Day 01

1. Create a function that gets 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.

   Example:

   ```
   Input: nums = [1,3,5,6], target = 5
   Output: 2
   ```

2. Create a function that gets a 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 <= index1 < index2 <= numbers.length`.

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

   Example:

   ```
   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].
   ```

Day 02

1. Create a function to get an integer array `nums` sorted in **non-decreasing order**, return an array of the squares of each number sorted in non-decreasing order.

   Example:

   ```
   Input: nums = [-4,-1,0,3,10]
   Output: [0,1,9,16,100]
   Explanation: After squaring, the array becomes [16,1,0,9,100].
   After sorting, it becomes [0,1,9,16,100].
   ```

2. Create a function to get an array, rotate the array to the right by `k` steps, where `k` is non-negative.

   Example:

   ```
   Input: nums = [1,2,3,4,5,6,7], k = 3
   Output: [5,6,7,1,2,3,4]
   Explanation:
   rotate 1 steps to the right: [7,1,2,3,4,5,6]
   rotate 2 steps to the right: [6,7,1,2,3,4,5]
   rotate 3 steps to the right: [5,6,7,1,2,3,4]
   ```

Day 03

1. Create a function to get an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in `nums`. If target exists, then return its `index`. Otherwise, return `-1`.

   Example:

   ```
   Input: nums = [-1,0,3,5,9,12], target = 9
   Output: 4
   Explanation: 9 exists in nums and its index is 4
   ```

2. Design your implementation of the circular queue. The circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle.

   Implementation the `MyCircularQueue` class:

   - **MyCircularQueue(k)** Initializes the object with the size of the queue to be k.
   - **int Front()** Gets the front item from the queue. If the queue is empty, return -1.
   - **int Rear()** Gets the last item from the queue. If the queue is empty, return -1.
   - **boolean enQueue(int value)** Inserts an element into the circular queue. Return true if the operation is successful.
   - **boolean deQueue()** Deletes an element from the circular queue. Return true if the operation is successful.
   - **boolean isEmpty()** Checks whether the circular queue is empty or not.
   - **boolean isFull()** Checks whether the circular queue is full or not.

     Example:

     ```
     Input
     ["MyCircularQueue", "enQueue", "enQueue", "enQueue", "enQueue", "Rear",
     "isFull", "deQueue", "enQueue", "Rear"]
     [[3], [1], [2], [3], [4], [], [], [], [4], []]
     Output
     [null, true, true, true, false, 3, true, true, true, 4]

     Explanation
     MyCircularQueue myCircularQueue = new MyCircularQueue(3);
     myCircularQueue.enQueue(1); // return True
     myCircularQueue.enQueue(2); // return True
     myCircularQueue.enQueue(3); // return True
     myCircularQueue.enQueue(4); // return False
     myCircularQueue.Rear();     // return 3
     myCircularQueue.isFull();   // return True
     myCircularQueue.deQueue();  // return True
     myCircularQueue.enQueue(4); // return True
     myCircularQueue.Rear();     // return 4
     ```

Day 04

1. Create a function that gets a string s consisting of words and spaces, return the length of the **last word in the string**.

   Example 1:

   ```
   Input: s = "Hello World"
   Output: 5
   Explanation: The last word is "World" with length 5.
   ```

2. The **array-form** of an integer num is an array representing its digits in left to right order.

   - For example, for num = `1321`, the array form is `[1,3,2,1]`.

     Given num, the **array-form** of an integer, and an integer `k`, return the array-form of the integer num + k.

     Example 1:

     ```
     Input: num = [1,2,0,0], k = 34
     Output: [1,2,3,4]
     Explanation: 1200 + 34 = 1234
     ```

Day 05

1. Create a function that gets two binary strings `a` and `b`, return their sum as a binary string.

   Example:

   ```
   Input: a = "11", b = "1"
   Output: "100"
   ```

2. 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].
   ```

Day 06

1. Create a function that gets a string `s`, check if it can be constructed by taking a substring of it and appending multiple copies of the substring together.

   Example 1:

   ```
   Input: s = "abab"
   Output: true
   Explanation: It is the substring "ab" twice.
   ```

   Example 2:

   ```
   Input: s = "aba"
   Output: false
   ```

2. Create a function that checks array is **monotonic** if it is either monotone increasing or monotone decreasing.

   An array nums is monotone increasing if for all `i <= j`, `nums[i] <= nums[j]`. An array `nums` is monotone decreasing if for all `i <= j`, `nums[i] >= nums[j]`.

   Given an integer array nums, return `true` if the given array is monotonic, or `false` otherwise.

   Example:

   ```
   Input: nums = [1,2,2,3]
   Output: true
   ```