21. Merge Two Sorted Lists

```
# 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, list1: Optional[ListNode], list2:
Optional[ListNode]) -> Optional[ListNode]:
        dummy = ListNode(0)
        current = dummy
        while list1 and list2:
            if list1.val<list2.val:
                current.next = list1
                list1 = list1.next
            else:
                current.next = list2
                list2 = list2.next
            current = current.next
        if list1:
            current.next = list1
        if list2:
            current.next = list2
        return dummy.next
```

22. Generate Parentheses

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        res = []
        stack = []
        def backtrack(open, close):
            if open == close == n:
                res.append("".join(stack))
                return
            if open<n:
                stack.append("(")
                backtrack(open+1, close)
                stack.pop()
            if close<open:
                stack.append(")")
```

```
                backtrack(open, close+1)
                stack.pop()
        backtrack(0,0)
        return res
```

23. Merge k Sorted Lists

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def mergeKLists(self, lists: List[Optional[ListNode]]) ->
Optional[ListNode]:
        interval = 1
        while interval<len(lists):
            for i in range(0, len(lists)-interval, interval*2):
                lists[i] = self.mergeTwoList(lists[i], lists[i+interval])
            interval*=2
        return lists[0] if lists else None

    def mergeTwoList(self,list1, list2):
        dummy = ListNode(0)
        current = dummy
        while list1 and list2:
            if list1.val < list2.val:
                current.next = list1
                list1 = list1.next
            else:
                current.next = list2
                list2 = list2.next
            current = current.next
        if list1:
            current.next = list1
        if list2:
            current.next = list2
        return dummy.next
```

24. Swap Nodes in Pairs

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def swapPairs(self, head: Optional[ListNode]) -> Optional[ListNode]:
        dummy = ListNode(0)
        dummy.next = head
        current = dummy
        while current.next and current.next.next:
            node1 = current.next
            node2 = current.next.next
            current.next = node2
            node1.next = node2.next
            node2.next = node1
            current = node1
        return dummy.next
```

26. Remove Duplicates from Sorted Array

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        k = 1
        for i in range(1, len(nums)):
            if nums[i-1]!=nums[i]:
                nums[k] = nums[i]
                k = k+1
        return k
```

27. Remove Element

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        k = 0
        for i in range(len(nums)):
            if nums[i]!=val:
                nums[k]=nums[i]
                k = k+1
        return k
```

28. Find the Index of the First Occurrence in a String

```
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        if not needle or len(needle)>len(haystack):
            return -1
        for i in range(len(haystack)-len(needle)+1):
            if haystack[i:i+len(needle)]==needle:
                return i
        return -1
```

```
[7,2,5,3,1]
```

31. Next Permutation

```
class Solution:
    def nextPermutation(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        i = len(nums)-2
        while i>=0 and nums[i]>=nums[i+1]:
            i=i-1
        if i==-1:
            nums.reverse()
            return
        j = len(nums)-1
        while nums[j]<=nums[i]:
            j = j-1
        nums[i],nums[j]=nums[j],nums[i]
        nums[i+1:]=nums[i+1:][::-1]
```

32. Longest Valid Parentheses

```
class Solution:
    def longestValidParentheses(self, s: str) -> int:
        stack = [-1]
        maxLength = 0
        for i in range(len(s)):
            if s[i]=="(":
                stack.append(i)
            else:
                stack.pop()
```

```
                if not stack:
                    stack.append(i)
                else:
                    maxLength = max(maxLength, i-stack[-1])
        return maxLength
```

33. Search in Rotated Sorted Array

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        l = 0
        r = len(nums)-1
        while l<=r:
            m = (l+r)//2
            if nums[m]==target:
                return m
            if nums[l]<=nums[m]:
                if nums[l]<=target<nums[m]:
                    r = m-1
                else:
                    l = m+1
            else:
                if nums[m]<target<=nums[r]:
                    l = l+1
                else:
                    r = m-1
        return -1
```

34. Find First and Last Position of Element in Sorted Array

```
class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:
        def firstOccurance(nums,target):
            result = -1
            l = 0
            r = len(nums)-1
            while l<=r:
                m = (l+r)//2
                if nums[m]==target:
                    result = m
                    r = m-1
```

```
                elif nums[m]<target:
                    l = m+1
                else:
                    r = m-1
            return result
        def lastOccurance(nums,target):
            result = -1
            l = 0
            r = len(nums)-1
            while l<=r:
                m = (l+r)//2
                if nums[m]==target:
                    result = m
                    l = m+1
                elif nums[m]<target:
                    l = m+1
                else:
                    r = m-1
            return result

        first = firstOccurance(nums,target)
        last = lastOccurance(nums,target)
        return [first, last]
```

35. Search Insert Position

```
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        l = 0
        r = len(nums)-1
        while l<=r:
            m = (l+r)//2
            if nums[m]==target:
                return m
            elif target<nums[m]:
                r = m-1
            else:
                l = m+1
        return l
```

36. Valid Sudoku

```
class Solution:
    def isValidSudoku(self, board: List[List[str]]) -> bool:
        from collections import defaultdict
        rows = defaultdict(set)
        cols = defaultdict(set)
        squars = defaultdict(set)
        for i in range(9):
            for j in range(9):
                if board[i][j]=='.':
                    continue
                if (board[i][j] in rows[i] or board[i][j] in cols[j] or
board[i][j] in squars[(i//3,j//3)]):
                    return False
                rows[i].add(board[i][j])
                cols[j].add(board[i][j])
                squars[(i//3, j//3)].add(board[i][j])
        return True
```

39. Combination Sum

```
class Solution:
    def combinationSum(self, candidates: List[int], target: int)
-> List[List[int]]:
        res = []
        def backtrack(candidates, target, path):
            if target==0:
                res.append(path)
                return
            for i in range(len(candidates)):
                if candidates[i]>target:
                    continue
                backtrack(candidates[i:], target-candidates[i],
path+[candidates[i]])
        backtrack(candidates, target, [])
        return res
```

40. Combination Sum II

```
class Solution:
    def combinationSum2(self, candidates: List[int], target:
int) -> List[List[int]]:
        res = []
        candidates.sort()
        def backtrack(candidates, target, path):
            if target==0:
                res.append(path)
                return
            for i in range(len(candidates)):
                if candidates[i]>target:
                    continue
                if i>0 and candidates[i]==candidates[i-1]:
                    continue
                backtrack(candidates[i+1:],
target-candidates[i], path+[candidates[i]])
        backtrack(candidates, target, [])
        return res
```