Leetcode solutions in Python

Q1-50

Q1:

Given an array of integers, return indices of the two numbers such that they add up to a specific target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

Example:

Given nums = [2, 7, 11, 15], target = 9,Because nums[0] + nums[1] = 2 + 7 = 9,return [0, 1].

**Solution:**

```
class Solution(object):
    def twoSum(self, nums, target):
        d = {}
        for i, num in enumerate(nums):
            if target - num in d:
                return [d[target - num], i]
            d[num] = i
```

Q2:

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

**Solution:**

```
class Solution(object):
    def addTwoNumbers(self, l1, l2):
            p = dummy = ListNode(-1)
            carry = 0
            while l1 or l2 or carry:
                val = (l1 and l1.val or 0) + (l2 and l2.val or 0) + carry
                carry = val / 10
                p.next = ListNode(val % 10)
                l1 = l1 and l1.next
                l2 = l2 and l2.next
                p = p.next
            return dummy.next
```

Q3:

Given a string, find the length of the longest substring without repeating characters.

Examples:

Given "abcabcbb", the answer is "abc", which the length is 3.

**Solution:**

```
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        d = {}
        start = 0
        ans = 0
        for i,c in enumerate(s):
            if c in d:
                start = max(start, d[c] + 1)
            d[c] = i
            ans = max(ans, i - start + 1)
        return ans
```

Q4:

There are two sorted arrays nums1 and nums2 of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

Example:

nums1 = [1, 3]

nums2 = [2]

The median is 2.0

**Solution:**

```
class Solution(object):
    def findMedianSortedArrays(self, nums1, nums2):
        a, b = sorted((nums1, nums2), key=len)
        m, n = len(a), len(b)
        after = (m + n - 1) / 2
        lo, hi = 0, m
        while lo < hi:
            i = (lo + hi) / 2
            if after-i-1 < 0 or a[i] >= b[after-i-1]:
                hi = i
            else:
                lo = i + 1
                i = lo
                nextfew = sorted(a[i:i+2] + b[after-i:after-i+2])
        return (nextfew[0] + nextfew[1 - (m+n)%2]) / 2.0
```

Q5:

Given a string s, find the longest palindromic substring in s. You may assume that the maximum length of s is 1000.

Example:

Input: "babad"

Output: "bab"

**Solution:**

```
class Solution(object):
    def longestPalindrome(self, s):
        left = right = 0
        n = len(s)
        for i in range(n - 1):
            if 2 * (n - i) + 1 < right - left + 1:
                break
            l = r = i
            while l >= 0 and r < n and s[l] == s[r]:
                l -= 1
                r += 1
            if r - l - 2 > right - left:
                left = l + 1
                right = r - 1
            l = i
            r = i + 1
            while l >= 0 and r < n and s[l] == s[r]:
                l -= 1
                r += 1
            if r - l - 2 > right - left:
                left = l + 1
                right = r - 1
        return s[left:right + 1]
```

Q6:

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

Write the code that will take a string and make this conversion given a number of rows:

string convert(string text, int nRows);

convert("PAYPALISHIRING", 3) should return "PAHNAPLSIIGYIR".

**Solution:**

```
class Solution(object):
    def convert(self, s, numRows):
        if numRows <= 1:
            return s
        n = len(s)
        ans = []
        step = 2 *(numRows - 1)
        for i in range(numRows):
            one = i
            two = -i
            while one < n or two < n:
                if 0 <= two < n and one != two and i != numRows - 1:
                    ans.append(s[two])
                if one < n:
```

```
                ans.append(s[one])
            one += step
            two += step
        return "".join(ans)
```

Q7:

Reverse digits of an integer.

Example1: x = 123, return 321

Example2: x = -123, return -321

For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

**Solution:**

```
class Solution(object):
    def reverse(self, x):
        sign = x < 0 and -1 or 1
        x = abs(x)
        ans = 0
        while x:
            ans = ans * 10 + x % 10
            x /= 10
     if ans<=0x7fffffff:
        return sign*ans
     else:
        return 0
```

Q8:

Implement atoi to convert a string to an integer.

**Solution:**

```
class Solution(object):
    def myAtoi(self, s):
        s = s.strip()
        sign = 1
        ans=0
        if not s:
            return 0
        if s[0] in ["+", "-"]:
            if s[0] == "-":
                sign = -1
            s = s[1:]
        for c in s:
            if c.isdigit():
                ans = ans * 10 + int(c)
            else:
                break
```

```
        ans *= sign
        if ans > 2147483647:
            return 2147483647
        if ans < -2147483648:
            return -2147483648
        return ans
```

Q9:

Determine whether an integer is a palindrome（回文）. Do this without extra space.

**Solution:**

```
class Solution(object):
    def _isPalindrome(self, x):
        z = x
        y = 0
        while x > 0:
            y = y * 10 + x % 10
            x /= 10
        return z == y
```

Q10:

Implement regular expression matching with support for '.' and '*'.

'.' Matches any single character.

'*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

The function prototype should be:bool isMatch(const char *s, const char *p)

**Solution:**

```
class Solution(object):
    def isMatch(self, s, p):
        dp = [[False] * (len(p) + 1) for _ in range(len(s) + 1)]
        dp[0][0] = True
        for j in range(1, len(p) + 1):
            if p[j - 1] == "*":
                dp[0][j] = dp[0][j - 2]
        for i in range(1, len(s) + 1):
            for j in range(1, len(p) + 1):
                if p[j - 1] != "*":
                    dp[i][j] = dp[i-1][j-1] and (s[i -1] == p[j - 1] or p[j - 1] == ".")
                else:
                    dp[i][j] = dp[i][j - 2] or dp[i - 1][j] and (p[j - 2] == s[i - 1] or p[j - 2] == ".")
        return dp[-1][-1]
```

Q11:

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.

**Solution:**

```
class Solution(object):
        def maxArea(self, height):
        ans = left = 0
        right = len(height) - 1
        while left < right:
                ans = max(ans, (right - left) * min(height[left], height[right]))
                if height[left] <= height[right]:
                        left += 1
                else:
                   right -= 1
        return ans
```

Q12:

Given an integer, convert it to a roman numeral.

Input is guaranteed to be within the range from 1 to 3999.

**Solution:**

```
class Solution(object):
    def intToRoman(self, num): ans = ""
        values = {"M": 1000, "D": 500, "C": 100, "L": 50, "X": 10, "V": 5, "I": 1}
        literals = ["M", "D", "C", "L", "X", "V", "I"]
        for idx in [0, 2, 4]:
            k = num / values[literals[idx]]
            re = (num % values[literals[idx]]) / values[literals[idx + 2]]
            ans += k * literals[idx]
            if re >= 9:
                ans += literals[idx + 2] + literals[idx]
            elif re >= 5:
                ans += literals[idx + 1] + (re - 5) * literals[idx + 2]
            elif re == 4:
                ans += literals[idx + 2] + literals[idx + 1]
            else:
                ans += re * literals[idx + 2]
            num %= values[literals[idx + 2]]
        return ans
```

Q13:

Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

**Solution:**

```
class Solution(object):
    def romanToInt(self, s):
```

```
        d = {"I":1, "V": 5, "X":10,"L":50,"C":100, "D":500, "M":1000}
        ans = 0
        for i in xrange(0, len(s) - 1):
            c = s[i]
            cafter = s[i + 1]
            if d[c] < d[cafter]:
                ans -= d[c]
            else:
                ans += d[c]
        ans += d[s[-1]]
        return ans
```

Q14:

Write a function to find the longest common prefix string amongst an array of strings.

**Solution:**

```
class Solution(object):
    def longestCommonPrefix(self, strs):
        if len(strs) == 0:
            return ""
        i = 0
        j = 0
        end = 0
        while j < len(strs) and i < len(strs[j]):
            if j == 0:
                char = strs[j][i]
            else:
                if strs[j][i] != char:
                    break
            if j == len(strs) - 1:
                i += 1
                j = 0
                end += 1
            else:
                j += 1
        return strs[j][:end]
```

Q15:

Given an array S of n integers, are there elements a, b, c in S such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

Note: The solution set must not contain duplicate triplets.

For example, given array S = [-1, 0, 1, 2, -1, -4],

A solution set is:

[[-1, 0, 1],[-1, -1, 2]]

**Solution:**

```
class Solution(object):
    def threeSum(self, nums):
        res = []
        nums.sort()
        for i in xrange(0, len(nums)):
            if i > 0 and nums[i] == nums[i - 1]:
                continue
            target = 0 - nums[i]
            start, end = i + 1, len(nums) - 1
            while start < end:
                if nums[start] + nums[end] > target:
                    end -= 1
                elif nums[start] + nums[end] < target:
                    start += 1
                else:
                    res.append((nums[i], nums[start], nums[end]))
                    end -= 1
                    start += 1
                    while start < end and nums[end] == nums[end + 1]:
                        end -= 1
                    while start < end and nums[start] == nums[start - 1]:
                        start += 1
        return res
```

Q16:

Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

For example, given array S = {-1 2 1 -4}, and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

**Solution:**

```
class Solution(object):
    def threeSumClosest(self, nums, target):
        nums.sort()
        ans = 0
        diff = float("inf")
        for i in xrange(0, len(nums)):
            start, end = i + 1, len(nums) - 1
            while start < end:
                sum = nums[i] + nums[start] + nums[end]
                if sum > target:
                    if abs(target - sum) < diff:
                        diff = abs(target - sum)
                        ans = sum
                    end -= 1
```

```
                else:
                    if abs(target - sum) < diff:
                        diff = abs(target - sum)
                        ans = sum
                    start += 1
        return ans
```

Q17:

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.

Input:Digit string "23"

Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

**Solution:**

```
class Solution(object):
    def letterCombinations(self, digits):
        if len(digits) == 0:
            return []
        d = {1: "", 2: "abc", 3: "def", 4: "ghi", 5: "jkl", 6: "mno", 7: "pqrs", 8: "tuv", 9: "wxyz"}

        def dfs(digits, index, path, res, d):
            if index == len(digits):
                res.append("".join(path))
                return

            digit = int(digits[index])
            for c in d.get(digit, []):
                path.append(c)
                dfs(digits, index + 1, path, res, d)
                path.pop()
        res = []
        dfs(digits, 0, [], res, d)
        return res
```

Q18:

Given an array S of n integers, are there elements a, b, c, and d in S such that a + b + c + d = target?

Find all unique quadruplets in the array which gives the sum of target.

Note: The solution set must not contain duplicate quadruplets.

**Solution:**

```
class Solution(object):
    def fourSum(self, nums, target):
        nums.sort()
        res = []
        for i in xrange(0, len(nums)):
            if i > 0 and nums[i] == nums[i - 1]:
```

```
                continue
            for j in xrange(i + 1, len(nums)):
                if j > i + 1 and nums[j] == nums[j - 1]:
                    continue
                start = j + 1
                end = len(nums) - 1
                while start < end:
                    sum = nums[i] + nums[j] + nums[start] + nums[end]
                    if sum < target:
                        start += 1
                    elif sum > target:
                        end -= 1
                    else:
                        res.append((nums[i], nums[j], nums[start], nums[end]))
                        start += 1
                        end -= 1
                        while start < end and nums[start] == nums[start - 1]:
                            start += 1
                        while start < end and nums[end] == nums[end + 1]:
                            end -= 1
        return res
```

Q19:

Given a linked list, remove the nth node from the end of list and return its head.

For example,

Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.

**Solution:**

```
class Solution(object):
    def removeNthFromEnd(self, head, n):
        dummy = ListNode(-1)
        dummy.next = head
        fast = slow = dummy
        while n and fast:
            fast = fast.next
            n -= 1
        while fast.next and slow.next:
            fast = fast.next
            slow = slow.next
        slow.next = slow.next.next
        return dummy.next
```

Q20:

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

valid.The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

**Solution:**

```
class Solution(object):
    def isValid(self, s):
        stack = []
        d = ["()", "[]", "{}"]
        for i in xrange(0, len(s)):
            stack.append(s[i])
            if len(stack) >= 2 and stack[-2]+stack[-1] in d:
                stack.pop()
                stack.pop()
        return len(stack) == 0
```

Q21:

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

**Solution:**

```
class Solution(object):
    def mergeTwoLists(self, l1, l2):
        head = dummy = ListNode(-1)
        while l1 and l2:
            if l1.val < l2.val:
                head.next = l1
                l1 = l1.next
            else:
                head.next = l2
                l2 = l2.next
            head = head.next
        if l1:
            head.next = l1
        if l2:
            head.next = l2
        return dummy.next
```

Q22:

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given n = 3, a solution set is:["((()))","(()())","(())()","()(())", "()()()"]

**Solution:**

```
class Solution(object):
    def generateParenthesis(self, n):
       def dfs(left, path, res, n):
            if len(path) == 2 * n:
```

```
                if left == 0:
                    res.append("".join(path))
                return
            if left < n:
                path.append("(")
                dfs(left + 1, path, res, n)
                path.pop()
            if left > 0:
                path.append(")")
                dfs(left - 1, path, res, n)
                path.pop()

        res = []
        dfs(0, [], res, n)
        return res
```

Q23:

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

**Solution:**

```
class Solution(object):
    def mergeKLists(self, lists):
        heap = []
        p = dummy = ListNode(-1)
        for i in xrange(0, len(lists)):
            node = lists[i]
            if not node:
                continue
            heapq.heappush(heap,(node.val, node))

        while heap:
            value, node = heapq.heappop(heap)
            p.next = node
            p = p.next
            if node.next:
                node = node.next
                heapq.heappush(heap, (node.val, node))
        return dummy.next
```

Q24:

Given a linked list, swap every two adjacent nodes and return its head.

For example,

Given 1->2->3->4, you should return the list as 2->1->4->3.

**Solution:**

```
class Solution(object):
```

```
    def swapPairs(self, head):
        def reverseList(head, k):
            pre = None
            cur = head
            while cur and k > 0:
                tmp = cur.next
                cur.next = pre
                pre = cur
                cur = tmp
                k -= 1
            head.next = cur
            return cur, pre
        if not head or not head.next:
            return head
        ret = head.next
        p = head
        pre = None
        while p:
            next, newHead = reverseList(p, 2)
            if pre:
                pre.next = newHead
            pre = p
            p = next
        return ret
```

Q25:

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

You may not alter the values in the nodes, only nodes itself may be changed.

Only constant memory is allowed.

For example,Given this linked list: 1->2->3->4->5For k = 2, you should return: 2->1->4->3->5

**Solution:**

```
class Solution(object):
    def reverseKGroup(self, head, k):
        def reverseList(head, k):
            pre = None
            cur = head
            while cur and k > 0:
                tmp = cur.next
                cur.next = pre
                pre = cur
                cur = tmp
                k -= 1
            head.next = cur
```

```
            return cur, pre

        length = 0
        p = head
        while p:
            length += 1
            p = p.next
        if length < k:
            return head
        step = length / k
        ret = None
        pre = None
        p = head
        while p and step:
            next, newHead = reverseList(p, k)
            if ret is None:
                ret = newHead
            if pre:
                pre.next = newHead
            pre = p
            p = next
            step -= 1
        return ret
```

Q26:

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example,Given input array nums = [1,1,2],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 new length.

**Solution:**

```
class Solution(object):
    def removeDuplicates(self, nums):
        if len(nums) <= 1:
            return len(nums)
        slow = 0
        for i in xrange(1, len(nums)):
            if nums[i] != nums[slow]:
                slow += 1
                nums[slow] = nums[i]
        return slow + 1
```

Q27:

Given an array and a value, remove all instances of that value in place and return the new length.
Do not allocate extra space for another array, you must do this in place with constant memory.
The order of elements can be changed. It doesn't matter what you leave beyond the new length.
Example:Given input array nums = [3,2,2,3], val = 3.Your function should return length = 2, with the first two elements of nums being 2.

**Solution:**

```
class Solution(object):
    def removeElement(self, nums, val):
        slow = -1
        for i in xrange(0, len(nums)):
            if nums[i] != val:
                slow += 1
                nums[slow] = nums[i]
        return slow + 1
```

Q28:

Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

**Solution:**

```
class Solution(object):
    def strStr(self, haystack, needle):
        if len(haystack) == len(needle):
            if haystack == needle:
                return 0
            else:
                return -1
        for i in xrange(0, len(haystack)):
            k = i
            j = 0
            while j < len(needle) and k < len(haystack) and haystack[k] == needle[j]:
                j += 1
                k += 1
            if j == len(needle):
                return i
        return -1 if needle else 0
```

Q29:

Divide two integers without using multiplication, division and mod operator.If it is overflow, return MAX_INT.

**Solution:**

```
class Solution(object):
    def divide(self, dividend, divisor):
        if divisor == 0:
            return 0x7fffffff
```

```
        sign = 1
        if dividend * divisor < 0:
            sign = -1
        ans = 0
        cnt = 1
        dividend = abs(dividend)
        divisor = abs(divisor)
        subsum = divisor
        while dividend >= divisor:
            while (subsum << 1) <= dividend:
                cnt <<= 1
                subsum <<= 1
            ans += cnt
            cnt = 1
            dividend -= subsum
            subsum = divisor
        return max(min(sign * ans, 0x7fffffff), -2147483648)
```

Q30:

You are given a string, s, and a list of words, words, that are all of the same length. Find all starting indices of substring(s) in s that is a concatenation of each word in words exactly once and without any intervening characters.

For example, given:

s: "barfoothefoobarman"

words: ["foo", "bar"]

You should return the indices: [0,9].

(order does not matter).

**Solution:**

```
from collections import deque
class Solution(object):
    def findSubstring(self, s, words):
        if len(words) > len(s):
            return []
        d = {}
        t = {}
        ans = []
        deq = deque([])
        wl = len(words[0])
        fullscore = 0
        for word in words:
            d[word] = d.get(word, 0) + 1
            fullscore += 1

        for i in xrange(0, len(s)):
```

```
            head = start = i
            t.clear()
            score = 0

            while start + wl <= len(s) and s[start:start + wl] in d:
                cword = s[start:start + wl]
                t[cword] = t.get(cword, 0) + 1
                if t[cword] <= d[cword]:
                    score += 1
                else:
                    break
                start += wl

            if score == fullscore:
                ans.append(head)

        return ans
```

Q31：

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.
If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).
The replacement must be in-place, do not allocate extra memory.

**Solution:**

```
class Solution(object):
    def nextPermutation(self, nums):
        if nums is None or len(nums) <= 1:
            return
        pos = None
        p = len(nums) - 2
        while p >= 0:
            if nums[p + 1] > nums[p]:
                pos = p
                break
            p -= 1

        if pos is None:
            self.reverse(nums, 0, len(nums) - 1)
            return
        minPos, minV = pos + 1, nums[pos + 1]
        for i in xrange(pos + 1, len(nums)):
            if nums[i] <= minV and nums[i] > nums[pos]:
                minV = nums[i]
```

```
                minPos = i
        nums[pos], nums[minPos] = nums[minPos], nums[pos]
        self.reverse(nums, pos + 1, len(nums) - 1)

    def reverse(self, nums, start, end):
        while start < end:
            nums[start], nums[end] = nums[end], nums[start]
            start += 1
            end -= 1
```

Q32:

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

For "(()", the longest valid parentheses substring is "()", which has length = 2.

**Solution:**

```
class Solution(object):
    def longestValidParentheses(self, s):
        dp = [0 for _ in xrange(0, len(s))]
        left = 0
        ans = 0
        for i in xrange(0, len(s)):
            if s[i] == "(":
                left += 1
            elif left > 0:
                left -= 1
                dp[i] = dp[i-1] + 2
                j = i - dp[i]
                if j >= 0:
                    dp[i] += dp[j]
                ans = max(ans, dp[i])
        return ans
```

Q33:

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand. (i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

**Solution:**

```
class Solution(object):
    def search(self, nums, target):
        if not nums:
            return -1
        left = 0
        right = len(nums) - 1
```

```
        while left <= right:
            mid = (right + left) / 2
            if nums[mid] == target:
                return mid
            if nums[mid] >= nums[left]:
                if nums[left] <= target <= nums[mid]:
                    right = mid - 1
                else:
                    left = mid + 1
            else:
                if nums[mid] <= target <= nums[right]:
                    left = mid + 1
                else:
                    right = mid - 1
        return -1
```

Q34:

Given an array of integers sorted in ascending order, find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of O(log n).

If the target is not found in the array, return [-1, -1].

For example,Given [5, 7, 7, 8, 8, 10] and target value 8,return [3, 4].

**Solution:**

```
class Solution(object):
    def searchRange(self, nums, target):
        l, r = 0, len(nums) - 1
        found = 0
        start, end = 0, 0
        while l < r:
            m = l + (r - l) / 2
            if target > nums[m]:
                l = m + 1
            else:
                if target == nums[m]:
                    found += 1
                r = m - 1

        if nums[l] == target:
            found += 1

        start = r
        if nums[r] != target or r < 0:
            start = r + 1
```

```
        l, r = 0, len(nums) - 1
        while l < r:
            m = l + (r - l) / 2
            if target < nums[m]:
                r = m - 1
            else:
                if target == nums[m]:
                    found += 1
                l = m + 1
        end = l
        if nums[l] != target:
            end = l - 1

        if found == 0:
            return [-1, -1]
        return [start, end]
```

Q35:

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.

**Solution:**

```
class Solution(object):
    def searchInsert(self, nums, target):
            lo = 0
        hi = len(nums)
        while lo < hi:
            mid = lo + (hi - lo) / 2
            if nums[mid] > target:
                hi = mid
            elif nums[mid] < target:
                lo = mid + 1
            else:
                return mid
        return lo
```

Q36:

Determine if a Sudoku is valid, according to: Sudoku Puzzles - The Rules.

The Sudoku board could be partially filled, where empty cells are filled with the character '.'.

**Solution:**

```
class Solution(object):
    def isValidSudoku(self, board):
        cacheCol = [[0] * 9 for _ in xrange(0, 10)]
        cacheRow = [[0] * 9 for _ in xrange(0, 10)]
```

```
        cacheBox = [[0] * 9 for _ in xrange(0, 10)]

        for i in xrange(0, 9):
            for j in xrange(0, 9):
                ib = (i/3)*3 + j/3
                if board[i][j] == ".":
                    continue
                num = int(board[i][j]) - 1
                if cacheRow[i][num] != 0 or cacheCol[j][num] != 0 or cacheBox[ib][num] != 0:
                    return False
                cacheRow[i][num] = 1
                cacheCol[j][num] = 1
                cacheBox[ib][num] = 1
        return True
```

Q37:

Write a program to solve a Sudoku puzzle by filling the empty cells.

Empty cells are indicated by the character '.'.

You may assume that there will be only one unique solution.

**Solution:**

```
class Solution(object):
    def solveSudoku(self, board):
        cacheBox = [[0] * len(board) for _ in range(len(board))]
        cacheRow = [[0] * len(board) for _ in range(len(board))]
        cacheCol = [[0] * len(board) for _ in range(len(board))]

        def helper(board, i, j, cacheRow, cacheCol, cacheBox):
            if board[i][j] == ".":
                for k in range(1, 10):
                    if i < 0 or i >= len(board) or j < 0 or j >= len(board):
                        continue
                    ib = (i/3) * 3 + j / 3
                    if cacheRow[i][k - 1] == 1 or cacheCol[j][k - 1] == 1or cacheBox[ib][k - 1]
== 1:
                        continue

                    cacheRow[i][k - 1] = cacheCol[j][k - 1] = cacheBox[ib][k - 1] = 1
                    board[i][j] = str(k)
                    if i == j == len(board) - 1:
                        return True
                    if i + 1 < len(board):
                        if helper(board, i + 1, j, cacheRow, cacheCol, cacheBox):
                            return True
                    elif j + 1 < len(board):
```

```
                        if helper(board, 0, j + 1, cacheRow, cacheCol, cacheBox):
                            return True
                    board[i][j] = "."
                    cacheRow[i][k - 1] = cacheCol[j][k - 1] = cacheBox[ib][k - 1] = 0
            else:
                if i == j == len(board) - 1:
                    return True
                if i + 1 < len(board):
                    if helper(board, i + 1, j, cacheRow, cacheCol, cacheBox):
                        return True
                elif j + 1 < len(board):
                    if helper(board, 0, j + 1, cacheRow, cacheCol, cacheBox):
                        return True
            return False

        for i in range(len(board)):
            for j in range(len(board)):
                if board[i][j] != ".":
                    ib = (i/3) * 3 + j / 3
                    k = int(board[i][j]) - 1
                    cacheRow[i][k] = cacheCol[j][k] = cacheBox[ib][k] = 1
        print helper(board, 0, 0, cacheRow, cacheCol, cacheBox)
```

Q38:

The count-and-say sequence is the sequence of integers with the first five terms as following:

1. 1
2. 11
3. 21
4. 1211
5. 111221

1 is read off as "one 1" or 11.

11 is read off as "two 1s" or 21.

21 is read off as "one 2, then one 1" or 1211.

Given an integer n, generate the nth term of the count-and-say sequence.

Note: Each term of the sequence of integers will be represented as a string.

**Solution:**

```
class Solution(object):
    def countAndSay(self, n):

        ans = "1"
        n -= 1
        while n > 0:
            res = ""
            pre = ans[0]
```

```
            count = 1
            for i in range(1, len(ans)):
                if pre == ans[i]:
                    count += 1
                else:
                    res += str(count) + pre
                    pre = ans[i]
                    count = 1
            res += str(count) + pre
            ans = res
            n -= 1
        return ans
```

Q39:

Given a set of candidate numbers (C) (without duplicates) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.

Note:

All numbers (including target) will be positive integers.

The solution set must not contain duplicate combinations.

**Solution:**

```
class Solution(object):
    def combinationSum(self, candidates, target):

        def dfs(candidates, start, target, path, res):
            if target == 0:
                return res.append(path + [])

            for i in range(start, len(candidates)):
                if target - candidates[i] >= 0:
                    path.append(candidates[i])
                    dfs(candidates, i, target - candidates[i], path, res)
                    path.pop()
        res = []
        dfs(candidates, 0, target, [], res)
        return res
```

Q40:

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

Each number in C may only be used once in the combination.

Note:

All numbers (including target) will be positive integers.

The solution set must not contain duplicate combinations.

**Solution:**

```
class Solution(object):
    def combinationSum2(self, candidates, target):
        def dfs(nums, target, start, visited, path, res):
            if target == 0:
                res.append(path + [])
                return

            for i in xrange(start, len(nums)):
                if i > start and nums[i] == nums[i - 1]:
                    continue
                if target - nums[i] < 0:
                    return 0
                if i not in visited:
                    visited.add(i)
                    path.append(nums[i])
                    dfs(nums, target - nums[i], i + 1, visited, path, res)
                    path.pop()
                    visited.discard(i)

        candidates.sort()
        res = []
        visited = set([])
        dfs(candidates, target, 0, visited, [], res)
        return res
```

Q41:

Given an unsorted integer array, find the first missing positive integer.

For example,Given [1,2,0] return 3,and [3,4,-1,1] return 2.

**Solution:**

```
class Solution(object):
    def firstMissingPositive(self, nums):
        i = 0
        while i < len(nums):
            if 0 < nums[i] <= len(nums) and nums[nums[i] - 1] != nums[i]:
                nums[nums[i] - 1], nums[i] = nums[i], nums[nums[i] - 1]
            else:
                i += 1

        for i in xrange(0, len(nums)):
            if nums[i] != i + 1:
                return i + 1
        return len(nums) + 1
```

Q42:

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

For example, Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.

**Solution:**

```
class Solution(object):
    def trap(self, height):
        ans = left = 0
        right = len(height) - 1
        leftWall = rightWall = float("-inf")
        while left <= right:
            if leftWall <= rightWall:
                ans += max(0, leftWall - height[left])
                leftWall = max(leftWall, height[left])
                left += 1
            else:
                ans += max(0, rightWall - height[right])
                rightWall = max(rightWall, height[right])
                right -= 1
        return ans
```

Q43:

Given two non-negative integers num1 and num2 represented as strings, return the product of num1 and num2.

**Solution:**

```
class Solution(object):
    def multiply(self, num1, num2):
        ans = [0] * (len(num1) + len(num2))
        for i, n1 in enumerate(reversed(num1)):
            for j, n2 in enumerate(reversed(num2)):
                ans[i + j] += int(n1) * int(n2)
                ans[i + j + 1] += ans[i + j] / 10
                ans[i + j] %= 10
        while len(ans) > 1 and ans[-1] == 0:
            ans.pop()
        return "".join(map(str, ans[::-1]))
```

Q44:

Implement wildcard pattern matching with support for '?' and '*'.

'?' Matches any single character.

'*' Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial).

The function prototype should be:bool isMatch(const char *s, const char *p)

**Solution:**

```
class Solution(object):
    def isMatch(self, s, p):
        i = j = 0
        lenS = len(s)
        lenP = len(p)
        lastMatchPos = 0
        lastStarPos = -1
        while i < len(s):
            if j < lenP and p[j] in (s[i], "?"):
                i += 1
                j += 1
            elif j < lenP and p[j] == "*":
                lastMatchPos = i
                lastStarPos = j
                j += 1
            elif lastStarPos > -1:
                i = lastMatchPos + 1
                lastMatchPos += 1
                j = lastStarPos + 1
            else:
                return False
        while j < lenP and p[j] == "*":
            j += 1
        return j == lenP
```

Q45:

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

**Solution:**

```
class Solution(object):
    def jump(self, nums):
        pos = 0
        ans = 0
        bound = len(nums)
        while pos < len(nums) - 1:
            dis = nums[pos]
            farthest = posToFarthest = 0
            for i in xrange(pos + 1, min(pos + dis + 1, bound)):
                canReach = i + nums[i]
                if i == len(nums) - 1:
                    return ans + 1
                if canReach > farthest:
                    farthest = canReach
```

```
                    posToFarthest = i
            ans += 1
            pos = posToFarthest
        return ans
```

Q46:

Given a collection of distinct numbers, return all possible permutations.

**Solution:**

```
class Solution(object):
    def permute(self, nums):
        res = []
        visited = set([])
        def dfs(nums, path, res, visited):
            if len(path) == len(nums):
                res.append(path + [])
                return

            for i in xrange(0, len(nums)):
                if i not in visited:
                    visited.add(i)
                    path.append(nums[i])
                    dfs(nums, path, res, visited)
                    path.pop()
                    visited.discard(i)

        dfs(nums, [], res, visited)
        return res
```

Q47:

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

**Solution:**

```
class Solution(object):
    def permuteUnique(self, nums):
                    res = []
        nums.sort()
        def dfs(nums, res, path, visited):
            if len(path) == len(nums):
                res.append(path + [])
                return

            for i in range(len(nums)):
                if i in visited:
                    continue
```

```
                if i > 0 and nums[i] == nums[i - 1] and i - 1 not in visited:
                    continue
                visited |= {i}
                path.append(nums[i])
                dfs(nums, res, path, visited)
                path.pop()
                visited -= {i}

        dfs(nums, res, [], set())
        return res
```

Q48:

You are given an n x n 2D matrix representing an image.Rotate the image by 90 degrees (clockwise).

Note:You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

**Solution:**

```
class Solution(object):
    def rotate(self, matrix):
        if len(matrix) == 0:
            return
        h = len(matrix)
        w = len(matrix[0])
        for i in xrange(0, h):
            for j in xrange(0, w/2):
                matrix[i][j], matrix[i][w - j - 1] = matrix[i][w - j - 1], matrix[i][j]

        for i in xrange(0, h):
            for j in xrange(0, w - 1 - i):
                matrix[i][j], matrix[w - 1 - j][h - 1 - i] = matrix[w - 1 - j][h - 1 - i], matrix[i][j]
```

Q49:

Given an array of strings, group anagrams together.

For example, given: ["eat", "tea", "tan", "ate", "nat", "bat"],

Return:[  ["ate", "eat","tea"], ["nat","tan"],["bat"]]

**Solution:**

```
class Solution(object):
    def groupAnagrams(self, strs):
        def hash(count):
            p1, p2 = 2903, 29947
            ret = 0
            for c in count:
                ret = ret * p1 + c
                p1 *= p2
```

```
            return ret

        d = {}
        for str in strs:
            count = [0] * 26
            for c in str:
                count[ord(c) - ord('a')] += 1
            key = hash(count)
            if key not in d:
                d[key] = [str]
            else:
                d[key].append(str)
        return [d[k] for k in d]
```

Q50:

Implement pow(x, n).

**Solution:**

```
class Solution(object):
    def myPow(self, x, n):
        if n < 0:
            n = -n
            x = 1 / x
        ans = 1
        while n:
            if n & 1:
                ans *= x
            x *= x
            n >>= 1
        return ans
```