31. Next Permutation

```
class Solution:
    def nextPermutation(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place instead.

        1 5 2 4 3 5 6 7 8 - > 8 7
        1 3 5 4 3 2 -> 2 1 3 4 5
        """


        # find the first number that is larger than the last number backward

        ind = len(nums) - 1 # 5
        if ind <=0:
            return
        while ind > 0 and nums[ind] <= nums[ind-1]:
            ind = ind-1

        if ind == 0:
            nums=  nums.reverse()
            return

        # ind = 2 for the second example
        ind2 = len(nums)-1
        while ind2 > 0 and nums[ind2] <= nums[ind-1]:
            ind2 = ind2-1

        temp = nums[ind-1]
        nums[ind-1] = nums[ind2]
        nums[ind2] = temp

        l = ind
        r = len(nums) - 1

        while l < r:
            temp = nums[l]
            nums[l] = nums[r]
            nums[r] =temp
            l = l+1
            r= r-1
```

33. Search in Rotated Sorted Array

this problem is still very hard for me to solve bug-free.

```
class Solution:
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """

        if len(nums) == 0 :return -1
        l = 0
        r = len(nums) - 1




        if r == 0:
            return 0 if nums[0] == target else -1

        while l <= r:
            mid = (l + r) // 2

            if nums[mid] < nums[l] and target < nums[l]:
                num_mid = nums[mid]
            elif nums[mid] >= nums[l] and target >= nums[l]:
                num_mid = nums[mid]
            elif target < nums[l]:
                num_mid = -9999
            else:
                num_mid = 9999

            if num_mid == target:
                return mid

            if num_mid > target:
                r = mid
            else:
                l = mid + 1

        return -1
```

41. First Missing Positive

```
class Solution:
    def firstMissingPositive(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        l = len(nums)
        for i in range(l):

            if nums[i] < 1 or nums[i] > l:
                continue
            else:
                while nums[i] != i+1 and nums[i] >=1 and nums[i] <=l:
                    if nums[nums[i] - 1] == nums[i]:
                        break
                    temp = nums[i]
                    nums[i] = nums[nums[i] - 1]
                    nums[temp - 1] = temp

        for i in range(l):
            if nums[i] != i+1:
                return  i+1
        return l+1
```

54. Spiral Matrix

```
class Solution:
    def spiralOrder(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: List[int]
        """
        res = []

        num_row = len(matrix)
        if num_row == 0 or len(matrix[0]) == 0: return res
        num_col = len(matrix[0])

        row_s = 0
        row_e = num_row - 1
        col_s = 0
        col_e = num_col - 1

        while row_s <= row_e and col_s <= col_e:
            for i in range(col_s, col_e + 1):
                res.append(matrix[row_s][i])

            row_s += 1
            if row_s > row_e:
                break

            for i in range(row_s, row_e + 1):
                res.append(matrix[i][col_e])
            col_e = col_e - 1
            if col_s > col_e:
                break

            for i in range(col_e, col_s - 1, -1):
                res.append(matrix[row_e][i])
            row_e = row_e - 1
            if row_s > row_e:
                break

            for i in range(row_e, row_s - 1, -1):
                res.append(matrix[i][col_s])
            col_s += 1
            if col_s > col_e:
                break

        return res
```

128. Longest Consecutive Sequence

need to redo

```
class Solution:
    def longestConsecutive(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if len(nums) == 0 : return  0
        res = 1

        nums = set(nums)

        for n in nums:
            if n - 1 not in nums:
                y = n+1
                while y in nums:
                    y = y+1
                    res = max(res, y-n)
        return res
```

153. Find Minimum in Rotated Sorted Array

```
class Solution:
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """

        l = 0
        r = len(nums) - 1

        while l < r:
            if nums[l] < nums[r]:
                return nums[l]
            mid = (l + r) // 2
            if nums[mid] > nums[l]:
                l = mid + 1
            elif nums[mid] < nums[l]:
                r = mid
            else:
                return min(nums[l], nums[r])
        return nums[l]
```

229. Majority Element II

don't have any idea, need to redo this one

```
class Solution:
    def majorityElement(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]

        Boyer–Moore Majority Vote
        """

        if not nums:
            return []

        count1, count2, candidate1,candidate2 = 0,0,0,1

        for n in nums:
            if n == candidate1:
                count1 +=1
            elif n == candidate2:
                count2 +=1
            elif count1 == 0:
                candidate1, count1 = n, 1
            elif count2 == 0:
                candidate2,count2 = n,1
            else:
                count1, count2 = count1 - 1, count2 - 1

        return  [n for n in (candidate1,candidate2)  if nums.count(n) > len(nums)//3
]
```

287. Find the Duplicate Number

```
class Solution:
    def findDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        slow = nums[0]
        fast = nums[slow]

        while slow != fast:
            slow= nums[slow]
            fast = nums[nums[fast]]

        fast = 0
        while fast != slow:
            slow = nums[slow]
            fast = nums[fast]
        return fast
```

289. Game of Life

```
class Solution:
    def isAlive(self, board,i,j ):
        if i<0:
            return 0
        if j<0:
            return 0
        if i>= len(board):
            return 0
        if j>=len(board[0]):
            return 0

        if board[i][j] % 2 == 1:
            return 1
        else:
            return  0


    def count(self, board, i, j ):

        return self.isAlive(board, i-1,j-1) + self.isAlive(board, i-1,j) + self.isAl
ive(board, i-1,j+1) + self.isAlive(board, i,j-1) + self.isAlive(board, i,j+1) + sel
f.isAlive(board, i+1,j-1) + self.isAlive(board, i+1,j) + self.isAlive(board, i+1,j+
1)



    def gameOfLife(self, board):
        """
        :type board: List[List[int]]
        :rtype: void Do not return anything, modify board in-place instead.
        """
        n_row = len(board)
        n_col = len(board[0])

        for i in range(n_row):
            for j in range(n_col):
                n_live = self.count(board,i,j)
                if board[i][j] == 1:
                    if n_live == 2 or n_live==3:
                        board[i][j] = 3
                    else:
                        board[i][j] = 1
                else:
                    if n_live == 3:
                        board[i][j] = 2

        for i in range(n_row):
            for j in range(n_col):
                board[i][j]  =  board[i][j] //2
```

695. Max Area of Island

```
class Solution:
    def mark(self, grid,i,j):
        if i<0 or j<0 or i >= len(grid) or j>= len(grid[0]) or grid[i][j] == 0:
            return 0

        grid[i][j] = 0
        return self.mark(grid, i, j+1) + self.mark(grid, i, j-1) + self.mark(grid, i
-1, j) + self.mark(grid, i+1, j) +1

    def maxAreaOfIsland(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int

        """
        res = 0

        n_row = len(grid)
        n_col = len(grid[0])
        for i in range(n_row):
            for j in range(n_col):
                if grid[i][j] == 1:
                    res = max(res, self.mark(grid,i,j))
        return  res
```