1)

```
def twoSum(nums, target):
    num_dict = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in num_dict:
            return [num_dict[complement], i]
        num_dict[num] = i
    return None

# Example usage:
nums = [2, 7, 11, 15]
target = 9
result = twoSum(nums, target)
print(result)
```

2)

```
def remove_element(nums, val):
    k = 0  # Initialize the count of elements not equal to val

    for i in range(len(nums)):
        if nums[i] != val:
            nums[k] = nums[i]  # Move the non-val element to the beginning of nums
            k += 1

    return k

# Test case
nums = [3, 2, 2, 3]
val = 3
result = remove_element(nums, val)
print("Output:", result)
print("nums:", nums[:result], end='')
if result < len(nums):
    print(", _*_" * (len(nums) - result))
```

3)

```
def searchInsert(nums, target):
    left = 0
    right = len(nums) - 1

    while left <= right:
        mid = (left + right) // 2

        if nums[mid] == target:
```

```
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return left

# Example usage
nums = [1, 3, 5, 6]
target = 5
index = searchInsert(nums, target)
print(index)

4)
def plusOne(digits):
    carry = 1
    for i in range(len(digits)-1, -1, -1):
        digits[i] += carry
        carry = digits[i] // 10
        digits[i] %= 10
        if carry == 0:
            break
    if carry == 1:
        digits.insert(0, 1)
    return digits

5)

def merge(nums1, m, nums2, n):
    # Initialize two pointers for nums1 and nums2
    i = m - 1
    j = n - 1
    k = m + n - 1

    # Merge the arrays from the end
    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(nums1)  # Output: [1, 2, 2, 3, 5, 6]
```

6)

```
def containsDuplicate(nums):
    # Create an empty set to store unique elements
    unique_nums = set()

    for num in nums:
        # If the current element is already in the set, it means it's a duplicate
        if num in unique_nums:
            return True
        else:
            # Add the current element to the set
            unique_nums.add(num)

    # If we reach this point, it means there are no duplicates
    return False

# Test case
nums = [1, 2, 3, 1]
print(containsDuplicate(nums))
```

7)

```
def moveZeroes(nums):
    """
    Moves all 0's to the end of the array while maintaining the relative order of the nonzero
elements.
    """
    # Initialize two pointers, one to track the position of the last non-zero element
    # and another to iterate through the array
    last_non_zero = 0

    # Iterate through the array
    for i in range(len(nums)):
        # If the current element is non-zero, move it to the last_non_zero position
        if nums[i] != 0:
            nums[last_non_zero] = nums[i]
```

```
            last_non_zero += 1

    # Fill the remaining positions with zeros
    for i in range(last_non_zero, len(nums)):
        nums[i] = 0

    return nums

8)
def findErrorNums(nums):
    n = len(nums)
    xor_sum = 0
    for i in range(1, n+1):
        xor_sum ^= i
    for num in nums:
        xor_sum ^= num
    # Get the rightmost set bit in xor_sum
    rightmost_set_bit = xor_sum & -xor_sum
    num1, num2 = 0, 0
    for num in nums:
        if num & rightmost_set_bit:
            num1 ^= num
        else:
            num2 ^= num
    for i in range(1, n+1):
        if i & rightmost_set_bit:
            num1 ^= i
        else:
            num2 ^= i
    for num in nums:
        if num == num1:
            return [num1, num2]
    return [num2, num1]
```