Visier Coding Interview Question

1. List Interval

1.1 Merge Intervals (Leetcode 56)

Given a collection of intervals, merge all overlapping intervals.

```
Exapmle 1:
Input: [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlaps, merge them into [1,6
].
```

```python
class Solution(object):
    def merge(self, intervals):
        """
        :type intervals: List[List[int]]
        :rtype: List[List[int]]
        """

        intervals = sorted(intervals, key = lambda a: a[0])
        res = []
        for interval in intervals:
            if not res or interval[0] > res[-1][1]:
                res.append(interval)
            else:
                res[-1][1] = max(res[-1][1], interval[1])
        return res
```

1.2 Interval List Intersections (Leetcode 986)

Given two lists of **closed** intervals, each list of intervals is pairwise disjoint and in sorted order.

```
Input: A = [[0,2],[5,10],[13,23],[24,25]], B = [[1,5],[8,12],[15,24],[25,26
]]
Output: [[1,2],[5,5],[8,10],[15,23],[24,24],[25,25]]
Reminder: The inputs and the desired output are lists of Interval objects,
and not arrays or lists.
```

```python
class Solution(object):
    def intervalIntersection(self, A, B):
        """
        :type A: List[List[int]]
        :type B: List[List[int]]
        :rtype: List[List[int]]
        """
        if not A or not B:
            return None
        i, j = 0, 0
        res = []
        while i < len(A) and j < len(B):
            max_start = max(A[i][0], B[j][0])
            min_end = min(A[i][1], B[j][1])
            if max_start <= min_end:
                res.append([max_start, min_end])
            if A[i][1] < B[j][1]:
                i += 1
            else:
                j += 1
        return res
```

1.3 Range Union

```
Input: A = [[1,4],[7,8],[10,13]], B = [[2,7],[10,15]]
Output: [[1,8],[10,15]]
Reminder: The inputs and the desired output are lists of Interval objects,
and not arrays or lists.
```

```python
class Solution(object):
    def intervalIntersection(self, A, B):
        """
        :type A: List[List[int]]
        :type B: List[List[int]]
        :rtype: List[List[int]]
        """
        if not A or not B:
            return A
        i, j = 0, 0
        res = []
        while i < len(A) and j < len(B):
            if A[i][0] < B[j][0]:
                tmp = A[i]
```

```
                i += 1
            else:
                tmp = B[j]
                j += 1
            if not res or res[-1][1] < tmp[0]:
                res.append(tmp)
            else:
                res[-1][1] = max(res[-1][1], tmp[1])

        if i < len(A):
            tmp = A[i]
            i += 1
            if not res or res[-1][1] < tmp[0]:
                res.append(tmp)
            else:
                res[-1][1] = max(res[-1][1], tmp[1])
            res = res + A[i:]

        if j < len(B):
            tmp = B[j]
            i += 1
            if not res or res[-1][1] < tmp[0]:
                res.append(tmp)
            else:
                res[-1][1] = max(res[-1][1], tmp[1])
            res = res + B[j:]
        return res
```

2. Missing Number

2.1 Find the Duplicate Number

Give a function that given an array of **N-1** integers which contains all numbers from 1 to N except for one number, returns the missing number.

```
For example, findMissing([1, 3, 5, 4]) should return 2
```

```
class Solution(object):
    def findMissingNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
```

```
        """
        n = len(nums) + 1
        m = sum(nums)
        return (n + 1) * n // 2 - m
```

2.2 First Missing Positive (Leetcode 41)

Given an **unsorted integer array**, find the smallest missing positive integer. Your algorithm should run in $O(n)$ time and uses constant extra space.

```
Example 1:
Input: [1,2,0]
Output: 3

Example 2:
Input: [3,4,-1,1]
Output: 2

Example 3:
Input: [7,8,9,11,12]
Output: 1
```

```
class Solution:
    def firstMissingPositive(self, nums: List[int]) -> int:
        n, i = len(nums), 0
        if n == 0: return 1
        while i < n:
        # 获取当前位置的数据，减去1是为了得到 在list要插入的位置
            w = nums[i] - 1
        # 0 < nums[i] <= n 判断是否出界
        # nums[i] != nums[w] 如果相等或者是本身就没必要替换了，避免死循环
            if 0 < nums[i] <= n and nums[i] != nums[w]:
                nums[i], nums[w] = nums[w], nums[i]
            else:
        # 当前位置上的数，没有找到合适的位置，进行下一个位置
                i += 1

        for i in range(n):  # 遍历返回
            if i + 1 != nums[i]:
                return i + 1
        return (n + 1)
```

3. Product Array Puzzle

Given an array **nums** of **n** integers where n > 1, return an array output such that **output[i]** is equal to the product of all the elements of nums except **nums[i]**.

```
Input:  [1,2,3,4]
Output: [24,12,8,6]
```

```python
class Solution(object):
    def productExceptSelf(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        if len(nums) < 2:
            return nums
        left_product = [1]
        for i in range(len(nums) - 1):
            left_product.append(left_product[-1] * nums[i])

        right_product = [1]
        for i in range(len(nums) - 1, 0, -1):
            right_product.append(right_product[-1] * nums[i])
        right_product = right_product[::-1]
        res = []

        for i in range(len(nums)):
            res.append(left_product[i] * right_product[i])
        return res
```

4. Find All Anagrams in a String

Given a string **s** and a non-empty string **p**, find all the **start indices of p's anagrams in s**.

```
Input: s: "cbaebabacd" p: "abc"
Output: [0, 6]
Explanation:
    The substring with start index = 0 is "cba", which is an     anagram of
```

```
"abc".
    The substring with start index = 6 is "bac", which is an anagram of "ab
c".
Output: [24,12,8,6]
```

```python
class Solution(object):
    def findAnagrams(self, s, p):
        """
        :type s: str
        :type p: str
        :rtype: List[int]
        """
        res = []
        if len(s) < len(p):
            return res
        dict_p_word_count = {}
        dict_s_word_count = {}
        for char in p:
            if char in dict_p_word_count:
                dict_p_word_count[char] += 1
            else:
                dict_p_word_count[char] = 1
                dict_s_word_count[char] = 0

        len_p = len(p)
        len_s = len(s)
        for i in range(len(s)):
            if i < len(p):
                if s[i] in dict_s_word_count:
                    dict_s_word_count[s[i]] += 1
            else:
                if dict_s_word_count == dict_p_word_count:
                    res.append(i - len_p)
                if s[i] in dict_s_word_count:
                    dict_s_word_count[s[i]] += 1
                if s[i - len_p] in dict_s_word_count:
                    dict_s_word_count[s[i - len_p]] -= 1
        if dict_s_word_count == dict_p_word_count:
            res.append(len_s - len_p)
        return res
```

4. Longest Substring with At Most K Distinct Characters (Lintcode

386)

Given a string **S**, find the length of the **longest substring T** that contains at most **k distinct characters**.

```
Input: S = "eceba" and k = 3
Output: 4
Explanation: T = "eceb"
```

```python
class Solution:
    """
    @param s: A string
    @param k: An integer
    @return: An integer
    """
    def lengthOfLongestSubstringKDistinct(self, s, k):
        # write your code here
        if not s or k == 0:
            return 0
        dict_word_index = {}
        start_index = 0
        res = 1
        dict_word_index[s[start_index]] = 1

        for i in range(1, len(s)):
            if s[i] in dict_word_index:
                dict_word_index[s[i]] += 1
            else:
                dict_word_index[s[i]] = 1
                if len(dict_word_index) > k:
                    poiter = start_index
                    while len(dict_word_index) > k:
                        dict_word_index[s[poiter]] -= 1
                        if dict_word_index[s[poiter]] == 0:
                            del dict_word_index[s[poiter]]
                        poiter += 1
                    start_index = poiter
            res = max(res, i - start_index + 1)
        return res
```

5. Subarray Sum Equals K (Leetcode 560)

Given **an array of integers** and **an integer k**, you need to find the total number of **continuous subarrays** whose **sum** equals to k.

```
Input:nums = [1,1,1], k = 2
Output: 2
```

```python
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        dict_prefixSum_cnt = {}
        dict_prefixSum_cnt[0] = 1
        prefixSum = 0
        res = 0
        for num in nums:
            prefixSum += num
            if prefixSum - k in dict_prefixSum_cnt:
                res += dict_prefixSum_cnt[prefixSum - k]

            if prefixSum in dict_prefixSum_cnt:
                dict_prefixSum_cnt[prefixSum] += 1
            else:
                dict_prefixSum_cnt[prefixSum] = 1
        return res
```

6. Pow(x, n) (Leetcode 50)

Implement **pow(x, n)**, which calculates **x** raised to the power n **(x^n)**.

```python
class Solution:
    def myPow(self, x: float, n: int) -> float:
        if n < 0:
            return 1 / self.powCalc(x, -n)
        else:
            return self.powCalc(x, n)

    def powCalc(self, a, b):
        if b == 0:
            return 1
        half = self.powCalc(a, b//2)
        if b % 2 == 0:
            return half * half
        else:
```

```
            return half * half * a
```

7. LRU Cache (Leetcode 146)

https://leetcode.com/problems/lru-cache/

8. Find the number (Binary search)

1. Single Element in a Sorted Array (Leetcode 540)

You are given a **sorted array** consisting of only integers where every element appears exactly twice, except for one element which appears exactly once. Find this **single element** that appears only once.

```
Input: [1,1,2,3,3,4,4,8,8]
Output: 2
```

```
class Solution:
    def singleNonDuplicate(self, nums: List[int]) -> int:
        start = 0
        l = len(nums)
        end = len(nums) - 1
        while start <=end:
            mid = (start + end)//2
            if mid<l-1:
                if nums[mid-1] != nums[mid] and nums[mid+1] !=nums[mid]:
                    return nums[mid]
            # if mid is an odd index then:
            # if nums[mid-1] is equal to nums[mid] then single element is after mid else it is before mid
            if mid % 2 == 1:
                if nums[mid-1] == nums[mid]:
                    start = mid + 1
                else:
                    end = mid - 1
            # if mid is even
            # then if nums[mid] == nums[mid+1] then then single element is after mid
            # else it is before mid
```

```
                else:
                    if mid < l -1 and  nums[mid] == nums[mid+1]:
                        start = mid + 1
                    else:
                        end = mid -1
            return nums[mid]
```

2. Search in Rotated Sorted Array (Leetcode 33)

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        if not nums:
            return -1
        l, r, n = 0, len(nums) - 1, len(nums) - 1
        while l <= r:
            mid = (l+r)//2
            if nums[mid] == target:
                return mid
            elif (nums[mid] < nums[r] and not(nums[mid] <= target <= nu
ms[r])) or (nums[mid]>nums[r] and nums[r] <target < nums[mid]):
                r = mid - 1
            else:
                l = mid + 1
        return -1
```

3. Binary Search

```
def binarySearch(arr, l, r, x):

    while l <= r:
        mid = l + (r - l)/2;
        # Check if x is present at mid
        if arr[mid] == x:
            return mid
        # If x is greater, ignore left half
        elif arr[mid] < x:
            l = mid + 1
        # If x is smaller, ignore right half
        else:
            r = mid - 1
```

```
14.         # If we reach here, then the element
15.         # was not present
16.         return -1
```