* Review Array, Stack, Queue
* Exercise

**Exercise: two sum**

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.

**Example 1:**

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

**Output:** [0,1]

**Output:** 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]

**Exercise: valid parentheses**

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

- Open brackets must be closed by the same type of brackets.

- Open brackets must be closed in the correct order.

**Example 1:**

**Input:** s = "()"

**Output:** true

**Example 2:**

**Input:** s = "()[]{}"

**Output:** true

**Example 3:**

**Input:** s = "(]"

**Output:** false

**Example 4:**

**Input:** s = "([)]"

**Output:** false

**Example 5:**

**Input:** s = "{[]}"

**Output:** true

**Exercise: remove duplicate numbers**

Given a sorted array nums, remove the duplicates in-place such that each element appears only once and returns the new length.

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

**Example 1:**

**Input:** nums = [1,1,2]

**Output:** 2, nums = [1,2]

**Explanation:** Your function should return length = **2**, with the first two elements of *nums* being **1** and **2** respectively. It doesn't matter what you leave beyond the returned length.

**Example 2:**

**Input:** nums = [0,0,1,1,1,2,2,3,3,4]

**Output:** 5, nums = [0,1,2,3,4]

**Explanation:** Your function should return length = **5**, with the first five elements of *nums* being modified to **0**, **1**, **2**, **3**, and **4** respectively. It doesn't matter what values are set beyond the returned length.

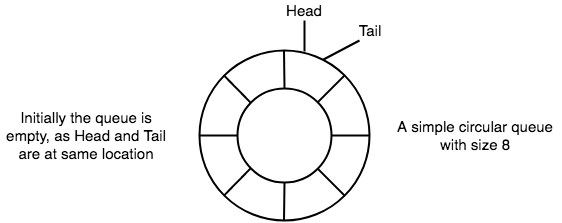
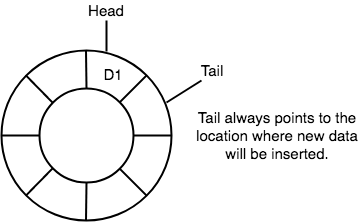
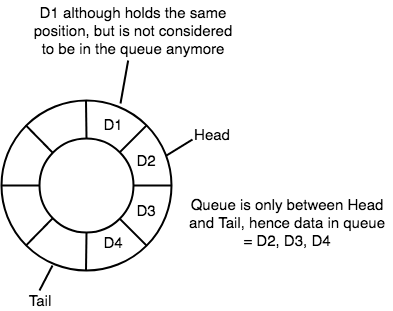
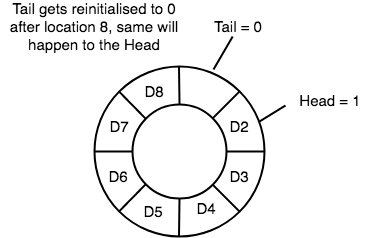
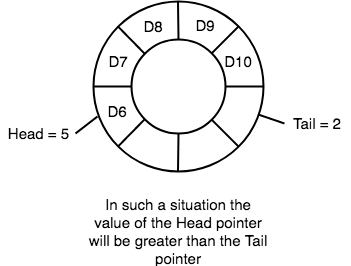
**Exercise: priority queue**

Priority Queue is similar to queue but the element with higher priority can be moved forward to the front. Use exiting Queue class to implement a priority queue.

Element with lower value has higher priority

**Exercise: circular queue**

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. It is also called "Ring Buffer".

1. Init  
   
2. Enqueue D1  
     
   
3. Enqueue D2, D3, D4 and Dequeue D1  
     
   
4. Enqueue D5, D6, D7, D8  
     
   
5. Dequeue D2, D3, D4, D5 and Enqueue D9, D10  
     
   

One of the benefits of the circular queue is that we can make use of the spaces in front of the queue. In a normal queue, once the queue becomes full, we cannot insert the next element even if there is a space in front of the queue. But using the circular queue, we can use the space to store new values.

Implementation of CircularQueue class:

* enqueue(): insert the element
* dequeue(): delete the element
* front(): return the first element in the queue, if queue is empty, return None
* rear(): return the last element in the queue, if queue is empty, return None
* isEmpty(): return true if queue is empty
* isFull(): return true if queue is full

Note: in our implementation of Stack and Queue, we did not fix a size, which is bad as it does not prevent the program accidently create a large stack or queue and use up the memory.