```
Q1.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:**
Input: nums = [2,7,11,15], target = 9
Output0 [0,1]

**Explanation:** Because nums[0] + nums[1] == 9, we return [0, 1]
```

In [68]:

```python
def twosum(nums,target):
    sorted_nums = sorted(nums)
    sorted_nums = list(enumerate(sorted_nums)) #Sorted in ascending order
    left = 0
    right = len(nums)-1

    while left<right:
        curr_sum = sorted_nums[left][1] + sorted_nums[right][1]
        if curr_sum == target:
            return [sorted_nums[left][0],sorted_nums[right][0]]
        elif curr_sum < target:
            left+=1
        else:
            right-=1
    return []
twosum([2,7,11,15],9)
```

Out[68]:

```
[0, 1]
```

```
Q2. Given an integer array nums and an integer val, remove all occurrences of val in nums in-place. The order of
the elements may be changed. Then return the number of elements in nums which are not equal to val.

Consider the number of elements in nums which are not equal to val be k, to get accepted, you need to do the
following things:
```

```
- Change the array nums such that the first k elements of nums contain the elements which are not equal to val. The
remaining elements of nums are not important as well as the size of nums.
- Return k.

**Example :**
Input: nums = [3,2,2,3], val = 3
Output: 2, nums = [2,2,_*,_*]

**Explanation:** Your function should return k = 2, with the first two elements of nums being 2. It does not matter
what you leave beyond the returned k (hence they are underscores)
```

In [70]:

```python
def remove_val(arr,v):
    i = 0
    for j in range(len(arr)):
        if arr[j] != v:
            arr[i]=arr[j]
            i+=1
    return i
remove_val([3,2,2,3],3)
```

Out[70]:

```
2
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
Q3 Given a sorted array of distinct integers and a target value, return the index if the target is found. If not,
return the index where it would be if it were inserted in order.
```

You must write an algorithm with O(log n) runtime complexity.

Example 1:
Input: nums = [1,3,5,6], target = 5

Output: 2

In [75]:
```
def search(arr, target):
    left = 0
    right = len(arr) - 1

    while left <= right:
        mid = left + (right - left) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return left

index = search([1, 3, 5, 6], 5)
print("Output:", index)
```

```
Output: 2
```

In [ ]:

<aside>
💡 **Q4.** You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return the resulting array of digits.

```
**Example 1:**
Input: digits = [1,2,3]
Output: [1,2,4]

**Explanation:** The array represents the integer 123.

Incrementing by one gives 123 + 1 = 124.
Thus, the result should be [1,2,4].

</aside>
```

In [76]:

```
def plusOne(digits):
    n = len(digits)

    # Start from the least significant digit
    for i in range(n - 1, -1, -1):
        # Increment the current digit by one
        digits[i] += 1

        # Check if there is a carry
        if digits[i] < 10:
            # No carry, we can stop
            return digits
        else:
            # Carry, set the current digit to 0
            digits[i] = 0

    # If there is still a carry at this point, insert a new digit at the beginning
    digits.insert(0, 1)
    return digits

# Example usage:
digits = [1, 2, 3]
result = plusOne(digits)
print("Output:", result)
```

```
Output: [1, 2, 4]
```

```
<aside>
```

💡 **Q5.** You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in non-decreasing order.

The final sorted array should not be returned by the function, but instead be stored inside the array nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

Example 1:
Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
Output: [1,2,2,3,5,6]

Explanation: The arrays we are merging are [1,2,3] and [2,5,6].
The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

</aside>

In [77]:

```python
def merge(nums1, m, nums2, n):
    i = m - 1  # Pointer for nums1
    j = n - 1  # Pointer for nums2
    k = m + n - 1  # Pointer for the merged array

    # Merge from the end of the arrays
    while i >= 0 and j >= 0:
        if nums1[i] >= nums2[j]:
            nums1[k] = nums1[i]
            i -= 1
        else:
            nums1[k] = nums2[j]
            j -= 1
        k -= 1

    # Copy remaining elements from nums2 if any
    while j >= 0:
        nums1[k] = nums2[j]
        j -= 1
        k -= 1

# Example usage:
nums1 = [1, 2, 3, 0, 0, 0]
m = 3
nums2 = [2, 5, 6]
n = 3

merge(nums1, m, nums2, n)
print("Output:", nums1)
```

```
Output: [1, 2, 2, 3, 5, 6]
```

```
<aside>
💡  **Q6.** Given an integer array nums, return true if any value appears at least twice in the array, and return
false if every element is distinct.

**Example 1:**
Input: nums = [1,2,3,1]
```

```
Output: true

</aside>
```

In [78]:

```
def containsDuplicate(nums):
    seen = set()

    for num in nums:
        if num in seen:
            return True
        seen.add(num)

    return False

# Example usage:
nums = [1, 2, 3, 1]
result = containsDuplicate(nums)
print("Output:", result)
```

```
Output: True
```

```
<aside>
💡 **Q7.** Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the
nonzero elements.

Note that you must do this in-place without making a copy of the array.

**Example 1:**
Input: nums = [0,1,0,3,12]
Output: [1,3,12,0,0]

</aside>
```

In [79]:

```python
def moveZeroes(nums):
    left = 0

    # Iterate through the array
    for right in range(len(nums)):
        # If the current element is nonzero
        if nums[right] != 0:
            # Swap it with the element at the left pointer
            nums[left], nums[right] = nums[right], nums[left]
            # Move the left pointer forward
            left += 1

# Example usage:
nums = [0, 1, 0, 3, 12]
moveZeroes(nums)
print("Output:", nums)
```

```
Output: [1, 3, 12, 0, 0]
```

```
<aside>
💡 **Q8.** You have a set of integers s, which originally contains all the numbers from 1 to n. Unfortunately, due to some error, one of the numbers in s got duplicated to another number in the set, which results in repetition of one number and loss of another number.

You are given an integer array nums representing the data status of this set after the error.

Find the number that occurs twice and the number that is missing and return them in the form of an array.

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

</aside>
```

In [80]:

```
def findErrorNums(nums):
    n = len(nums)
    nums_set = set(nums)
    missing_num = sum(range(1, n + 1)) - sum(nums_set)
    duplicate_num = sum(nums) - sum(nums_set)
    return [duplicate_num, missing_num]

# Example usage:
nums = [1, 2, 2, 4]
result = findErrorNums(nums)
print("Output:", result)
```

```
Output: [2, 3]
```

In [ ]: