1.两数之和

```python
### 双指针+排序 复杂度nlogn
class Solution:
    def twoSum(self, nums, target) :
        left= 0
        right = len(nums) - 1
        sorted_id = sorted(range(len(nums)),key = lambda x:nums[x])
        while right > left:
            if nums[sorted_id[left]] + nums[sorted_id[right]] == target:
                return sorted_id[left],sorted_id[right]
            elif nums[sorted_id[left]] + nums[sorted_id[right]] > target:
                right -=1
            else:
                left +=1


### 哈希表 时间复杂度N 空间复杂度N
class Solution:
    def twoSum(self, nums, target) :
        dict1 = {}
        for index,_ in enumerate(nums):
            if target - _ in dict1:
                return dict1[target - _],index
            dict1[_] = index
```

15.三数之和(有要求不能重复的三数)

```
###自己写的 能过 复杂应该是N**2
### 感觉是运气过的 现在有点理不清楚了
### 不看这个
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        res = []
        if not nums or len(nums)<3:
            return []
        nums.sort()
        for left in range(len(nums)-2):
            if left != 0 and nums[left-1] == nums[left]:
                continue
            tmp_left = nums[left]
            if tmp_left > 0:### 能快进一些
                return res
            mid = left+1
            right = len(nums)-1
            while right > mid:
                if mid != (left+1) and nums[mid-1] == nums[mid]:
                    mid +=1
                    continue
                if right != len(nums)-1 and nums[right] == nums[right+1]:
                    right -=1
                    continue
                if tmp_left + nums[mid] + nums[right] == 0:
                    res.append([tmp_left,nums[mid],nums[right]])
                    mid +=1
                elif tmp_left + nums[mid] + nums[right] >0:
                    right -=1
                else:
                    mid +=1
        return res

### 为啥还没两年前写的快捏
class Solution:
    def threeSum(self, nums):
        c = []
        nums.sort()
        for i in range(0,len(nums)-2):
            if i==0 or nums[i]>nums[i-1]:
                l = i+1
                r = len(nums)-1
                while r>l:
                    if nums[i] + nums[l] + nums[r] == 0:
                        c.append([nums[i],nums[l],nums[r]])
                        l+=1
                        r-=1
                        while r>l and nums[r] ==nums[r+1]:
                            r-=1
                        while r>l and nums[l] == nums[l-1]:
                            l +=1
                elif nums[i] + nums[l] + nums[r] <0:
                    l +=1
                else:
                    r -=1
        return c

### 按照这个思路再重新写
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        res = []
        nums.sort()
        for left in range(len(nums)-2):
            if left != 0 and nums[left-1] == nums[left]:
                continue
            ### 加两个条件加快
            if nums[left]+nums[left+1] + nums[left+2] >0:
                return res
            if nums[left] + nums[-1] + nums[-2] <0:
                continue
            mid = left+1
            right = len(nums)-1
            while right > mid:
                tmp = nums[left] + nums[mid] + nums[right]
                if tmp == 0:
                    res.append([nums[left],nums[mid],nums[right]])
                    right -=1
                    mid +=1
                    ### 如果满足条件的情况下 mid 和 right
                    ### 只有都在下一个新的位置才会触发下一次
                    while right > mid and nums[right] == nums[right+1]:
                        right -=1
                    while right > mid and nums[mid] == nums[mid-1]:
                        mid +=1
                elif tmp>0:
                    right -=1
                    while right > mid and nums[right] == nums[right+1]:
                        right -=1
                else:
                    mid +=1
                    while right > mid and nums[mid] == nums[mid-1]:
                        mid +=1
        return res
```

633.平方数之和

```
#### 可以相等  双指针法要想到确定最大值
import math
class Solution:
    def judgeSquareSum(self, c: int) -> bool:
        right = int(math.sqrt(c))+1
        left = 0
        while right >= left:
            if right**2 + left**2 == c:
                return True
            elif  right**2 + left**2 > c:
                right -=1
            else:
                left +=1
        return False
```

345.反转字符串中的元音字母

```
#### 双指针法
class Solution:
    def reverseVowels(self, s):
        yuanyin = 'aeiouAEIOU'
        set1 = set(list(yuanyin))
        s = list(s)
        left = 0
        right = len(s)-1
        while right > left:
            if s[right] not in set1:
                right -=1
            if s[left] not in set1:
                left +=1
            if s[right] in set1 and s[left] in set1:
                s[right],s[left] = s[left],s[right]
                right -=1
                left +=1
        return ''.join(s)
```

680.验证回文字符串2

```
###自己写的 有点慢
class Solution:
    def validPalindrome(self, s):
        s = list(s)
        left = 0
        right = len(s)-1
        mark = 0
        mark_1 = True
        mark_2 = True
        while right > left:
            if s[right] == s[left]:
                right -=1
                left +=1
            elif s[right] == s[left+1]:
                left +=1
                mark +=1
            elif s[right-1] == s[left]:
                right -=1
                mark +=1
            else:
                mark_1 = False
                break
            if mark >=2:
                mark_1 = False
                break
        if mark_1:
            return True
        mark = 0
        left = 0
        right = len(s)-1
        while right > left:
            if s[right] == s[left]:
                right -=1
                left +=1
            elif  s[right-1] == s[left]:
                right -=1
                mark +=1
            elif s[right] == s[left+1]:
                left +=1
                mark +=1
            else:
                return False
            if mark >=2:
                return False
        return True
### 遇到不一样删除左边或者右边的 然后判断剩下的是不是回文字符串

class Solution:
    def validPalindrome(self, s):
        left = 0
        right = len(s) -1
```

```python
        if s == s[::-1]:
            return True
        while right > left:
            if s[right] == s[left]:
                left +=1
                right -=1
            elif s[right-1] == s[left] or s[left+1] == s[right]:
                a = s[left:(right)]
                b = s[(left+1):(right+1)]
                return a==a[::-1] or b==b[::-1]
            else:
                return False
```

88.合并两个有序数组

```python
### 双指针
## 思路的重点一个是从后往前确定两组中该用哪个数字
## 从前往后需要把Num1的空间腾出来
## 另一个是结束条件以第二个数组全都插入进去为止
class Solution:
    def merge(self, nums1, m, nums2, n):
        """
        Do not return anything, modify nums1 in-place instead.
        """
        if m == 0:
            nums1[:] = nums2[:]
        if n != 0:

            ind1 = m-1
            ind2 = n-1
            mark_pos = m+n-1
            while mark_pos>=0:
                if ind1<0:
                    nums1[:ind2+1] = nums2[:ind2+1]
                    break
                if ind2 <0:
                    break
                #print(mark_pos)
                if nums1[ind1] > nums2[ind2]:
                    nums1[mark_pos] = nums1[ind1]
                    ind1 -=1
                    mark_pos -=1
                else:
                    nums1[mark_pos] = nums2[ind2]
                    ind2 -=1
                    mark_pos -=1
```

141.环形链表

```
###空间复杂度O (N)
###时间复杂度O (N)
class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        # if head is None:
        #     return False
        hash_set = set()
        while head:
            if head in hash_set:
                return True
            if head.next is None:
                return False
            hash_set.add(head)
            head = head.next
### 快慢指针法
class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        fast_node = head
        slow_node = head
        while True:
            if fast_node is None or fast_node.next is None:
                return False
            fast_node = fast_node.next.next
            slow_node = slow_node.next
            if fast_node == slow_node:
                return True
```

142.环形链表

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def detectCycle(self, head: ListNode) -> ListNode:
        slow,fast = head,head
        while True:
            if fast is None or fast.next is None:
                return
            fast = fast.next.next
            slow = slow.next
            if fast == slow:
                break
        fast = head
        while fast != slow:
            fast = fast.next
            slow = slow.next
        return fast
```

删除链表的倒数第N个节点

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
    #### 建立一个伪头结点
        dummy = ListNode(0,head)
        fast,slow = dummy,dummy
        for _ in range(n):
            fast = fast.next
        while fast.next is not None:
            fast = fast.next
            slow = slow.next
        slow.next = slow.next.next
        return dummy.next
```

524.通过删除字母匹配到字典里的最长单词

```
###自己的傻吊做法
### 时间复杂度  O(N*M)

### 从答案里学到一些trick
### d.sort(key = lambda x: (-len(x), x)) 用于一个降序 一个升序
### str.find('a',1) 从1之后找a(包括1)
class Solution:
    def findLongestWord(self, s, d):
        work_list = []
        #max_len = 0
        for word in d:
            left_d = 0
            right_d = len(word)-1
            n = len(word)
            mark = 0
            left = 0
            right = len(s) - 1
            while right >= left:
                if right_d>=left_d and word[right_d] == s[right]:
                    right_d -=1
                    #right -=1
                    mark+=1
                if right_d>=left_d and word[left_d] == s[left]:
                    left_d +=1
                    #left +=1
                    mark+=1
                #print(mark)
                left+=1
                right-=1
                if mark >= n:
                    work_list.append(word)
        if len(work_list) == 0:
            return ''
        else:
            len_max = sorted([len(_) for _ in work_list])[-1]
            return sorted([_ for _ in work_list if len(_) == len_max])[0]
```

215.数组中的第K个最大元素

```
### 待补充堆排方法
### 快排思路 复杂度O (N)  空间O (1)
### 转化为第n-k+1小问题
### key采用中间作为初始点
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        def partition(nums,k):
            i = 0
            j = len(nums)-1
            mid = (i + j)//2
            nums[i],nums[mid] = nums[mid],nums[i]
            key = nums[i]
            while j>i:
                while j>i and nums[j] > key:
                    j -=1
                nums[i] = nums[j]
                while j>i and nums[i] <= key:
                    i+=1
                nums[j] = nums[i]
            nums[i] = key
            if k== i+1:
                return nums[i:i+1],1
            elif k<i+1:
                return nums[:i],k
            else:
                return nums[i+1:],k-(i+1)
        k = len(nums)+1-k
        while len(nums)>1:
            ##print(nums,k)
            nums,k = partition(nums,k)
        return nums[0]

class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:

        def topk(nums,left,right,k):
            #print(k)
            ind= partition(nums,left,right)
            if ind - left >= k:
                return topk(nums,left,ind-1,k)
            elif (ind-left) ==k-1:
                return nums[ind]
            else:
                return topk(nums,ind+1,right,k - (ind-left+1))



        def partition(nums,left,right):
            mid = (left+right)//2
            nums[left],nums[mid] = nums[mid],nums[left]
            key = nums[left]
            while right>left:
                while right>left and nums[right]>key:
                    right -=1
                nums[left] = nums[right]
                while right>left and nums[left]<=key:
                    left+=1
                nums[right] = nums[left]
            nums[left] = key
            return left
        return topk(nums,0,len(nums)-1,len(nums)-k+1)

#### 堆排思想
#### 复杂度Nlog(K)
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        def min_heapify(heap,heap_size,root):
            left = 2 * root + 1
            right = 2 * root + 2
            min_node = root
            if left < heap_size and heap[left] < heap[min_node]:
                min_node = left
            if right < heap_size and heap[right] < heap[min_node]:
                min_node = right
            if min_node != root:
                heap[root],heap[min_node] = heap[min_node],heap[root]
                min_heapify(heap,heap_size,min_node)
        def create_heap(heap,heap_size):
            for i in range((heap_size-2)//2,-1,-1):
                min_heapify(heap,heap_size,i)
        create_heap(nums,k)
        for i in range(k,len(nums)):
            if nums[i] > nums[0]:
                nums[i],nums[0] = nums[0],nums[i]
                min_heapify(nums,k,0)
        return nums[0]
### unique数小的话 用
```

347.前K个高频元素

```
### 哈希存储键值对
### 转化为Topk问题
### 哈希 时间复杂度O (N)  空间O (N)
### Topk 快排 O (N)
class Solution:
    def topKFrequent(self,nums, k):
        def partition(nums,k):
            i = 0
            j = len(nums) - 1
            mid = (i+j)//2
            key = nums[mid]
            nums[i],nums[mid] = nums[mid],nums[i]
            while j>i :
                while j>i and nums[j] < key:
                    j-=1
                nums[i] = nums[j]
                while j>i and nums[i] >= key:
                    i+=1
                nums[j] = nums[i]
            nums[i] = key
            if i == k-1:
                return nums[i:i+1],1
            elif k < i + 1:
                return nums[:i],k
            else:
                return nums[(i+1):],k-i-1

        def topk(nums,k):
            while len(nums)>1:
                nums,k= partition(nums,k)
            return nums[0]
        dict1 ={}
        for _ in nums:
            if _ in dict1:
                dict1[_] +=1
            else:
                dict1[_] = 1
        threshold = topk(list(dict1.values()),k)
        return  [a for a,b in dict1.items() if b>=threshold]
```

75.颜色分类

```
#### 快排思路
class Solution:
    def sortColors(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        def quick_sort(nums,start,end):
            if start >= end:
                return
            i = start
            j = end
            key = nums[start]
            while j>i:
                while j>i and nums[j] > key:
                    j-=1
                nums[i] = nums[j]
                while j>i and nums[i] <= key:
                    i+=1
                nums[j] = nums[i]
            nums[i] = key
            quick_sort(nums,start,i-1)
            quick_sort(nums,i+1,end)
        quick_sort(nums,0,len(nums)-1)
#### 指针思路
### https://leetcode-cn.com/problems/sort-colors/solution/yan-se-fen-lei-by-leetcode/
class Solution:
    def sortColors(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        p0 = 0
        curr = 0
        p2 = len(nums)-1
        while curr<=p2:
            if nums[curr] == 0:
                nums[curr],nums[p0] = nums[p0],nums[curr]
                p0+=1
                curr +=1
            elif nums[curr] == 2:
                nums[curr],nums[p2] = nums[p2],nums[curr]
                p2-=1
            else:
                curr+=1
```

455.分发饼干

```
### 贪心算法
class Solution:
    def findContentChildren(self, g: List[int], s: List[int]) -> int:
        g.sort()
        s.sort()
        g_index = 0
        s_index = 0
        while g_index<= len(g)-1 and s_index <= len(s)-1:
            if g[g_index]<= s[s_index]:
                s_index+=1
                g_index+=1
            else:
                s_index+=1
        return g_index
```

435.无重叠区间

```
###贪心算法
### https://leetcode-cn.com/problems/non-overlapping-intervals/solution/tan-xin-suan-fa-zhi-qu-j
### https://leetcode-cn.com/problems/non-overlapping-intervals/solution/wu-zhong-die-qu-jian-by-
class Solution:
    def eraseOverlapIntervals(self, intervals: List[List[int]]) -> int:
        intervals.sort(key = lambda x:x[0])
        last_interval = None
        drop_num = 0
        for _ in intervals:
            print(_)
            if not last_interval:
                last_interval = _
                continue
            if _[0]>=last_interval[1]:
                last_interval = _
            elif _[1] < last_interval[1]:
                drop_num+=1
                last_interval = _
            else:
                drop_num +=1
            print(_,last_interval,drop_num)
        return drop_num
```

56.合并区间

```
###和上题类似
### 时间复杂度O(NLOGN) 排序复杂度
class Solution:
    def merge(self, intervals: List[List[int]]) -> List[List[int]]:
        intervals.sort(key = lambda x:x[0])
        index = 0
        if len(intervals) <=1:
            return intervals
        while index <= len(intervals)-2:
            first_list = intervals[index]
            #print(first_list)
            second_list = intervals[index+1]
            #print(second_list)
            if second_list[0] > first_list[1]:
                index+=1
            else:
                intervals[index] = [first_list[0],max(first_list[1],second_list[1])]
                intervals.pop(index+1)
        return intervals
```

最大子序和

```
### https://leetcode-cn.com/problems/maximum-subarray/solution/zui-da-zi-xu-he-cshi-xian-si-chor
### 贪心算法
# 若当前指针所指元素之前的和小于0，说明它对子序列和没贡献，我们就丢弃当前元素之前的数列。
# max保存全局最大子序列和；subMax保存当前指针所指元素之前的子序列和。每次将max与subMax进行比较，将较大的

class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        sub_sum = nums[0]###不从0开始 万一都是负数 从0开始需要提前判断sub_num<0
        max_sum = nums[0]
        for _ in nums[1:]:
            if sub_sum < 0:
                sub_sum = 0
            sub_sum +=_
            if sub_sum > max_sum:
                max_sum = sub_sum
        return max_sum

### 动态规划
def maxSubArray(nums):
    max_num = nums[0]
    f_state = nums[0]
    for _ in nums[1:]:
        f_state = max(f_state+_,_)
        max_num = max(max_num,f_state)
    return max_num

### 分治法
### 后期再来补充
```

94.二叉树的前序遍历

```
### https://leetcode-cn.com/problems/binary-tree-preorder-traversal/solution/tu-jie-er-cha-shu-c


### 递归法
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def preorderTraversal(self, root: TreeNode) -> List[int]:
        res  =  []
        def dfs(root1):
            nonlocal res
            if root1 is None:
                return
            res.append(root1.val)
            dfs(root1.left)
            dfs(root1.right)
        dfs(root)
        return res

### 迭代法

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def preorderTraversal(self, root: TreeNode) -> List[int]:
        if root is None:
            return []
        cur,stack,res = root,[],[]
        while stack or cur:
            while cur:
                res.append(cur.val)
                stack.append(cur)
                cur = cur.left
            tmp = stack.pop()
            cur = tmp.right
        return res
```

144.二叉树的中序遍历

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def inorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        def dfs(root1):
            nonlocal res
            if root1 is None:
                return
            dfs(root1.left)
            res.append(root1.val)
            dfs(root1.right)
        dfs(root)
        return res
```

### 迭代法

```python
class Solution:
    def inorderTraversal(self, root: TreeNode) -> List[int]:
        if root is None:
            return []
        cur,stack,res = root,[],[]
        while stack or cur:
            while cur:
                stack.append(cur)
                cur = cur.left
            tmp = stack.pop()
            res.append(tmp.val)
            cur = tmp.right
        return res
```

145.二叉树的后序遍历

```
class Solution:
    def postorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        def dfs(root1):
            nonlocal res
            if root1 is None:
                return
            dfs(root1.left)
            dfs(root1.right)
            res.append(root1.val)
        dfs(root)
        return res
```

迭代

```
class Solution:
    def postorderTraversal(self, root: TreeNode) -> List[int]:
        if root is None:
            return
        cur,stack,res = root,[],[]
        while stack or cur:
            while cur:
                res.append(cur.val)
                stack.append(cur)
                cur = cur.right
            tmp = stack.pop()
            cur = tmp.left
        return res[::-1]
```

###迭代2

```
class Solution:
    def postorderTraversal(self, root: TreeNode) -> List[int]:
        if root is None:
            return []
        stack,res = [(0,root)],[]
        while stack:
            flag,node = stack.pop()
            if node is None:
                continue
            if flag ==1:###右结点返回的root才能进入res
                res.append(node.val)
            else:
                stack.append((1,node))
                stack.append((0,node.right))
                stack.append((0,node.left))
        return res
```

102.二叉树的层序遍历

```
###https://leetcode-cn.com/problems/binary-tree-level-order-traversal/solution/bfs-de-shi-yong-c
### 晚点再看
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        res = []
        if root is None:
            return []
        queue = [root]
        while queue:
            res_level = []
            size_level = len(queue)
            for _ in range(size_level):
                tmp = queue.pop(0)
                res_level.append(tmp.val)
                if tmp.left:
                    queue.append(tmp.left)
                if tmp.right:
                    queue.append(tmp.right)
            res.append(res_level)
        return res
```

剑指offer27.二叉树的镜像/226.翻转二叉树

```
#### 递归法

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def mirrorTree(self, root: TreeNode) -> TreeNode:
        if root is None:
            return
        root.left,root.right = root.right,root.left
        self.mirrorTree(root.left)
        self.mirrorTree(root.right)
        return root

#### 迭代 广度优先遍历 队列

# https://leetcode-cn.com/problems/er-cha-shu-de-jing-xiang-lcof/solution/dong-hua-yan-shi-liang

class Solution:
    def mirrorTree(self, root: TreeNode) -> TreeNode:
        if root is None:
            return []
        queue = [root]
        while queue:
            tmp = queue.pop(0)
            tmp.left,tmp.right = tmp.right,tmp.left
            if tmp.left is not None:
                queue.append(tmp.left)
            if tmp.right is not None:
                queue.append(tmp.right)
        return root
```

104.二叉树的最大深度

```
### 自己想的 层序遍历
class Solution:
    def maxDepth(self, root: TreeNode) -> int:
        if root is None:
            return 0
        level = 0
        queue = [root]
        while queue:
            level_size = len(queue)
            for _ in range(level_size):
                tmp = queue.pop(0)
                if tmp.left:
                    queue.append(tmp.left)
                if tmp.right:
                    queue.append(tmp.right)
            level+=1
        return level

### 递归
class Solution:
    def maxDepth(self, root: TreeNode) -> int:
        if root is None:
            return 0
        else:
            left_depth = self.maxDepth(root.left)
            right_depth = self.maxDepth(root.right)
        return max(left_depth+1,right_depth+1)
```

101.对称二叉树

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isSymmetric(self, root: TreeNode) -> bool:
        ### 对称二叉树定义
        ### 要么左右子节点为None
        ### 左右子节点相等 且 左子节点的右子节点 = 右子结点的左子节点 且 左右反过来
        if root is None:
            return True
        def issym(left,right):
            if left is None and right is None:
                return True
            if left is None or right is None:
                return False
            if left.val != right.val:
                return False
            return issym(left.left,right.right) and issym(left.right,right.left)
        return issym(root.left,root.right)
### 迭代法下次看
```

112.路径总和

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def hasPathSum(self, root: TreeNode, sum: int) -> bool:
        if root is None:
            return False
        if not (root.left or root.right):
            return root.val == sum
        # 这两个可以不要 包含在return的里面
        # if root.left is None:
        #     return self.hasPathSum(root.right,sum-root.val)
        # if root.right is None:
        #     return self.hasPathSum(root.left,sum - root.val)

        return self.hasPathSum(root.left,sum - root.val) or self.hasPathSum(root.right,sum - roc
```

263.丑数

```
### 只包含质因数2/3/5
class Solution:
    def isUgly(self, num: int) -> bool:
        if num ==1:
            return True
        if num < 1:
            return False
        if num%2 == 0:
            num = num/2
        elif num%3 ==0:
            num = num /3
        elif num%5 == 0:
            num = num/5
        else:
            return False
        return self.isUgly(num)

### 迭代
class Solution:
    def isUgly(self,num):
        if num < 1:
            return False
        while num%2 ==0:
            num = num/2
        while num%3 == 0 :
            num = num/3
        while num%5 ==0:
            num = num/5
        return num == 1
```

88.合并两个有序数组

```
### 先确定好长度
### 两边都从后面开始 双指针
### 就不影响了
class Solution:
    def merge(self, nums1, m, nums2, n):
        """
        Do not return anything, modify nums1 in-place instead.
        """
        p1 = m-1
        p2 = n-1
        p = m+n-1
        while p1>= 0 and p2>=0:
            if nums1[p1] > nums2[p2]:
                nums1[p] = nums1[p1]
                p1-=1
                p-=1
            else:
                nums1[p] = nums2[p2]
                p2-=1
                p-=1
        if p2 >=0:
            nums1[:p+1] = nums2[:p2+1]###改成p2+1也行
        return nums1
```

35. 搜索插入位置

```
### 查找左边插入位置 或者 左边界
### 具体分析得出来的结果
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        left,right = 0,len(nums)-1
        while right >= left:
            mid = (left + right)//2
            if nums[mid] >= target:
                right = mid - 1
            else:
                left = mid + 1
        return left

### 找右边插入位置
def bisep(nums,target):
    left,right = 0 ,len(nums)-1
    while right >= left:
        mid = (left + right)//2
        if nums[mid] <= target:
            left = mid +1
        else:
            right = mid -1
    return left###如果target在num里找右边位置return right

### 这样也行
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        left,right = 0,len(nums)-1
        while right >= left:
            mid = (left + right)//2
            if nums[mid] == target:
                return mid
            if nums[mid] > target:
                right = mid - 1
            else:
                left = mid + 1
        return left
```

1608.特征数组的特征值

```
### 二分讲解
### 基本二分
def search(num,target):
    left,right  = 0,len(nums)-1
    while right >= left:
        mid = (left+right)//2
        if nums[mid] == target:
            return mid
        elif nums[mid] > target:
            right = mid -1
        else:
            left = mid +1
    return -1
### 找左边界 --如果在里面就是最左边 不在里面就是插入位置
while right >= left:
    mid = (left + right)//2
    if nums[mid] >= target:
        right = mid -1
    else:
        left = mid +1
    return left
### 右边插入
    if nums[mid]<= target:
        left = mid +1
    else:
        right = mid -1
    return left
###  右边找位置
    return right
class Solution:
    def specialArray(self, nums: List[int]) -> int:
        n = len(nums)
        nums.sort()
        for _ in range(n+1):
            if n - self.bileft(nums,_) ==_:
                return _
        return -1
    def bileft(self,nums1,target):
        left,right = 0,len(nums1)-1
        while right>=left:
            mid = (left+right)//2
            if nums1[mid]>=target:
                right = mid -1
            else:
                left = mid +1
        return left
```

106.中序后序遍历构造二叉树

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def buildTree(self, inorder: List[int], postorder: List[int]) -> TreeNode:
        self.dic,self.po = {},postorder
        for a,b in enumerate(inorder):
            self.dic[b] = a
        return self.recur(len(inorder)-1,0,len(inorder)-1)

    def recur(self,post_root,in_left,in_right):
        if in_left > in_right:
            return None
        root = TreeNode(self.po[post_root])
        ind = self.dic[self.po[post_root]]
        root.left = self.recur(post_root-in_right+ind-1,in_left,ind-1)
        root.right = self.recur(post_root-1,ind+1,in_right)
        return root
```

46.全排列(不可以重复)

```python
### 回溯做法
### 第一次通过的做法
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        final_list= []
        def backtrack(path,choice):
            if len(choice) == 0:
                final_list.append(path[:])
                return
            for _ in choice[:]:
                choice.remove(_)
                path.append(_)
                backtrack(path[:], choice[:])
                choice.append(_)
                path.remove(_)
        backtrack([], nums[:])
        return final_list
### 看了答案
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        final_list= []
        def backtrack(path,choice):
            if len(choice) == 0:
                final_list.append(path[:])
                return
            for _ in range(len(choice)):
                ### 将选择从选择列表删除
                ### 路径加上选择
                ### 这两步直接在backtrack完成
                backtrack(path+[choice[_]], choice[:_] + choice[_+1:])
                ### 退回 撤销选择
                ### 路径删除
                ### 选择加回来
        backtrack([], nums[:])
        return final_list
```

47.全排列(可以重复)

```
#### 第一次做解法 不知道如何判断重复 只能用In了 巨慢 但是还是通过了
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        final_list = []
        def backtrack(path,choice):
            if len(choice) ==0 and path not in final_list:
                final_list.append(path)
                return
            for i in range(len(choice)):
                backtrack(path + [choice[i]], choice[:i]+choice[i+1:])
        backtrack([], nums)
        return final_list
##### 思考一下为什么会重复 在每次choice的时候 如果有两个数字一样 那么就一定会重复
#####
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        final_list = []
        def backtrack(path,choice):
            if len(choice) ==0:
                final_list.append(path)
                return
            tmp = set()
            for i in range(len(choice)):
                if choice[i] in tmp:
                    continue
                tmp.add(choice[i])
                backtrack(path + [choice[i]], choice[:i]+choice[i+1:])
        backtrack([], nums)
        return final_list
### 再或者 先将nums排序 如果nums[i] == nums[i-1]时候就剪纸
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        final_list = []
        nums.sort()
        def backtrack(path,choice):
            if len(choice) == 0:
                final_list.append(path)
            for i in range(len(choice)):
                if i>0 and choice[i] == choice[i-1]:
                    continue
                backtrack(path+[choice[i]], choice[:i]+choice[i+1:])
        backtrack([], nums)
        return final_list
```

54.螺旋矩阵

```python
### 顺时针 矩阵
### 设定u/d/l/r 边界
### https://leetcode-cn.com/problems/spiral-matrix/solution/cxiang-xi-ti-jie-by-youlookdelicious
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        if len(matrix) == 0:
            return []
        res = []
        u = 0
        d = len(matrix)-1
        l = 0
        r = len(matrix[0])-1
        while True:
            for _ in range(l,r+1):
                res.append(matrix[u][_])
            u +=1
            if u > d:
                break
            for _ in range(u,d+1):
                res.append(matrix[_][r])
            r-=1
            if l>r:
                break
            for _ in range(r,l-1,-1):
                res.append(matrix[d][_])
            d -=1
            if u > d:
                break
            for _ in range(d,u-1,-1):
                res.append(matrix[_][l])
            l +=1
            if l>r:
                break
        return res
### 或者是把走过的路标记为None
### 定义个方向 directions = [[0, 1], [1, 0], [0, -1], [-1, 0]]
### 每遇到一次None,加一个方向
```

59.螺旋矩阵2

```python
### 和54基本条件差不多
### 由于限定了最大值是n**2
### 所以While可以加条件
class Solution:
    def generateMatrix(self, n: int) -> List[List[int]]:
        matrix = [[0 for _ in range(n)] for _ in range(n)]  ### [[0]*n]*n 这个会有问题 会让list中
        u,d,l,r = 0,n-1,0,n-1
        num,tar = 1,n**2
        while num<=tar:
            #print(matrix)
            for _ in range(l,r+1):
                matrix[u][_] = num
                num +=1
            u+=1
            for _ in range(u,d+1):
                matrix[_][r] = num
                num +=1
            r-=1
            for _ in range(r,l-1,-1):
                matrix[d][_] = num
                num+=1
            d-=1
            for _ in range(d,u-1,-1):
                matrix[_][l] = num
                num+=1
            l+=1
        return matrix
```

885.螺旋矩阵3

```
### 自己写的哦
class Solution:
    def spiralMatrixIII(self, R: int, C: int, r0: int, c0: int) -> List[List[int]]:
        num = 1
        ind = 0
        step = 1
        u,d,l,r = 0,R-1,0,C-1
        res = [[r0,c0]]
        while num< R*C:
            dir = [[0,1],[1,0],[0,-1],[-1,0]]
            for _ in range(int(step)):
                dir_tmp = dir[ind]
                r0 += dir_tmp[0]
                c0 += dir_tmp[1]
                if r0>=u and r0<= d and c0 >=l and c0<=r:
                    res.append([r0,c0])
                    num +=1
            ind = (ind + 1)%4
            step += 0.5
        return res
####自己第二次写出了一个又不一样的版本
class Solution:
    def spiralMatrixIII(self, R: int, C: int, r0: int, c0: int) -> List[List[int]]:
        u = 0
        d = R-1
        l = 0
        r = C - 1
        step = 1
        ans = [[r0,c0]]
        while len(ans)<R*C:
            for i in range(step):
                c0+=1
                if r0>=u and r0 <=d and c0>=l and c0<=r:
                    ans.append([r0,c0])
            for i in range(step):
                r0+=1
                if r0>=u and r0 <=d and c0>=l and c0<=r:
                    ans.append([r0,c0])
            step+=1
            for i in range(step):
                c0-=1
                if r0>=u and r0 <=d and c0>=l and c0<=r:
                    ans.append([r0,c0])
            for i in range(step):
                r0-=1
                if r0>=u and r0 <=d and c0>=l and c0<=r:
                    ans.append([r0,c0])
            step +=1
        return ans
```

3.无重复字符的最长子串

```python
### 滑动窗口
class Solution:
    def lengthOfLongestSubstring(self,s):
        set_tmp= set()
        max_len = 0
        cur_len = 0
        left = 0
        for _ in s:
            while _ in set_tmp:
                set_tmp.remove(s[left])
                left +=1
                cur_len -=1
            set_tmp.add(_)
            cur_len +=1
            max_len = max(max_len,cur_len)
            #print(_,max_len)
        return max_len
### 自己模板写
class Solution:
    def lengthOfLongestSubstring(self,s):
        set_tmp = set()
        left,right = 0,0
        max_len = 0
        while right < len(s):
            if s[right] not in set_tmp:
                set_tmp.add(s[right])
            else:
                while s[right] in set_tmp:
                    set_tmp.remove(s[left])
                    left+=1
                set_tmp.add(s[right])
            max_len = max(max_len,right - left +1)
            right +=1
        return max_len
### 修改下模板(这题按模板写太丑陋了)
class Solution:
    def lengthOfLongestSubstring(self,s):
        set_tmp = set()
        left,right = 0,0
        max_len = 0
        while right < len(s):
            while s[right] in set_tmp:
                set_tmp.remove(s[left])
                left+=1
            set_tmp.add(s[right])
            max_len = max(max_len,right - left +1)
            right +=1
        return max_len
```

209.长度最小的子数组

```
### 每次先加最后一个
### 然后再往前删
class Solution:
    def minSubArrayLen(self, s: int, nums: List[int]) -> int:
        left,right = 0,0
        min_len = len(nums)+1
        sum1 = 0
        while right < len(nums):
            sum1 += nums[right]
            while sum1 >=s :
                min_len = min(min_len,right - left +1)
                sum1 -= nums[left]
                left +=1
            right +=1
        if min_len != len(nums)+1:
            return min_len
        else:
            return 0
```

1004 最大连续1的个数3

```
class Solution:
    def longestOnes(self, A: List[int], K: int) -> int:
        left,right = 0,0
        max_len = 0
        cnt_0 = 0
        while right < len(A):
            if A[right] ==0:
                cnt_0 +=1###第一步是往后推一个
            while cnt_0 >K:###这层条件是否是符合题目意思的条件得看找最大值还是最小值
                if A[left] ==0:
                    cnt_0 -=1
                left +=1
            max_len = max(max_len,right-left+1)
            right +=1
        return max_len
```

76.最小覆盖子串

```
###这题关键要维护一个n_s去探测是否满足条件
class Solution:
    def minWindow(self, s: str, t: str) -> str:
        left,right = 0,0
        n_s = len(s)
        n_t = len(t)
        min_len = n_s +1
        min_left = 0
        min_right = 0
        dict_t = dict()
        num_s = 0###标记含t中元素的个数
        #dict_s = {}
        for _ in t:
            if _ not in dict_t:
                dict_t[_] = 1
            else:
                dict_t[_] +=1
        while right < n_s:
            if s[right] in dict_t:
                if dict_t[s[right]] > 0:
                    num_s +=1
                dict_t[s[right]] -=1
            while num_s == n_t:
                if right-left +1 < min_len:
                    min_len = right - left +1
                    min_left =left
                    min_right = right
                if s[left] in dict_t:
                    if dict_t[s[left]] ==0:
                        num_s -=1
                    dict_t[s[left]] +=1
                left +=1
            right +=1
        return s[min_left:min_right+1] if min_len != n_s+1 else ''
```

42.接雨水

```
####解析看
#### https://leetcode-cn.com/problems/trapping-rain-water/solution/dong-tai-gui-hua-shuang-zhi-z
#### 看不懂看下面评论
class Solution:
    def trap(self, height: List[int]) -> int:
        #### 每个点上是否能装雨水取决于每个左右最高点的位置
        #### 最高点比它高就能装
        if len(height) ==0:
            return 0
        n = len(height)
        left,right = 0,n-1
        max_left,max_right = height[0],height[n-1]
        res = 0
        while right >=left:
            max_left = max(max_left,height[left])
            max_right = max(max_right,height[right])
            if max_left > max_right:
                res += max_right - height[right]
                right -=1
            else:
                res += max_left - height[left]
                left +=1
        return res
```

283.移动零

```
### 第一种 快慢指针
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        n = len(nums)
        i = 0
        for _ in nums:
            if _ !=0:
                nums[i] = _
                i+=1
        nums[i:] = [0]*(n-i)
### 第二种双指针
### left找为0的
### right找非0的
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        n = len(nums)
        left = 0
        right = 1
        while left < n and right < n :
            if nums[left] !=0:
                left +=1
                right+=1
            else:
                while right<n and nums[right] ==0:
                    right +=1
                if right !=n:
                    nums[left],nums[right] = nums[right],nums[left]
                left +=1
                right+=1
```

287.寻找重复数

```
#### 复杂度 nlogn
#### 不行 不能改变原数组
class Solution:
    def findDuplicate(self, nums: List[int]) -> int:
        nums.sort()
        for _ in range(len(nums)-1):
            if nums[_] == nums[_+1]:
                return nums[_]

#### 复杂度 nlogn 用二分
#### 关键在于 确定整数范围的题也能用二分
#### https://leetcode-cn.com/problems/find-the-duplicate-number/solution/er-fen-fa-si-lu-ji-dai-
#### 题解看这里
class Solution:
    def findDuplicate(self, nums: List[int]) -> int:
        left,right = 0,len(nums)-1
        while right > left:
            mid = (left + right)//2
            cnt = 0
            for _ in nums:
                if _ <=mid:
                    cnt +=1
            if cnt > mid:
                right = mid
            else:
                left = mid +1
        return left
```

2.两数相加

```
#### 自己写的 感觉有点丑陋
#### 别人的思路也是这样还行 就是我的代码稍微冗长了点
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:
        tmp1 = l1
        tmp2 = l2
        dummy_true = ListNode(0)
        dummy = dummy_true
        res = 0
        while tmp1 is not None and tmp2 is not None:
            tmp = tmp1.val +tmp2.val + res
            res = tmp//10
            val1 = tmp%10
            dummy.next = ListNode(val1)
            dummy = dummy.next
            tmp1 = tmp1.next
            tmp2 = tmp2.next
        if tmp1 is None:
            while tmp2 is not None:
                tmp = res + tmp2.val
                res = tmp//10
                val1 = tmp%10
                dummy.next = ListNode(val1)
                dummy = dummy.next
                tmp2 = tmp2.next
        else:
             while tmp1 is not None:
                tmp = res + tmp1.val
                res = tmp//10
                val1 = tmp%10
                dummy.next = ListNode(val1)
                dummy = dummy.next
                tmp1 = tmp1.next
        if res !=0:
            dummy.next = ListNode(res)
        return dummy_true.next
```

21.合并两个有序链表

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def mergeTwoLists(self, t1: ListNode, t2: ListNode) -> ListNode:
        dummy_head = dummy = ListNode(None)
        while t1 and t2:
            if t1.val > t2.val:
                dummy.next = ListNode(t2.val)
                dummy = dummy.next
                t2 = t2.next
            else:
                dummy.next = ListNode(t1.val)
                dummy = dummy.next
                t1 = t1.next
        if t1 is None:
            dummy.next = t2
        else:
            dummy.next = t1
        return dummy_head.next
```

206.反转链表

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        pre = None
        cur = head
        while cur:
            tmp = cur.next
            cur.next = pre
            pre = cur
            cur = tmp
        return pre
```

234.回文链表

```
### 方法1 复制到数组中 然后 双指针判断 双N
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def isPalindrome(self, head: ListNode) -> bool:
        list1  = []
        cur = head
        while cur:
            list1.append(cur.val)
            cur = cur.next
        left = 0
        right = len(list1) - 1
        while right > left:
            if list1[left] != list1[right]:
                return False
            left +=1
            right-=1
        return True


#### 要用到反转
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def isPalindrome(self, head: ListNode) -> bool:
        def find_mid(head):
            s,f = head,head
            while f.next and f.next.next:
                s = s.next
                f = f.next.next
            return s

        def reverse(head):
            pre = None
            cur = head
            while cur:
                tmp = cur.next
                cur.next = pre
                pre = cur
                cur = tmp
            return pre
        if not head:
            return True
        mid = find_mid(head)
        mid.next = reverse(mid.next)
        f = head
        s = mid.next
        res = True
        while f and s:
            if f.val!=s.val:
                res = False
                break
            f = f.next
            s = s.next

        mid.next = reverse(mid.next)
        return res
```

160.相交链表

```
class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
        pA = headA
        pB = headB
        while pA!=pB:
            pA = pA.next if pA else headB
            ####不是pB.next和pA.next 因为这样会在最后一直循环
            #### 不会变成两个None跳出来
            pB = pB.next if pB else headA
        return pA
```

148.排序链表

```
### 递归的归并排序
class Solution:
    def sortList(self, head: ListNode) -> ListNode:
        if not head or not head.next:
            return head
        f = s = head
        while f.next and f.next.next:
            f = f.next.next
            s = s.next
        mid = s.next
        s.next = None
        left = self.sortList(head)
        right = self.sortList(mid)
        return self.merge(left,right)

    def merge(self,a,b):
        dummy = h = ListNode(0)
        while a and b:
            if a.val <b.val:
                h.next = a
                a = a.next
            else:
                h.next = b
                b= b.next
            h = h.next
        if a:
            h.next = a
        if b:
            h.next = b
        return dummy.next
### 有迭代的还没看

### 快排思想
### 大致看下就好了
class Solution:
    def sortList(self, head: ListNode) -> ListNode:
        # 快速排序
        if not head: return None
        # small equal large 的缩写
        # 都指向相应链表的 head
        s = e = l = None
        target = head.val
        while head:
            nxt = head.next
            if head.val>target:
                head.next = l
                l = head
            elif head.val==target:
                head.next = e
                e = head
            else:
                head.next = s
                s = head
            head = nxt

        s = self.sortList(s)
        l = self.sortList(l)
        # 合并 3 个链表
        dummy = ListNode(0)
        cur = dummy # cur: 非 None 的尾节点
        # p: 下一个需要连接的节点
        for p in [s, e, l]:
            while p:
                cur.next = p
                p = p.next
                cur = cur.next
        return dummy.next
```

48.旋转图像

给定一个 n × n 的二维矩阵表示一个图像。

将图像顺时针旋转 90 度。

```
### 先转置后对称
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        if len(matrix) ==0:
            return []
        m = len(matrix)
        n = len(matrix[0])
        for i in range(m):
            for j in range(i,n):
                matrix[i][j],matrix[j][i] = matrix[j][i],matrix[i][j]
        for i in range(m):
            for j in range(n//2):
                matrix[i][j],matrix[i][n-1-j] = matrix[i][n-1-j],matrix[i][j]

####
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        if len(matrix)==0:
            return []
        pos1 = 0
        pos2 = len(matrix)-1
        while pos1 < pos2:
            add1 = 0
            while pos2>pos1+add1:
                tmp = matrix[pos1][pos1+add1]
                ### 1号位被4号位替换
                matrix[pos1][pos1+add1] = matrix[pos2-add1][pos1]
                ### 4号位被3号位替换
                matrix[pos2-add1][pos1] = matrix[pos2][pos2-add1]
                ### 3号位被2号位替换
                matrix[pos2][pos2-add1] = matrix[pos1+add1][pos2]
                ### 2号位被tmp替换
                matrix[pos1+add1][pos2] = tmp
                add1 +=1
            pos1+=1
            pos2-=1
```

560.和为K的子数组

```
#### 暴力遍历法O N^2
#### J代表以J结尾
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        ans = 0
        for j in range(len(nums)):
            sum1 = 0
            for i in range(j,-1,-1):
                sum1 +=nums[i]
                if sum1 ==k:
                    ans +=1
        return ans

#### 前缀和
#### 前j累加 减去 前i累加 就等于i+1到j
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        dict1 = {}
        dict1[0] = 1
        sum1 = 0
        ans = 0
        for num in nums:
            sum1 += num
            if sum1 - k in dict1:
                ans +=dict1[sum1-k]
            if sum1 in dict1:
                dict1[sum1]+=1
            else:
                dict1[sum1] = 1
        return ans
```

11.盛最多水的容器

给你 n 个非负整数 a1，a2，...，an，每个数代表坐标中的一个点 (i, ai) 。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, ai) 和 (i, 0) 。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

```
class Solution:
    def maxArea(self, height: List[int]) -> int:
        ####每次只有把长线往里缩才有可能变大
        left = 0
        right = len(height)-1
        max_area = 0
        while right > left:
            if height[right]>height[left]:
                max_area = max(max_area,(right-left)*height[left])
                left +=1
            else:
                max_area = max(max_area,(right-left)*height[right])
                right -=1
        return max_area
```

55.跳跃游戏

给定一个非负整数数组，你最初位于数组的第一个位置。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

判断你是否能够到达最后一个位置。

```
class Solution:
    def canJump(self, nums: List[int]) -> bool:
        #### 贪心算法
        #### 每一步都跳最远
        max_index = 0
        for ind,jump in enumerate(nums):
            if max_index >= len(nums)-1:
                return True
            if max_index < ind:
                return False
            else:
                max_index = max(max_index,ind+jump)
```

128.最长连续序列

给定一个未排序的整数数组 nums ，找出数字连续的最长序列（不要求序列元素在原数组中连续）的长度。

进阶：你可以设计并实现时间复杂度为 O(n) 的解决方案吗？

```
#### https://leetcode-cn.com/problems/longest-consecutive-sequence/solution/zui-chang-lian-xu-xu
class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:
        if len(nums) ==0:
            return 0
        num_set = set(nums)
        len_c_max = 1
        for num in num_set:
            len_c = 1
            if num - 1 in num_set:
                continue
            while num +1 in num_set:
                num +=1
                len_c+=1
            len_c_max = max(len_c,len_c_max)
        return len_c_max
```

121.买卖股票的最佳时机

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。

如果你最多只允许完成一笔交易（即买入和卖出一支股票一次），设计一个算法来计算你所能获取的最大利润。

注意：你不能在买入股票前卖出股票。

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        ### 动态规划 前i天的最大收益 = max(前i-1天的最大收益,第i天的价格-前i-1天的最小价格)
        max_profit = 0
        min_price = float('inf')
        for num in prices:
            max_profit = max(max_profit,num - min_price)
            min_price = min(min_price,num)
        return max_profit
```

238.除自身以外数组的乘积

给你一个长度为 n 的整数数组 nums，其中 n > 1，返回输出数组 output ，其中 output[i] 等于 nums 中除 nums[i] 之外其余各元素的乘积。

```
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        ### 构造left right分别表示i位置左边乘积和右边乘积
        ### 因为要常数空间 不用right空间 直接把Left空间当做输出
        left = [1 for _ in nums]
        for ind in range(len(nums)):
            if ind ==0:
                continue
            else:
                left[ind] = nums[ind-1]*left[ind-1]
        R = 1
        for ind in range(len(nums)-1,-1,-1):
            left[ind] = left[ind]*R
            R *= nums[ind]
        return left
```

78.子集

给定一组不含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。

```
### 击败10%的自己的回溯
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        nums.sort()
        ans = []
        def backtrack(path,choice,length):
            if len(path) == length:
                ans.append(path)
            for i in range(len(choice)):
                backtrack(path+[choice[i]],choice[i+1:],length)
        for l in range(len(nums)+1):
            backtrack([],nums,l)
        return ans

####
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        ### 遇到一个数就把所有子集加上该数组成新的子集，遍历完毕即是所有子集
        ans = [[]]
        for num in nums:
            ans += [[num] + i for i in ans]
        return ans
#### 改进版的回溯
#### 原来好像不用考虑长度
#### 只要往后加就行了
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        ans = []
        def backtrack(path,choice):
            ans.append(path)
            for i in range(len(choice)):
                backtrack(path+[choice[i]],choice[i+1:])
        backtrack([],nums)
        return ans
```

448. 找到所有数组中消失的数字

给定一个范围在 1 ≤ a[i] ≤ n (n = 数组大小) 的 整型数组，数组中的元素一些出现了两次，另一些只出现一次。

找到所有在 [1, n] 范围之间没有出现在数组中的数字。

您能在不使用额外空间且时间复杂度为O(n)的情况下完成这个任务吗? 你可以假定返回的数组不算在额外空间内。

```python
class Solution:
    def findDisappearedNumbers(self, nums: List[int]) -> List[int]:
        #### 可以用每个位置分别对应1-n
        #### 用负号标记出现过的数字
        for value in nums:
            if nums[abs(value)-1]>0:
                nums[abs(value)-1] *=-1
        ans = []
        for ind,num in enumerate(nums):
            if num>0:
                ans.append(ind+1)
        return ans
```

581.最短无序连续子数组

给定一个整数数组，你需要寻找一个连续的子数组，如果对这个子数组进行升序排序，那么整个数组都会变为升序排序。

你找到的子数组应是最短的，请输出它的长度。

```
### 排序 找到不一致的位置
class Solution:
    def findUnsortedSubarray(self, nums: List[int]) -> int:
        nums_copy = nums.copy()
        nums_copy.sort()
        for left in range(len(nums)):
            if nums[left]!=nums_copy[left]:
                break
        for right in range(len(nums)-1,-1,-1):
            if nums[right]!=nums_copy[right]:
                break
        return right - left +1 if right>left else 0

### https://zhuanlan.zhihu.com/p/161725943
class Solution:
    def findUnsortedSubarray(self, nums: List[int]) -> int:
        #### 从左到右 维护一个Max 如果num[i]<MAX i需要调整
        #### 同理，从右到左循环，记录最小值为 min，若 nums[i] > min，则表明位置 i 需要调整，循环结束，
        max_value = nums[0]
        min_value = nums[-1]
        end = 0
        start = len(nums)-1
        for max_ind in range(len(nums)):
            if nums[max_ind]<max_value:
                end = max_ind
            else:
                max_value = nums[max_ind]
        for min_ind in range(len(nums)-1,-1,-1):
            if nums[min_ind] > min_value:
                start = min_ind
            else:
                min_value = nums[min_ind]
        return end - start +1 if end > start else 0
```

5. 最长回文子串

给定一个字符串 s，找到 s 中最长的回文子串。你可以假设 s 的最大长度为 1000。

```
输入: "babad"
输出: "bab"
注意: "aba" 也是一个有效答案。
```

```
####动态规划
#### 复杂度都是O(N方)
class Solution:
    def longestPalindrome(self, s: str) -> str:
        ### dp[i][j]表示[i,j]是回文子串
        ### dp[i][j] = dp[i+1][j-1] and (s[i]==s[j])
        ### j-1-(i+1)+1<2是停止条件 j-i<3 j-i+1<4 如果ij只有长度3 只需要判断s[i]==s[j]
        n = len(s)
        dp = [[False]*n for _ in range(n)]
        max_len = 1
        max_start = 0
        for i in range(n):
            dp[i][i] = True
        for j in range(1,n):
            for i in range(0,j):
                if j-i<3:
                    dp[i][j] = s[i]==s[j]
                else:
                    dp[i][j] = s[i]==s[j] and (dp[i+1][j-1])
                if dp[i][j]:
                    if j-i+1 >max_len:
                        max_len = j-i+1
                        max_start = i
        return s[max_start:(max_len+max_start)]

#### 中心扩展方法
#### 只要从回文字符串中心往外扩展就能确定一个中心出来的最长子串 复杂度为N
#### 遍历N个 复杂度N方
class Solution:
    def longestPalindrome(self, s: str) -> str:
        if len(s)==1:
            return s
        #### 中心扩散方法
        max_len = 0
        for _ in range(len(s)-1):
            max_len1,max_start1 = self.findl(s,_,_)
            max_len2,max_start2 = self.findl(s,_,_+1)
            if max_len1>max_len:
                max_len = max_len1
                max_start = max_start1
            if max_len2>max_len:
                max_len = max_len2
                max_start = max_start2
        return s[max_start:(max_start+max_len)]

    def findl(self,s,left,right):
        ###输入的时候right要大于等于left
        while left>=0 and right < len(s) and s[left]==s[right]:
            left -=1
            right+=1
        return right-left-1,left+1
```

17.电话号码的字母组合

给定一个仅包含数字 2-9 的字符串，返回所有它能表示的字母组合。

给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。

```
输入："23"
输出：["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
```

```
### 自己写的
### 别人写的回头再看看
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        if len(digits)==0:
            return []
        ans = []
        n = len(digits)
        dict1 = {
            2:['a','b','c'],
            3:['d','e','f'],
            4:['g','h','i'],
            5:['j','k','l'],
            6:['m','n','o'],
            7:['p','q','r','s'],
            8:['t','u','v'],
            9:['w','x','y','z']
        }
        def backtrack(path,choice):
            if len(path)==n:
                ans.append(path[:])
                return
            for _ in dict1[int(choice[0])]:
                backtrack(path+_,choice[1:])
        backtrack('',digits)
        return ans
### 队列的思想
### 有点像层序遍历
### 广度优先?
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        if len(digits)==0:
            return []
        ans = []
        n = len(digits)
        dict1 = {
            2:['a','b','c'],
            3:['d','e','f'],
            4:['g','h','i'],
            5:['j','k','l'],
            6:['m','n','o'],
            7:['p','q','r','s'],
            8:['t','u','v'],
            9:['w','x','y','z']
        }
        queue = ['']
        for digit in digits:
            for _ in range(len(queue)):
                tmp = queue.pop(0)
                for di in dict1[int(digit)]:
```

```
                queue.append(tmp + di)
        return queue
```

22.括号生成

数字 n 代表生成括号的对数，请你设计一个函数，用于能够生成所有可能的并且 有效的 括号组合。

```
示例
输入: n = 3
输出: [
       "((()))",
       "(()())",
       "(())()",
       "()(())",
       "()()()"
     ]

来源: 力扣 (LeetCode)


### 回溯
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        ans = []
        def backtrack(path,left,right):
            if len(path)==2*n:
                ans.append(path)
                return
            for _ in ['(',')']:
                tmp_left = left
                tmp_right = right
                if _ =='(':
                    tmp_left+=1
                else:
                    tmp_right+=1
                if tmp_left>n or tmp_right>n or tmp_left<tmp_right:
                    continue
                backtrack(path+_,tmp_left,tmp_right)
        backtrack('',0,0)
        return ans
```

79.单词搜索

给定一个二维网格和一个单词，找出该单词是否存在于网格中。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

```
board =
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]

给定 word = "ABCCED", 返回 true
给定 word = "SEE", 返回 true
给定 word = "ABCB", 返回 false
```

```
####这样是不对的 这样回溯没有状态重置
                if board[tmp_i][tmp_j] == word[n]:
                    tmp_mark = mark.copy()
                    tmp_mark[tmp_i][tmp_j] = 1
                    if backtrack(tmp_mark,n+1,tmp_i,tmp_j):
                        return True
            return False
####
#### 从一个位置可以回溯
####
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        row = len(board)
        col = len(board[0])
        for i in range(row):
            for j in range(col):
                if board[i][j] == word[0]:
                    if self.find_exist(board,word,i,j):
                        return True
        return False



    def find_exist(self,board,word,i,j):
        row = len(board)
        col = len(board[0])
        u = 0
        d = row -1
        l = 0
        r = col -1
        mark = [[0]*col for _ in range(row)]
        mark[i][j] = 1
        ####方向 右下左上
        dir_list = [[0,1],[1,0],[0,-1],[-1,0]]
        n = 1
        def backtrack(mark,n,i,j):
            if n==len(word):
                return True
            for dir in dir_list:
                tmp_i = i+dir[0]
                tmp_j = j+dir[1]
                if tmp_i<u or tmp_i>d or tmp_j<l or tmp_j>r or mark[tmp_i][tmp_j] ==1:
                    continue
                if board[tmp_i][tmp_j] == word[n]:
                    mark[tmp_i][tmp_j] = 1
                    if backtrack(mark,n+1,i+dir[0],j+dir[1]):
                        return True
                    else:
                         mark[tmp_i][tmp_j] = 0
            return False
        return backtrack(mark,n,i,j)
```

70.爬楼梯

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢?

注意:给定 n 是一个正整数。

```
输入: 2
输出: 2
解释: 有两种方法可以爬到楼顶。
1.  1 阶 + 1 阶
2.  2 阶
```

```
class Solution:
    def climbStairs(self, n: int) -> int:
        #### 这也太巧妙了吧
        #### f(x)代表爬到x阶梯需要多少次
        #### f(x) = f(x-1)+f(x-2) 因为最后一次肯定从倒数第一或者倒数第二跳上来

        if n==1:
            return 1
        p2 = 1
        p1 = 2
        p = 2
        num = 2
        while num<n:
            p = p1 + p2
            p2 = p1
            p1 = p
            num+=1
        return p
```

198.打家劫舍

你是一个专业的小偷,计划偷窃沿街的房屋。每间房内都藏有一定的现金,影响你偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统,如果两间相邻的房屋在同一晚上被小偷闯入,系统会自动报警。

给定一个代表每个房屋存放金额的非负整数数组,计算你 不触动警报装置的情况下 , 一夜之内能够偷窃到的最高金额。

```
输入：[1,2,3,1]
输出：4
解释：偷窃 1 号房屋 (金额 = 1) ，然后偷窃 3 号房屋 (金额 = 3)。
     偷窃到的最高金额 = 1 + 3 = 4 。
```

```
class Solution:
    def rob(self, nums: List[int]) -> int:
        ### 只有一间房 偷1
        ### 2 偷其中最大的
        ### k>2 要么偷 k+(k-2)中最大的
        ### 要么k-1之中最大的
        if len(nums)==0:
            return 0
        if len(nums)==1:
            return nums[0]
        if len(nums) ==2:
            return max(nums[0],nums[1])
        dp = [0]*3
        dp[0] = nums[0]
        dp[1] = max(nums[0],nums[1])
        n = len(nums)
        t = 2
        while t<n:
            dp[2] = max(dp[1],nums[t]+dp[0])
            dp[0] = dp[1]
            dp[1] = dp[2]
            t+=1
        return dp[2]
```

647.回文子串

给定一个字符串，你的任务是计算这个字符串中有多少个回文子串。

具有不同开始位置或结束位置的子串，即使是由相同的字符组成，也会被视作不同的子串。

```
class Solution:
    def countSubstrings(self, s: str) -> int:
        ### dp[i][j]表示[i,j]是回文子串
        ### dp[i][j] = dp[i+1][j-1] and (s[i]==s[j])
        ### j-1-(i+1)+1<2是停止条件 j-i<3 j-i+1<4 如果ij只有长度3 只需要判断s[i]==s[j]
        n = len(s)
        dp = [[False]*n for _ in range(n)]
        max_len = 1
        max_start = 0
        num = n
        for i in range(n):
            dp[i][i] = True
        for j in range(1,n):
            for i in range(0,j):
                if j-i<3:
                    dp[i][j] = s[i]==s[j]
                else:
                    dp[i][j] = s[i]==s[j] and (dp[i+1][j-1])
                if dp[i][j]:
                    num+=1

        return num

class Solution:
    def countSubstrings(self, s: str) -> int:
        #### 中心扩散算法
        max_len = 1
        ans = s[0]
        num = 0
        for _ in range(len(s)-1):
            num1 = self.find_limit(s,_,_)
            num2 = self.find_limit(s,_,_+1)
            num+=(num1+num2)
        return num +1##+1是因为最后一个单字母

    def find_limit(self,s,left,right):
        num = 0
        while left >=0 and right < len(s) and s[left]==s[right]:
            left -=1
            right+=1
            num+=1
        return num
```

136.只出现过一次的数字

给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

说明：

你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？

```
输入: [2,2,1]
输出: 1

输入: [4,1,2,1,2]
输出: 4
```

```python
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        ans = 0
        for _ in nums:
            ans^= _
        return ans
```

739.每日温度

请根据每日 气温 列表，重新生成一个列表。对应位置的输出为：要想观测到更高的气温，至少需要等待的天数。如果气温在这之后都不会升高，请在该位置用 0 来代替。

例如，给定一个列表 temperatures = [73, 74, 75, 71, 69, 72, 76, 73]，你的输出应该是 [1, 1, 4, 2, 1, 1, 0, 0]。

提示：气温 列表长度的范围是 [1, 30000]。每个气温的值的均为华氏度，都是在 [30, 100] 范围内的整数。

```python
class Solution:
    def dailyTemperatures(self, T: List[int]) -> List[int]:
        ####单调递减栈
        stack = []
        n = len(T)
        ans = [0]*n
        for ind,num in enumerate(T):
            while stack and num>stack[-1][1]:
                tmp1,tmp2 = stack.pop()
                ans[tmp1] = ind - tmp1
            stack.append((ind,num))
        return ans
```