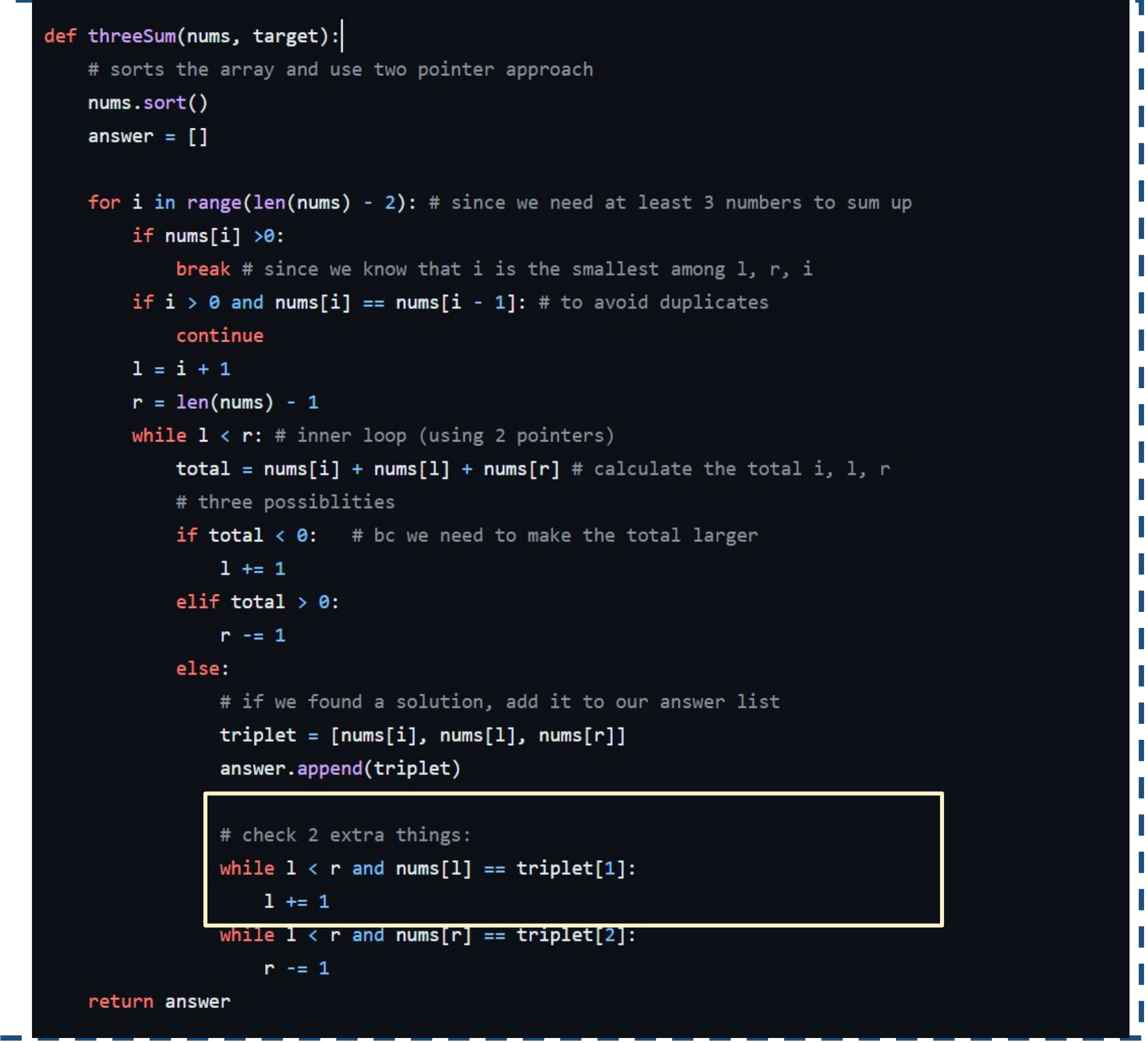


answer

```
[
  [-4, -1, 5]
]
```

total

0

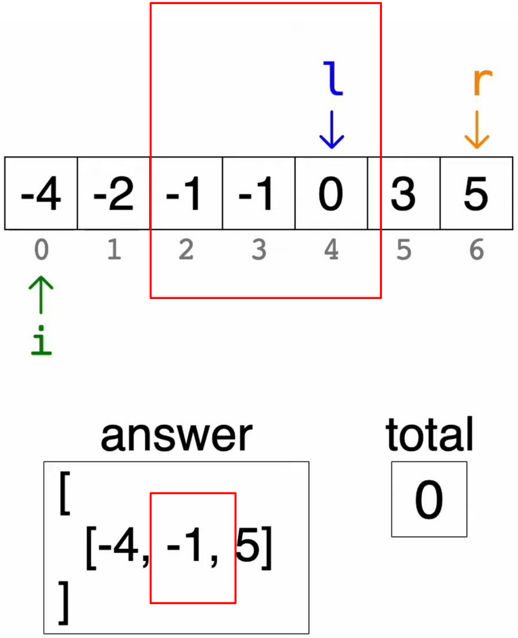


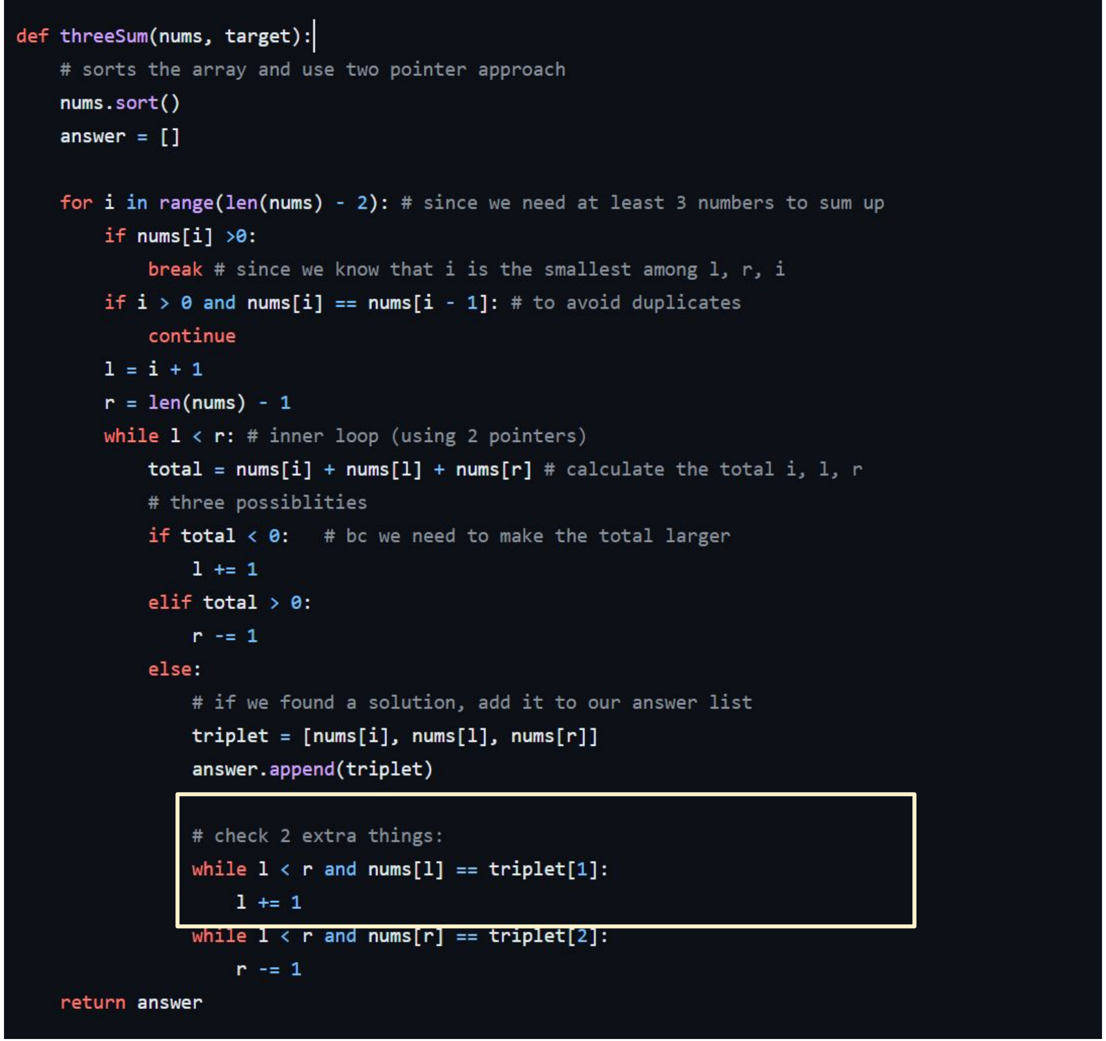
```
def threeSum(nums, target):
    # sorts the array and use two pointer approach
    nums.sort()
    answer = []

    for i in range(len(nums) - 2): # since we need at least 3 numbers to sum up
        if nums[i] >0:
            break # since we know that i is the smallest among l, r, i
        if i > 0 and nums[i] == nums[i - 1]: # to avoid duplicates
            continue
        l = i + 1
        r = len(nums) - 1
        while l < r: # inner loop (using 2 pointers)
            total = nums[i] + nums[l] + nums[r] # calculate the total i, l, r
            # three possiblities
            if total < 0:   # bc we need to make the total larger
                l += 1
            elif total > 0:
                r -= 1
            else:
                # if we found a solution, add it to our answer list
                triplet = [nums[i], nums[l], nums[r]]
                answer.append(triplet)

                # check 2 extra things:
                while l < r and nums[l] == triplet[1]:
                    l += 1
                while l < r and nums[r] == triplet[2]:
                    r -= 1
    return answer
```

1

3 Sum (New Approach)

Finally l is pointing to the different number! Now, check for the r pointer.



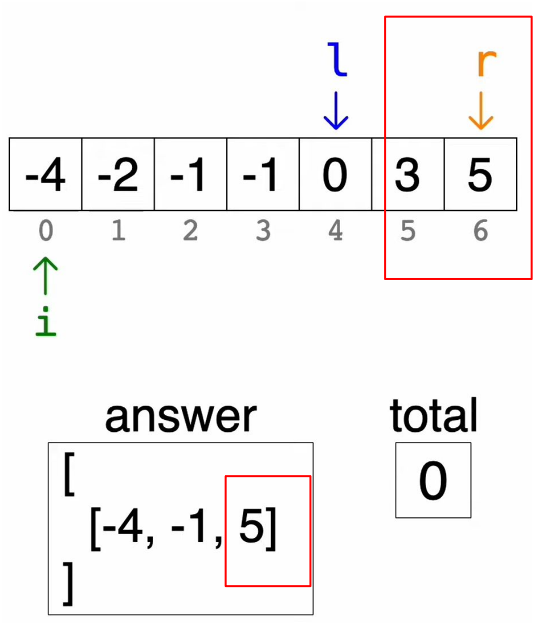


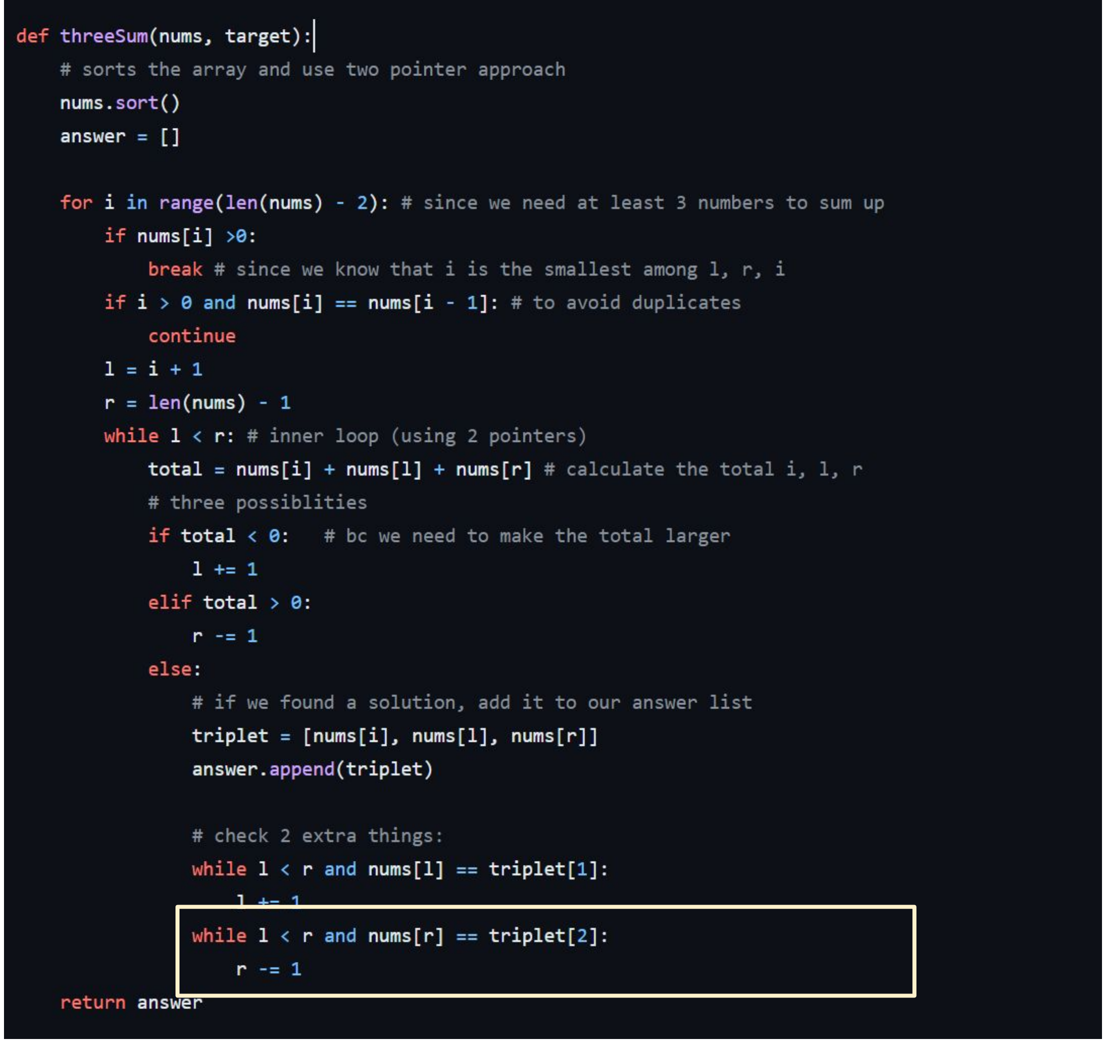
```
def threeSum(nums, target):
    # sorts the array and use two pointer approach
    nums.sort()
    answer = []

    for i in range(len(nums) - 2): # since we need at least 3 numbers to sum up
        if nums[i] >0:
            break # since we know that i is the smallest among l, r, i
        if i > 0 and nums[i] == nums[i - 1]: # to avoid duplicates
            continue
        l = i + 1
        r = len(nums) - 1
        while l < r: # inner loop (using 2 pointers)
            total = nums[i] + nums[l] + nums[r] # calculate the total i, l, r
            # three possiblities
            if total < 0:   # bc we need to make the total larger
                l += 1
            elif total > 0:
                r -= 1
            else:
                # if we found a solution, add it to our answer list
                triplet = [nums[i], nums[l], nums[r]]
                answer.append(triplet)

                # check 2 extra things:
                while l < r and nums[l] == triplet[1]:
                    l += 1
                while l < r and nums[r] == triplet[2]:
                    r -= 1
    return answer
```

1

3 Sum (New Approach)

Check if r is pointing to the different number

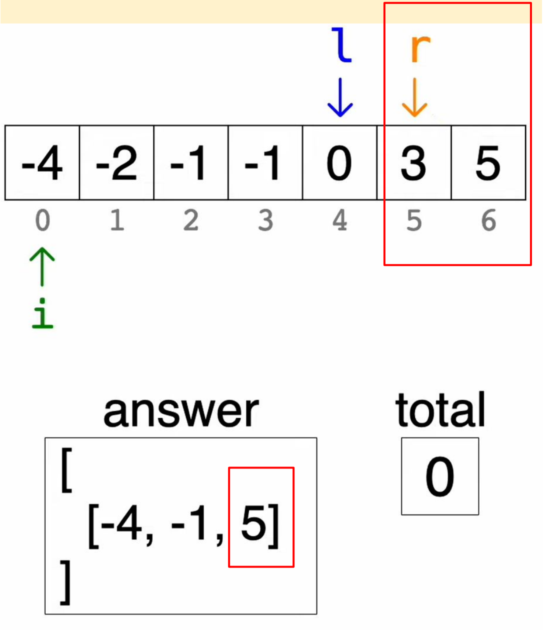


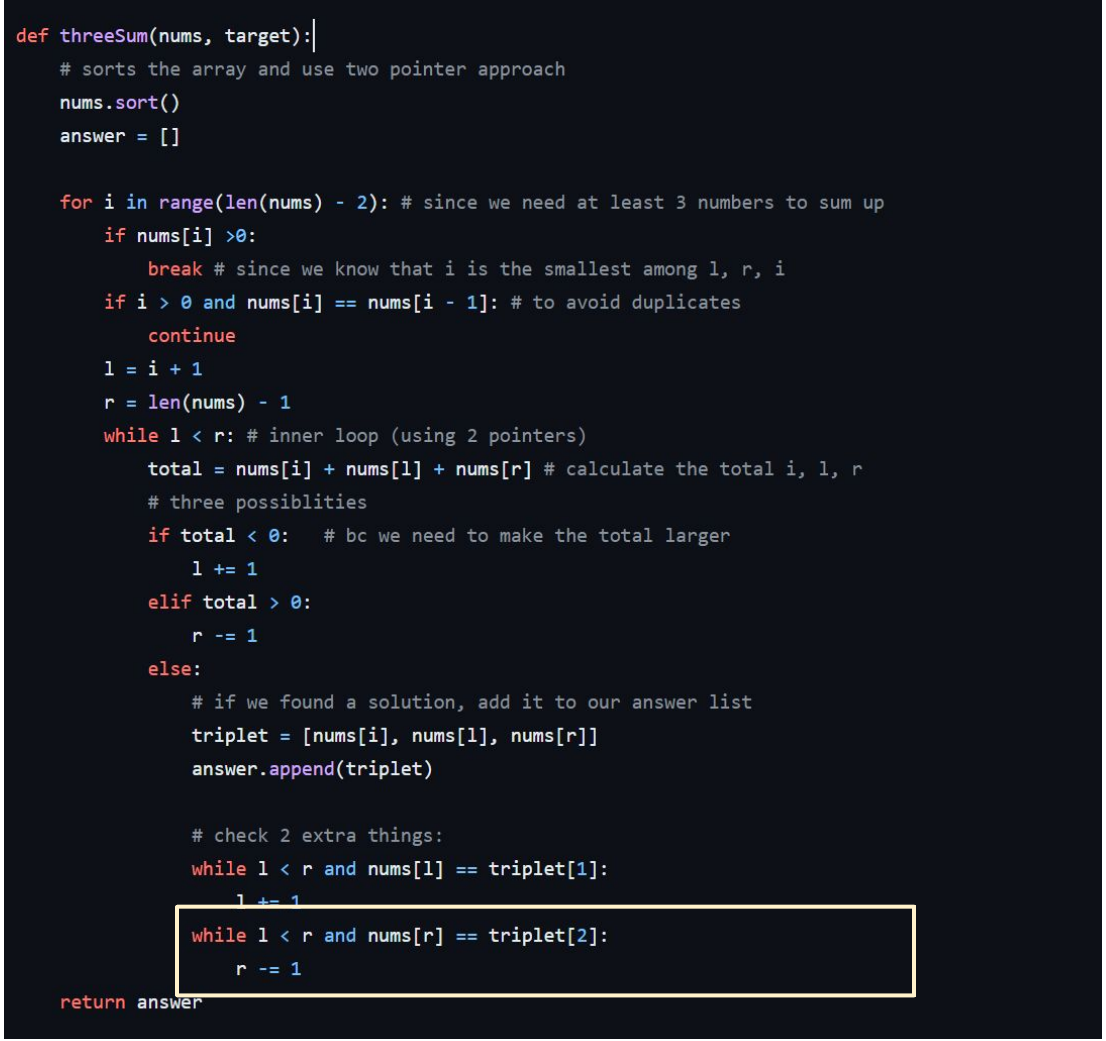


```
def threeSum(nums, target):
    # sorts the array and use two pointer approach
    nums.sort()
    answer = []

    for i in range(len(nums) - 2): # since we need at least 3 numbers to sum up
        if nums[i] >0:
            break # since we know that i is the smallest among l, r, i
        if i > 0 and nums[i] == nums[i - 1]: # to avoid duplicates
            continue
        l = i + 1
        r = len(nums) - 1
        while l < r: # inner loop (using 2 pointers)
            total = nums[i] + nums[l] + nums[r] # calculate the total i, l, r
            # three possiblities
            if total < 0:   # bc we need to make the total larger
                l += 1
            elif total > 0:
                r -= 1
            else:
                # if we found a solution, add it to our answer list
                triplet = [nums[i], nums[l], nums[r]]
                answer.append(triplet)

                # check 2 extra things:
                while l < r and nums[l] == triplet[1]:
                    l += 1
                while l < r and nums[r] == triplet[2]:
                    r -= 1
    return answer
```

1 3 Sum (New Approach)

**Decrement r until it points to the different number.
After that, we can finally calculate the new total.**





```
def threeSum(nums, target):
    # sorts the array and use two pointer approach
    nums.sort()
    answer = []

    for i in range(len(nums) - 2): # since we need at least 3 numbers to sum up
        if nums[i] >0:
            break # since we know that i is the smallest among l, r, i
        if i > 0 and nums[i] == nums[i - 1]: # to avoid duplicates
            continue
        l = i + 1
        r = len(nums) - 1
        while l < r: # inner loop (using 2 pointers)
            total = nums[i] + nums[l] + nums[r] # calculate the total i, l, r
            # three possiblities
            if total < 0:   # bc we need to make the total larger
                l += 1
            elif total > 0:
                r -= 1
            else:
                # if we found a solution, add it to our answer list
                triplet = [nums[i], nums[l], nums[r]]
                answer.append(triplet)

                # check 2 extra things:
                while l < r and nums[l] == triplet[1]:
                    l += 1
                while l < r and nums[r] == triplet[2]:
                    r -= 1
    return answer
```

감사합니다!

THANK YOU