

1. Two Sum(1)

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]`

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        dic = {}
        for i, n in enumerate(nums):
            diff = target - n
            if diff in dic:
                return [i, dic[diff]]
            dic[n] = i
```

167. Two Sum II - Input Array Is Sorted(2)

Given a **1-indexed** array of integers numbers that is already **sorted in non-decreasing order**, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index1] and numbers[index2] where $1 \leq \text{index1} < \text{index2} < \text{numbers.length}$.

Return *the indices of the two numbers*, index1 *and* index2, **added by one as an integer array** [index1, index2] *of length 2*.

The tests are generated such that there is **exactly one solution**. You **may not** use the same element twice.

Your solution must use only constant extra space.

Example 1:

Input: numbers = [2,7,11,15], target = 9

Output: [1,2]

Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2].

Example 2:

Input: numbers = [2,3,4], target = 6

Output: [1,3]

Explanation: The sum of 2 and 4 is 6. Therefore index1 = 1, index2 = 3. We return [1, 3].

Example 3:

Input: numbers = [-1,0], target = -1

Output: [1,2]

Explanation: The sum of -1 and 0 is -1. Therefore index1 = 1, index2 = 2. We return [1, 2].

```

class Solution:
    def twoSum(self, numbers: List[int], target: int) ->
List[int]:
    l = 0
    r = len(numbers)-1
    while l<r:
        if numbers[l]+numbers[r]==target:
            return [l+1, r+1]
        elif numbers[l]+numbers[r]>target:
            r = r-1
        else:
            l = l+1

```

1099. Two Sum Less Than K (3)

Given an array nums of integers and integer k, return the maximum sum such that there exists $i < j$ with $\text{nums}[i] + \text{nums}[j] = \text{sum}$ and $\text{sum} < k$. If no i, j exist satisfying this equation, return -1.

Example 1:

Input: nums = [34,23,1,24,75,33,54,8], k = 60

Output: 58

Explanation: We can use 34 and 24 to sum 58 which is less than 60.

Example 2:

Input: nums = [10,20,30], k = 15

Output: -1

Explanation: In this case it is not possible to get a pair sum less than 15.

```

class Solution:
    def twoSumLessThanK(self, nums: List[int], k: int) -> int:
        nums.sort()
        l = 0
        r = len(nums)-1
        res = -1
        while l<r:
            total = nums[l]+nums[r]
            if total<k:
                res = max(res,total)
                l = l+1
            else:
                r = r-1
        return res

```

15. 3Sum(4)

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that $i \neq j$, $i \neq k$, and $j \neq k$, 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.

[-4,-3,-3,-2,-2,-2,-1,0,2,4,4,4,6,6,6]

```
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        nums.sort()
        res = []
        for i in range(len(nums)-2):
            if i>0 and nums[i]==nums[i-1]:
                continue
            l = i+1
            r = len(nums)-1
            while l<r:
                total = nums[i]+nums[l]+nums[r]
                if total<0:
                    l = l+1
                elif total>0:
                    r = r-1
                else:
                    res.append([nums[i],nums[l], nums[r]])
                    while l<r and nums[l]==nums[l+1]:
                        l = l+1
                    while l<r and nums[r] == nums[r-1]:
                        r = r-1
                    l = l+1
                    r = r-1
            return res
```

16. 3Sum Closest(5)

Given an integer array `nums` of length `n` and an integer `target`, find three integers in `nums` such that the sum is closest to `target`. Return *the sum of the three integers*.

You may assume that each input would have exactly one solution.

Example 1:

Input: `nums = [-1,2,1,-4]`, `target = 1`

Output: 2

Explanation: The sum that is closest to the target is 2. $(-1 + 2 + 1 = 2)$.

Example 2:

Input: `nums = [0,0,0]`, `target = 1`

Output: 0

Explanation: The sum that is closest to the target is 0. $(0 + 0 + 0 = 0)$.

```
class Solution:
    def threeSumClosest(self, nums: List[int], target: int) ->
int:
    nums.sort()
    closestSum = float('inf')
    for i in range(len(nums)-2):
        l = i+1
        r = len(nums)-1
        while l<r:
            total = nums[i]+nums[l]+nums[r]
            if abs(total-target)<abs(closestSum-target):
                closestSum = total
            if total<target:
                l = l+1
            else:
                r = r-1
    return closestSum
```