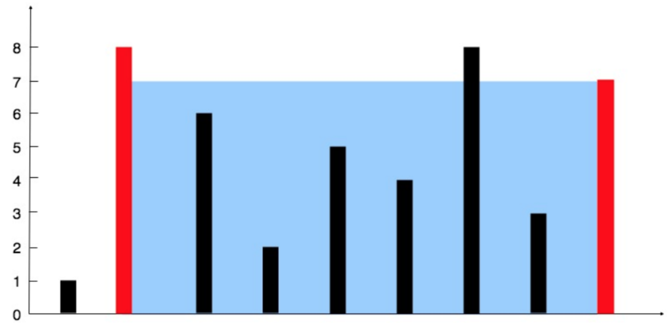
CCC Junior Lesson 6

Derrick Guo

Note: This course note is prepared based on questions from www.leetcode.com, under the answer of each question, you can find a reference number of that question on leetcode. When you teach students using slides, let them go to leetcode.com and copy there solutions there to run tests against their programs. That way we can provide better test cases in both quantity and quality.

1 Container With Most Water

Given n non-negative integers a1, a2, ..., an , where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.
Note: You may not slant the container and n is at least 2.



The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example:

```
Input: [1,8,6,2,5,4,8,3,7]
Output: 49
```

Idea:

Use the 2 pointer approach.

First consider a special case: if size of the list is 2, the return the larger one in the list;
Then define two ”pointers” l and r, pointing to the left and right sides of the list initially. Define current largest height (right), crh, and current largest height left, clh to be 0s initially to hold the current maximum values. Initialize maximum area = 0.
Then begin looping. while l ¡ r, then find the larger height between each side of the box. Say l ¡ r, then calculate the current maximum area, which is the maximum between (r - l) * height(l) and previous maxa. Let clh holds the value of height[l]. This is used for the following loop: ignore every following line if it has a length less than the current maximum length. (Less length means less area, since the length of the other side is fixed and distance is even smaller). If r ¡ l, vice versa.

Answer:

```
def maxArea(self, height):
    """
    :type height: List[int]
    :rtype: int
    """

    n=len(height)
    if n == 2:
        return min(height[0],height[1])
    else:
        l=0
        r=n-1
        clh=0
        crh=0
        maxa=0
        while l<r :

            if height[l]<height[r]:
                maxa=max(height[l]*(r-l),maxa)
                clh=height[l]
                while height[l]<=clh:
                    l+=1
                    if l>= r:
                        return maxa

            else:
                maxa=max(height[r]*(r-l),maxa)
                crh=height[r]
                while height[r]<=crh :
```

```
                r-=1
                if l>= r:
                    return maxa


        return maxa
```

Reference: leetcode #11
This question was a bit hard. Use it to make students take this course seriously :)

2 Search Insert Position

Calm down, not all questions are hard :) We'll do an easy one now.

Given a sorted array 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 may assume no duplicates in the array.

Example 1:

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

Example 2:

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

Example 3:

```
Input: [1,3,5,6], 7
Output: 4
```

Example 4:

```
Input: [1,3,5,6], 0
Output: 0
```

Idea:

Use binary search. If students need recall of how binary search works, go back to lesson 2 slides.

Answer:

```
def searchInsert(self, nums, key):
    if key > nums[len(nums) - 1]:
        return len(nums)

    if key < nums[0]:
        return 0

    l, r = 0, len(nums) - 1
    while l <= r:
        m = (l + r)/2
        if nums[m] > key:
            r = m - 1
            if r >= 0:
                if nums[r] < key:
                    return r + 1
            else:
                return 0

        elif nums[m] < key:
            l = m + 1
            if l < len(nums):
                if nums[l] > key:
                    return l
            else:
                return len(nums)
        else:
            return m
```

Reference: leetcode #35

3 Plus One

Given a non-empty array of digits representing a non-negative integer, plus one to the integer.
The digits are stored such that the most significant digit is at the

head of the list, and each element in the array contain a single digit. You may assume the integer does not contain any leading zero, except the number 0 itself.

Example1:

```
Input: [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
```

Example2:

```
Input: [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
```

Idea:

An easy way to think about this question is to take care of the 9s, since otherwise you can just add 1 to whatever digit at the end. Thus you can run in a loop wherer whenever you meet a 9, change it to 0, and add 1 to the digit where you are not seeing 9s anymore. (The digit before a bunch of 9s, if doesn't exist, ex:9999, just add 1 at the very front.)

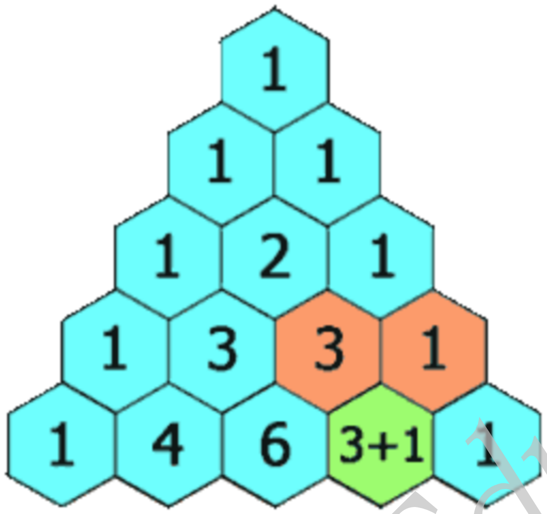
Answer:

```
def plusOne(self, digits):
    length = len(digits) - 1
    while digits[length] == 9:
        digits[length] = 0
        length -= 1
    if(length < 0):
        digits = [1] + digits
    else:
        digits[length] += 1
    return digits
```

Reference: leetcode #66

4 Pascal’s Triangle II

Given a non-negative index k where k 33, return the kth index row of the Pascal’s triangle.
Note that the row index starts from 0.



In Pascal's triangle, each number is the sum of the two numbers directly above it.

Example2:

```
Input: 3
Output: [1,3,3,1]
```

Idea:

Solve this using loops. Notice that each element in the jth position in the ith line is the addition of jth and (j-1)th position from (i-1)th line.

Answer:

```
def getRow(self, rowIndex):
        """
        :type rowIndex: int
        :rtype: List[int]
        """
        res = [1] + [0] * rowIndex
        for i in range(1, rowIndex+1):
            for j in range(i,0,-1):
                res[j] = res[j] + res[j-1]
        return res
```

Reference: leetcode #119

5 Two Sum II - Input array is sorted

Given an array of integers that is already sorted in ascending order, find two numbers such that they add up to a specific target number.
The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2.
Note:

Your returned answers (both index1 and index2) are not zero-based.
You may assume that each input would have exactly one solution and you may not use the same element twice.

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.
```

Idea:

Use the 2-pointer approach. Since the array is sorted, we can have a pointer pointing to the left side of the array, and another pointer pointing to the right side of the array. Then start looping. Once we hit list[l] + list[r] == target, report that we found what was needed; otherwise keep looping and at the end if still not found anything, return "not found".

Answer:

```
# two-pointer
def twoSum1(self, numbers, target):
    l, r = 0, len(numbers)-1
    while l < r:
        s = numbers[l] + numbers[r]
        if s == target:
            return [l+1, r+1]
        elif s < target:
            l += 1
        else:
            r -= 1
```

Reference: leetcode #167
Some more ways of approaching this question:
https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/discuss/51249/Python-different-solutions-(two-pointer-dictionary-binary-search).

6 Rotate Array

Given an array, rotate the array to the right by k steps, where k is non-negative.

Example 1:

```
Input: [1,2,3,4,5,6,7] and 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]
```

Example 2 :

```
Input: [-1,-100,3,99] and k = 2
Output: [3,99,-1,-100]
Explanation:
rotate 1 steps to the right: [99,-1,-100,3]
rotate 2 steps to the right: [3,99,-1,-100]
```

Idea:
This is an easy one :) Use the list slicing functionality of python.

Answer:

```
def rotate(self, nums, k):
        n = len(nums)
        k = k % n
        nums[:] = nums[n-k:] + nums[:n-k]
```

Reference: leetcode #189

7 Best Time to Buy and Sell Stock

Say you have an array for which the ith element is the price of a given stock on day i.
If you were only permitted to complete at most one transaction (i.e., buy one and sell one share of the stock), design an algorithm to find the maximum profit.
Note that you cannot sell a stock before you buy one.

Example 1:

```
Input: [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6),
             profit = 6−1 = 5. Not 7−1 = 6, as selling price needs to
             be larger than buying price.
```

Example 2:

```
Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is done, i.e. max profit = 0.
```

Idea:

Use a variable to hold the current lowest price, loop around and find the largest difference.

Answer:

```
def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        n=len(prices)
        if n<=1:
            return 0
        max_profit=0
        low_price=prices[0]
        for i in range(1,n):
            low_price=min(low_price,prices[i])
            max_profit=max(max_profit, prices[i]−low_price)
        return max_profi
```

Reference: leetcode # 121