Lists (LIST)

Reverse linked list

```
Class ListNode:
    def __init__(self, val=0, next=None):
        self.val=val
        self.next=next
def reverseList(head):
    prev=None
    cur=head
    while cur:
        next_node=cur.next
        cur.next=prev
        prev=cur
        cur=next_node
    return prev
```

Linked list cycle

```
def hasCycle(head):
    if not head:
        return False
    slow=head
    fast=head
    while fast and fast.next:
        slow=slow.next
        fast=fast.next.next
        if slow==fast:
            return True
    return False
```

Merge 2 sorted lists

```
def merge(list1, list2):
    dummy=ListNode(0)
cur=dummy
    while list1 and list2:
        if list1.val<list2.val:
            cur.next=list1
            list1=list1.next
        else:
            cur.next=list2
            list2=list2.next
        cur=cur.next
    if list1:
        cur.next=list1
    elif list2:
        cur.next=list2
    return dummy.next
```

Merge k sorted lists

```
import heapq
def mergeKLists(lists):
    heap=[]
    for l in lists:
        if l:
            heapq.heappush(heap,l)
    dummy=ListNode(0)
    cur=dummy
    while heap:
        smallest=heapq.heappop(heap)
        cur.next=smallest
        cur=cur.next
        if smallest.next:
            heapq.heappush(heap,
                           smallest.next)
    }
    return dummy.next
```

Remove nth node from end of list

```
def removeNthFromEnd(head):
    dummy=ListNode(0,head)
    fast=dummy
    slow=dummy
    for _ in range(n+1):
        fast=fast.next
    while fast:
        fast=fast.next
        slow=slow.next
    slow.next=slow.next.next
    return dummy.next
```

Reorder list: $L_0 \to L_1 \to \dots L_N$ into $L_0 \to L_N \to L_1 \to L_{N-1} \to \dots$

```
def reorderList(head):
    # find middle
    slow,fast=head,head.next
    while fast and fast.next:
        slow=slow.next
        fast=fast.next.next
    second=slow.next
    slow.next=None
    #reverse 2nd half
    prev=None
    while second:
        tmp=second.next
        second.next=prev
        prev=second
        second=tmp
    #merge two halves
    first,second=head,prev
    while second:
        tmp1,tmp2=first.next,second.next
        first.next=second
        second.next=tmp1
        first,second=tmp1,tmp2
```

MATRIX (MTRX)

matrix zeroes

```
def setZeroes(matrix):
    if not matrix:
        return
#determine if 1st row or 1st col
# should be zero
first_row_zero=any(mat[0][j]==0
    for j in range(len(matrix[0])))
first_col_zero=any(mat[i][0]==0
    for i in range(len(matrix)))
#use 1st row & 1st col as markers
for i in range(1,len(matrix)):
    for j in range(1,len(matrix[0])):
        if matrix[i][j]==0:
            matrix[i][0]=0
            matrix[0][j]=0
#set matrix cells to 0 based on markers:
for i in range(1,len(matrix)):
    for j in range(1,len(matrix[0])):
        if mat[i][0]==0 or mat[0][j]==0:
            mat[i][j]=0
#set 1st row to 0
if first_row_zero:
    for j in range(len(matrix[0])):
        matrix[0][j]=0
#set 1st col to 0:
if first_col_zero:
    for i in range(len(matrix)):
        matrix[i][0]=0
```

Spiral matrix

```
def spiralOrder(mat):
    if not mat:
        return []
    result=[]
    top,bottom=0,len(mat)-1
    left,right=0,len(mat[0])-1
    while top<=bottom and left<=right:
        for i in range(left,right+1):
            result.append(mat[top][i])
        top+=1
        #trav. from top to bottom along
        #right boundary
        for i in range(top,bottom+1):
            result.append(mat[i][right])
        right-=1
        if top<=bottom:
            # right to left along bottom
            for i in range(right,left-1,-1):
                result.append(mat[bottom][i])
            bottom-=1
        if left<=right:
            for i in range(bottom,top-1,-1):
                result.append(mat[i][left])
            left+=1
    return result
```

Rotate image/matrix by 90°

```
def rotate(mat):
    n=len(mat)
    #step 1: transpose the matrix
    for i in range(n):
        for j in range(i,n):
            mat[i][j],mat[j][i]=
                mat[j][i],mat[i][j]
    # step 2: reverse each row
    for i in range(n):
        matrix[i].reverse()
```

word search

```
def exist(board:List[List[str]],word):
    if not board:
        return False
    rows,cols=len(board),len(board[0])
    def dfs(r,c,index):
        if index==len(word):
            return True
        if r<0 or r>=rows or c<0 or
            c>=cols or board[r][c]!=
                word[index]:
                return False
        temp=board[r][c]
        board[r][c]='#' (#ie. visited)
        found=(dfs(r+1,c,index+1) or
                dfs(r-1,c,index+1) or
                dfs(r,c+1,index+1) or
                dfs(r,c-1,index+1))
        board[r][c]=temp
        return found
    for i in range(rows):
        for j in range(cols):
            if board[i][j]==word[0]
                and dfs(i,j,0):
                return True
    return False
```

STRING (STR)

Longest substr: w/o repeating chars

```
def lengthLongestSubstr(s):
    char_idx={}
    left=0
    max_length=0
    for right in range(len(s)):
        if s[right] in char_idx and
            char_idx[s[right]]>=left:
            left=char_idx[s[right]]+1
        char_idx[s[right]]=right
        max_length=max(max_length, right
                       -left+1)
    return max_length
```

Longest repeating character replacement

```
def characterReplacement(s,k):
    count={} #freq. of chars in the current window
    max_len=0
    max_count=0
    left=0
    for right in range(len(s)):
        count[s[right]]=count.get(s[right],0)+1
        max_count=max(max_count,count[s[right]])
        while (right-left+1)-max_count>k:
            count[s[left]]-=1
            left+=1
        max_len=max(max_len,right-left+1)
    return max_len
```

Minimum window substr. (min window in s that contains all characters in t)

```
def minWindow(s,t):
    if not t or not s: return ""
    t_count=Counter(t)
    required=len(t_count)
    left,right=0,0
    formed=0
    window_counts={}
    ans=(float("inf"),None,None)
    while right<len(s):
        char=s[right]
        window_counts[char]=window_coun-
            ts.get(char,0)+1
        if char in t_count and window_
                counts[char]==t_count[char]:
            formed+=1
        while left<=right and formed==req:
            char=s[left]
            if right-left+1<ans[0]:
                ans=(right-left+1,left,
                     right)
            window_counts[char]-=1
            left+=1
            if char in t_count & window_
                counts[char]<t_counts[char]:
                formed-=1
```