```
**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]
```

In [2]:

```python
def two_sum(nums,target):

    nums_to_index = {}
    for i, num in enumerate(nums):
        complement = target - num

        if complement in nums_to_index:
            return [nums_to_index[complement],i]
        nums_to_index[num] = i
    return []

two_sum([2,7,11,12],9)
```

Out[2]:

```
[0, 1]
```

```
**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.
```

In [1]:

```python
def searchInsert(nums,target):
    s, e = 0, len(nums)-1

    while s<=e:
        m = s+(e-s)//2

        if target == nums[m]:
            return m
        elif target > nums[m]:
            s = m + 1
        else:
            e = m - 1

    return s

searchInsert([1,3,5,6],5)
```

Out[1]:

```
2
```

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]

In [4]:

```python
def merge(nums1, m, nums2, n):
    nums1_copy = nums1[:m]
    p1=0
    p2=0
    p = 0

    while p < m+n:
        if p2 >= n or (p1 < m and nums1_copy[p1] < nums2[p2]):
            nums1[p] = nums1_copy[p1]
            p1 += 1

        else:
            nums1[p] = nums2[p2]
            p2 += 1

        p += 1

    return nums1

merge([1,2,3,0,0,0],3,[2,5,6],3)
```

Out[4]:

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

```
**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
```

In [5]:

```python
def contains_duplicate(nums):
    seen = set()

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

contains_duplicate([1,2,3,1])
```

Out[5]:

```
True
```

```
**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]
```

In [6]:

```python
def move_zeros(nums):
    i,j = 0,0

    # Iterate through the array
    while i<len(nums):
        if nums[i] != 0:
            # If the current element is non-zero, move it to the
            # j-th position
            nums[j] = nums[i]
            j += 1
        i += 1

    # Fill the remaining positions with zeros
    while j < len(nums):
        nums[j] = 0
        j += 1

    return nums

nums = [0,1,0,3,12]
move_zeros(nums)
```

Out[6]:

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

```
**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]

```

In [8]:

```
def find_missing_and_duplicate(nums):
    n = len(nums)
    nums_set = set(nums)
    missing_num = set(range(1,n+1)) - nums_set


    for num in nums:
        if nums.count(num) > 1:
            duplicate_num = num
            break
    return [duplicate_num, missing_num.pop()]

nums = [1,2,2,4]
find_missing_and_duplicate(nums)
```

Out[8]:

```
[2, 3]
```

In [ ]:

```

```