206. Reverse Linked List
This question is very typical. We could reverse either iteratively or recursively. The TC are both O(n)

```
class Solution(object):
    def reverseList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        dummy = ListNode(0)
        while head:
            next_node = head.next
            head.next = dummy.next
            dummy.next = head
            head = next_node
        return dummy.next
```

For recursively, it's a little difficult to come up with, we need to traverse to the last two nodes, and reverse them, then return the first node, and so on so forth. We could reverse the whole linkedlist.

253. Meeting Rooms II
For this question, we could use two arrays to store all meetings' start time and end time. Then we just sort these two arrays. We will compare the first element of start array and end array, if start one is smaller, we will increase count by 1, and move index to the next element of start, so as end array. In this process, we always store the max count, which is what we want. The TC is O(nlogn)

```
class Solution(object):
    def minMeetingRooms(self, intervals):
        """
        :type intervals: List[List[int]]
        :rtype: int
        """
        start = []
        end = []
        count = 0
        max_room = 0
        for interval in intervals:
            start.append(interval[0])
            end.append(interval[1])
        start.sort(reverse=True)
        end.sort(reverse=True)
        while start and end:
            if start[-1] < end[-1]:
                count += 1
                max_room = max(count, max_room)
                start.pop()
            elif start[-1] > end[-1]:
                count -= 1
                end.pop()
            else:
                start.pop()
                end.pop()
        return max_room
```

7. Reverse Integer
This question is pretty simple, we just need to transform it into string first and reverse it and transform to integer. Also don’t forget the sign if it’s negative.

```
class Solution(object):
    def reverse(self, x):
        """
        :type x: int
        :rtype: int
        """
        mark = 1
        if x < 0:
            x = -x
            mark = -1
        result = mark * int(str(x)[::-1])
        if -2**31 <= result <= 2**31 - 1:
            return result
        return 0
```

297. Serialize and Deserialize Binary Tree
For this question, I used layer traversal to serialize and deserialize binary tree, we would use ‘#’ to represent None. The TC is O(n). We use BFS. I would implement using DFS tomorrow, which I think is pretty cool.

```
from collections import deque
class Codec:

    def serialize(self, root):
        """Encodes a tree to a single string.

        :type root: TreeNode
        :rtype: str
        """
        if not root:
            return '#'
        cur = [root]
        result = [str(root.val)]
        while cur:
            next_layer = []
            for node in cur:
                if node.left:
                    next_layer.append(node.left)
                    result.append(str(node.left.val))
                else:
                    result.append('#')
                if node.right:
                    next_layer.append(node.right)
                    result.append(str(node.right.val))
                else:
                    result.append('#')
            cur = next_layer
        return ','.join(result)



    def deserialize(self, data):
        """Decodes your encoded data to tree.

        :type data: str
        :rtype: TreeNode
        """
        data_arr = data.split(',')[::-1]
        head = None
        nodes = deque()
        while data_arr:
            left_val = data_arr.pop()
            left = None if left_val == '#' else TreeNode(int(left_val))
            if left:
                nodes.append(left)
            if not head:
                head = left
                continue
            right_val = data_arr.pop()
            right = None if right_val == '#' else TreeNode(int(right_val))
            if right:
                nodes.append(right)
            node = nodes.popleft()
            node.left = left
            node.right = right
        return head
```

238. Product of Array Except Self

This question is quite tricky. We will divide whole process into two parts. For the first one, we would calculate the product of this element's left elements, then in the second traverse, we would iterate from right side, multiply its right elements.

```
class Solution(object):
    def productExceptSelf(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        res = [1]
        multi = 1
        for num in nums:
            multi *= num
            res.append(multi)
        multi = 1
        for i in range(len(res) - 2, -1, -1):
            res[i] *= multi
            multi *= nums[i]
        return res[:-1]
```