138. Copy List with Random Pointer
This question is kind of interesting. Because we cannot set random before we creating them. So we need to iterate through the whole linked list twice. In the first time, we need to copy the whole linked list except setting random pointer. Also we need to use map to associate origin nodes with copied nodes so that we could use them for random in the next iteration. The TC is O(n).

```
class Solution:
    def copyRandomList(self, head: 'Node') -> 'Node':
        map_node = {}
        map_node[None] = None
        dummy = Node(0, None, None)
        dummy_memo = dummy
        head_memo = head
        while head:
            dummy.next = Node(head.val, None, None)
            map_node[head] = dummy.next
            dummy = dummy.next
            head = head.next
        head = head_memo
        dummy = dummy_memo
        while head:
            dummy.next.random = map_node[head.random]
            dummy = dummy.next
            head = head.next
        return dummy_memo.next
```

33. Search In Rotated Sorted Array
For this question, we will try to move mid and target to the same part if they locate in two separated parts. Then we could do normal binary search.

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left, right = 0, len(nums) - 1
        while left <= right:
            mid = (left + right) // 2
            if nums[mid] == target:
                return mid
            if (nums[mid] >= nums[0] and target >= nums[0]) or (nums[mid] < nums[0] and target <
nums[0]):
                if nums[mid] > target:
                    right = mid - 1
                else:
```

```
                left = mid + 1
            else:
                if nums[0] > target:
                    left = mid + 1
                else:
                    right = mid - 1
        return -1
```

11. Container With Most Water
This question is quite simple. We could start from both sides. First, compare the height of both sides and set shorter one at our height to compute the container water area. At the same time, we will move the index with shorter height towards to center by one step. Repeat this process until we two indexes meet. The time complexity is O(n)

```
class Solution:
    def maxArea(self, height: List[int]) -> int:
        length = len(height)
        left, right = 0, length - 1
        max_water = 0
        while left < right:
            if height[left] < height[right]:
                area = (right - left) * height[left]
                left += 1
            else:
                area = (right - left) * height[right]
                right -= 1
            max_water = max(max_water, area)
        return max_water
```

269. Alien Dictionary
This question is quite complicated, we need to compare every two consecutive words and get the first different words and know their order in new dictionary. We will use map to remember each letter's parents and descendants. After this finished, we would find out all letter which doesn't have parents and put them first in the order list. Then delete all these letters from existing letter from their parents set, check whether there are 0 parent node. Get it out and repeat the process. Finally, we will get what we want. Also we need to check every letter is in our result. If not, it's illegal. Topological ordering.

```
from collections import defaultdict
class Solution:
    def alienOrder(self, words: List[str]) -> str:
        word_set = set()
        parents = defaultdict(set)
        decedents = defaultdict(set)
        length = len(words)
        ite_word = set()
        result = []
        for i in range(length - 1):
            word_set |= set(words[i])
            for j in range(min(len(words[i]), len(words[i + 1]))):
                if words[i][j] != words[i + 1][j]:
                    parents[words[i + 1][j]].add(words[i][j])
                    decedents[words[i][j]].add(words[i + 1][j])
                    break

        word_set |= set(words[length - 1])
        word_length = len(word_set)
        for word in word_set:
            if len(parents[word]) == 0:
                result.append(word)
                ite_word.add(word)

        word_set = ite_word
        ite_word = set()
        while len(word_set) > 0:
            for w in word_set:
                for word in decedents[w]:
                    parents[word].remove(w)
                    if len(parents[word]) == 0:
                        result.append(word)
                        ite_word.add(word)
            word_set = ite_word
            ite_word = set()
        if len(result) != word_length:
            return ''
        else:
            return ''.join(result)
```

49. Group Anagrams
This question is very simple, we only need to set strings in lexi order as key and store them in the same list. In the end, we only need to return all values.

```
from collections import defaultdict
class Solution:
```

```
def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
    str_map = defaultdict(list)
    for s in strs:
        key = ''.join(sorted(list(s)))
        str_map[key].append(s)
    return str_map.values()
```