81. Search in Rotated Sorted Array II
We would skip all duplications in case both two sides in case we cannot see which side is ordered. Except this, we will do the same thing as I. TC is O(n)

```
class Solution:
    def search(self, nums: List[int], target: int) -> bool:
        l, r = 0, len(nums) - 1
        if not nums:
            return False
        while l <= r:
            while l < r and nums[l] == nums[l + 1]:
                l += 1
            while l < r and nums[r] == nums[r - 1]:
                r -= 1
            mid = (l + r) // 2
            if nums[mid] == target:
                return True
            if nums[mid] < nums[0]:
                if target < nums[0]:
                    if nums[mid] < target:
                        l = mid + 1
                    else:
                        r = mid - 1
                else:
                    r = mid - 1
            else:
                if target >= nums[0]:
                    if nums[mid] < target:
                        l = mid + 1
                    else:
                        r = mid - 1
                else:
                    l = mid + 1

        return False
```

388. Longest Absolute File Path
We will use hashmap to record all layer's total length, when it's a title, we will compare its length with our max_length and always maintain the max length. Otherwise, we will add the previous one to current one. TC is O(n)

```
from collections import defaultdict
class Solution:
    def lengthLongestPath(self, input: str) -> int:
        memo = defaultdict(int)
        max_length = 0
        for line in input.split('\n'):
            name = line.lstrip('\t')
            if '.' in name:
                max_length = max(memo[len(line) - len(name) - 1] + len(name), max_length)
            else:
                memo[len(line) - len(name)] = memo[len(line) - len(name) - 1] + 1 + len(name)
        return max_length
```

86. Partition List
We will use a list to record all nodes that are larger or equal to given x. At the same time, delete it from current linked list. In the end, appending all these nodes to the end of linked list. TC is O(n)

```
class Solution:
    def partition(self, head: ListNode, x: int) -> ListNode:
        nodes = []
        dummy = ListNode(0)
        dummy.next = head
        dummy_mem = dummy
        while dummy.next:
            if dummy.next.val >= x:
                nodes.append(dummy.next)
                dummy.next = dummy.next.next
            else:
                dummy = dummy.next
        for node in nodes:
            dummy.next = node
            dummy = dummy.next
        dummy.next = None
        return dummy_mem.next
```

90. Subsets II
We will see duplicate elements as a special elements totally. We could add it in three different ways. So when using bfs, we append different number of elements to different lists. TC is O(n)

```
from collections import Counter
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        counter = Counter(nums)
        cur = [[]]
        for i in counter.keys():
            next_cur = []
            for e in cur:
                for num in range(1, counter[i] + 1):
                    next_cur.append(e + [i] * num)
            cur += next_cur
        return cur
```

695. Max Area of Island
We will use the same way as count islands. Using bfs to cund al connected 1s. TC is O(n)

```
class Solution:
    def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
        if not grid or not grid[0]:
            return 0
        visited = {}
        directions = [[0, 1], [0, -1], [1, 0], [-1, 0]]
        q = []
        count = 0
        rows, cols = len(grid), len(grid[0])
        for i in range(rows):
            for j in range(cols):
                if grid[i][j] == 1 and (i, j) not in visited:
                    temp = 1
                    q.clear()
                    q.append((i, j))
                    visited[(i, j)] = True
                    while len(q) > 0:
                        x, y = q.pop()
                        for d_x, d_y in directions:
                            if 0 <= x + d_x < rows and 0 <= y + d_y < cols and grid[x + d_x][y + d_y] == 1
and (x + d_x, y + d_y) not in visited:
                                q.append((x + d_x, y + d_y))
                                temp += 1
                                visited[(x + d_x, y + d_y)] = True
                    count = max(count, temp)
        return count
```