算法

两数之和

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        dic = {}
        for index, num in enumerate(nums):
            cp = target-num
            if cp in dic:
                return [dic.get(cp),index]
            else:
                dic[num] = index
```

三数之和

```
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        if len(nums) < 3: return []

        nums = sorted(nums)
        res = []
        for i in range(0, len(nums)):

            if (i>0) and (nums[i] == nums[i-1]): continue

            begin, end = i + 1, len(nums) - 1
            while begin < end:
                if nums[i] + nums[begin] + nums[end] < 0:
                    begin = begin + 1
                elif nums[i] + nums[begin] + nums[end] > 0:
                    end = end - 1
                else:
                    if (begin>i+1)and(nums[begin] == nums[begin-1]):
                        begin = begin + 1
                    elif (end<len(nums)-1) and nums[end] == nums[end+1]:
                        end = end - 1
                    else:
                        res.append([nums[i], nums[begin], nums[end]])
                        begin = begin + 1
                        end = end - 1

            if nums[i] > 0: break
        return res
```

TOPK

```
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        nums = [0]+nums
        def BuildMaxHead(A, l):#建立一个大顶堆
            for ii in range(l//2,0,-1):
                HeadAdjust(A,ii,l)

        def HeadAdjust(A, k, l):#将数组A的k开始到l的数组调整为大顶堆
            A[0] = A[k] #A[0]存放根的值
            i = 2*k #i指向k指向的节点的左孩子
            while i<=l:
                if i == l:
                    pass
                elif A[i+1] > A[i]:#i指向两个孩子中较大的那一个
                    i = i + 1
                if A[0]>A[i]: break
                else:
                    A[k] = A[i]
                    k = i
                i = i*2
            A[k] = A[0]

        n = len(nums)-1
        BuildMaxHead(nums,n)
        #nums = [3,6,5,4,3,2,1]
        for i in range(1,k+1):
            nums[1],nums[n-i+1] = nums[n-i+1],nums[1]
            HeadAdjust(nums,1,n-i)

        return nums[n-i+1]
```

字符串的排列

```
class Solution:
    def checkInclusion(self, s1: str, s2: str) -> bool:
        if len(s1)>len(s2): return False
        s1_L = len(s1)
        for i in range(0,len(s2)-s1_L+1):
            if sorted(s2[i:i+s1_L]) == sorted(s1):
                return True
        return False
```

链表归并排序

```
def sortList(head):
    #1. 分割
    #2. 归并

    if head == None or head.next == None:
        return head

    slow = head
    fast = head.next

    while fast and fast.next:
        fast = fast.next.next
        slow = slow.next

    mid = slow.next
    slow.next = None

    #至此 将一链表分割成head和mid

    left = sortList(head)
    right = sortList(mid)

    h = res = ListNode(0)

    while left and right :
        if left.val < right.val:
            h.next = left
            h = h.next
            #h.next = None
            left = left.next
        else:
            h.next = right
            h = h.next
            #h.next = None
            right = right.next

    if left:
        h.next = left
    elif right:
        h.next = right

    return res.next
```

相交链表

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

class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
        if headA == None or headB == None:
            return None
        boy = headA
        girl = headB
        flag1 = 0
        flag2 = 0
        while boy != girl:
            if boy.next:
                boy = boy.next
            else:
                if flag1 == 1:
                    return None
                else:
                    boy = headB
                    flag1 = 1

            if girl.next:
                girl = girl.next
            else:
                if flag2 == 1:
                    return None
                else:
                    girl = headA
                    flag2 = 1

        return boy
```

环形链表

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

class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        if head == None:
            return False
        if head.next == None:
            return False
        if head.next.next == None:
            return False

        slow = head
        quick = head.next

        while not (quick.next == None or quick.next.next == None):

            if slow == quick:
                return True
            else:
                slow = slow.next
                quick = quick.next.next
        return False
```

无重复字符的最长子串

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        ans = ''
        tmp = ''
        for i in s:
            if i not in tmp:
                tmp = tmp + i
            else:
                tmp = tmp[tmp.index(i) + 1:]
                tmp = tmp + i
            if len(tmp) > len(ans):
                ans = tmp
        return len(ans)
```

最长公共前缀

```
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        if not strs:
            return ''

        MinStr = min(strs)
        MaxStr = max(strs)

        res = ''
        for index, w in enumerate(MinStr):
            if w == MaxStr[index]:
                res = res + w
            else:
                break
        return res
```

字符串的翻转

```
class Solution:
    # def reverseWords(self, s: str) -> str:
    #     s = s.split()
    #     s = s[::-1]
    #     res = ''
    #     for i in s:
    #         res = res+i+' '
    #     return res[:-1]
    def reverseWords(self, s: str) -> str:
        s = s + ' '
        res = []
        temp = ''
        state = 0 # 1:writing
        i = 0
        while i<len(s):
            if state:
                if (s[i] == ' ') :
                    res.append(temp)
                    state = 0
                    temp = ''
                else:
                    temp = temp + s[i]
            else:
                if s[i] != ' ':
                    state = 1
                    temp = temp + s[i]
            i = i + 1

        return ' '.join(res[::-1])
```

搜索旋转数组

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left = 0
        right = len(nums)-1

        while left<right:
            mid = (left+right)//2
            if nums[mid] == target:
                left = mid
                right = mid

            elif nums[left]<nums[mid]:
                if target<nums[mid] and target>=nums[left]:
                    right = mid - 1
                else:
                    left = mid + 1
            elif nums[mid+1]<nums[right]:
                if target<=nums[right] and target>=nums[mid+1]:
                    left = mid + 1
                else:
                    right = mid - 1
            else:
                if nums[left] == target:
                    right = left
                elif nums[right] == target:
                    left = right
                else:
                    right = left
        if nums[left] == target:
            return left
        else:
            return -1
```

复原IP地址

```
class Solution:
    def restoreIpAddresses(self, s: str) -> List[str]:
        res = []

        def dfs(count, ip, s):
            if count == 4:
                if s == '':
                    res.append(ip[:-1])
                return

            if len(s) > 0:
                dfs(count+1, ip+ s[0] + '.', s[1:])
            if len(s) > 1 and s[0] != '0':
                dfs(count + 1, ip + s[:2] + '.', s[2:])
            if (len(s) > 2) and (int(s[:3]) < 256) and (s[0] != '0'):
                dfs(count + 1, ip + s[:3] + '.', s[3:])

        dfs(0, '', s)
        return res
```

最长连续递增序列

```
class Solution:
    def findLengthOfLCIS(self, nums: List[int]) -> int:

        if len(nums) == 0:
            return 0
        if len(nums) == 1:
            return 1

        length = [1]
        state = 0
        for i in range(1,len(nums)):
            if nums[i]>nums[i-1]:
                if state == 0:
                    length.append(2)
                    state = 1
                elif state == 1:
                    length[-1] = length[-1] + 1
            else:
                state = 0
        return max(length)
```

删除排序数组中的重复项

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        if len(nums) == 0:
            return 0
        if len(nums) == 1:
            return 1
        slow = 0
        fast = 1
        res = 1
        while fast<len(nums):
            if nums[fast] == nums[slow]:
                fast = fast + 1
            else:
                res = res + 1
                slow = fast
                fast = fast + 1
                nums[res-1] = nums[slow]
        return res
```