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 5 4 3 2 -> 2 1 3 4 5
        1 3 5 5 4 3 2 -> 1 3 2 3 4 5 5 ->  1 4 2 3 3 5 5
            i
        """

        ind = len(nums) - 1 # = 6 for the third example,
        while ind > 0:
            if nums[ind] > nums[ind-1]:
                break
            else:
                ind -= 1
        if ind == 0:
            nums.reverse()
            return

        ind2 = len(nums)-1
        while ind2 > 0 and nums[ind2] <= nums[ind-1]:
            ind2 = ind2-1


        nums[ind-1], nums[ind2] =nums[ind2], nums[ind-1]
        l = ind
        r = len(nums)-1

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

41. First Missing Positive

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

        for i in range(len(nums)):
            while nums[i] != i+1 and nums[i]>0 and nums[i]<=len(nums):
                if nums[nums[i]-1] == nums[i]:
                    break
                else:
                    nums[nums[i]-1], nums[i] =nums[i] ,  nums[nums[i]-1]
        for i in range(len(nums)):
            if nums[i] != i+1:
                return i+1
        return len(nums) + 1
```

48. Rotate Image

```
class Solution:
    def rotate(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: void Do not return anything, modify matrix in-place instead.
        """
        num_row = len(matrix)
        num_col = len(matrix[0])
        start = 0
        end = num_row - 1
        while start < end:
            matrix[start], matrix[end] = matrix[end], matrix[start]
            start += 1
            end -= 1

        for i in range(num_row):
            for j in range(i+1, num_col):
                matrix[i][j], matrix[j][i] =matrix[j][i], matrix[i][j]
```

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_start = 0
        col_start = 0
        row_end = num_row - 1
        col_end = num_col - 1
        while row_start<= row_end and col_start<= col_end:

            for i in range(col_start, col_end + 1):
                res.append(matrix[row_start][i])
            row_start += 1
            if row_start > row_end:
                break

            for i in range(row_start, row_end+1):
                res.append(matrix[i][col_end])
            col_end -= 1
            if col_start > col_end:
                break

            for i in range(col_end, col_start-1, -1):
                res.append(matrix[row_end][i])
            row_end -= 1
            if row_start > row_end:
                break

            for i in range(row_end, row_start-1, -1):
                res.append(matrix[i][col_start])
            col_start += 1
            if col_start > col_end:
                break

        return res
```

55. Jump Game

```
class Solution:
    def canJump(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        i = 0
        reach = 0
        while i < len(nums) and i <= reach:
            reach = max(i+nums[i], reach)
            i += 1
        return i == len(nums)
```

56. Merge Intervals

```
# Definition for an interval.
# class Interval:
#     def __init__(self, s=0, e=0):
#         self.start = s
#         self.end = e

class Solution:
    def merge(self, intervals):
        """
        :type intervals: List[Interval]
        :rtype: List[Interval]
        """
        intervals.sort(key = lambda x:x.start)

        res = []
        if intervals is None:
            return res
        if len(intervals) <= 1:
            return intervals

        start = intervals[0].start
        end = intervals[0].end

        for i in range(1, len(intervals)):
            if intervals[i].start <= end:
                start = min(start, intervals[i].start)
                end = max(end, intervals[i].end)

            else:
                res.append(Interval(start, end))
                start = intervals[i].start
                end = intervals[i].end
        res.append(Interval(start, end))
        return res
```

57. Insert Interval

```
# Definition for an interval.
# class Interval:
#     def __init__(self, s=0, e=0):
#         self.start = s
#         self.end = e

class Solution:
    def insert(self, intervals, newInterval):
        """
        :type intervals: List[Interval]
        :type newInterval: Interval
        :rtype: List[Interval]

        s, e = newInterval.start, newInterval.end
        left = [i for i in intervals if i.end < s]
        right = [i for i in intervals if i.start > e]
        if left + right != intervals:
            s = min(s, intervals[len(left)].start)
            e = max(e, intervals[~len(right)].end)
        return left + [Interval(s, e)] + right
        """
        start = newInterval.start
        end = newInterval.end
        left = [i for i in intervals if i.end < newInterval.start]
        right = [i for i in intervals if i.start > newInterval.end]

        if left + right != intervals:

            start = min(newInterval.start, intervals[len(left)].start)
            end = max(newInterval.end, intervals[~len(right)].end)
        return left + [Interval(start, end)] + right
```

59. Spiral Matrix II

```
def generateMatrix(self, n):
```

73. Set Matrix Zeroes

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

        first_row_zero = False
        first_col_zero = False

        num_row = len(matrix)
        num_col = len(matrix[0])

        for i in range(num_col):
            if matrix[0][i] == 0:
                first_row_zero = True
                break

        for i in range(num_row):
            if matrix[i][0] == 0:
                first_col_zero = True
                break

        for i in range(1, num_row):
            for j in range(1, num_col):
                if matrix[i][j] == 0:
                    matrix[i][0] = 0
                    matrix[0][j] = 0

        for i in range(1, num_row):
            if matrix[i][0] == 0:
                for j in range(num_col):
                    matrix[i][j] = 0
        for i in range(1, num_col):
            if matrix[0][i] == 0:
                for j in range(num_row):
                    matrix[j][i] = 0

        if first_row_zero:
            for i in range(num_col):
                matrix[0][i] = 0


        if first_col_zero:
            for i in range(num_row):
                matrix[i][0] = 0
```

169. Majority Element

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

        num = nums[0]
        count = 1

        for n in nums[1:]:
            if n == num:
                count+=1
            else:
                if count == 0:
                    num = n
                    count = 1
                else:
                    count -= 1
        return num
```

228. Summary Ranges

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

        res = []
        if nums is None or len(nums) == 0:
            return res



        start = nums[0]
        end = nums[0]

        for i in range(1,len(nums)):
            if nums[i] == end + 1:
                end += 1
            else:
                if start==end:
                    res.append((str)(start))
                else:
                    res.append("{}->{}".format(start,end))

                start = nums[i]
                end = start

        if start==end:
            res.append((str)(start))
        else:
            res.append("{}->{}".format(start,end))
        return res
```

442. Find All Duplicates in an Array

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

        res = []

        for num in nums:
            if nums[abs(num) - 1] < 0:
                res.append(abs(num))
            else:
                nums[abs(num) -1] *= -1
        return res
```

562. Longest Line of Consecutive One in Matrix

```
class Solution:
    def longestLine(self, board):
        """
        :type M: List[List[int]]
        :rtype: int
        """
        if len(board) == 0:
            return 0
        num_row, num_col = len(board), len(board[0])
        res = 0
        for i in range(num_row):
            for j in range(num_col):
                if board[i][j] == 1:

                    # the longest vertical
                    if i == 0 or board[i-1][j] != 1:
                        k = i
                        while k < num_row and board[k][j] == 1  :
                            k+=1
                        res = max(res, k-i)

                    if j == 0 or board[i][j-1] != 1:
                        k = j
                        while k < num_col and board[i][k] == 1  :
                            k+=1
                        res = max(res, k-j)
                    if (i == 0 or j ==0) or board[i-1][j-1] != 1:
                        k = 0
                        while i+k < num_row and j+k < num_col and board[k+i][j+k] ==
1  :
                            k+=1
                        res = max(res, k)
                    if (i == 0 or j == num_col - 1) or board[i-1][j+1] != 1:
                        k = 0
                        while i+k < num_row and j-k>=0 and board[k+i][j-k]==1:
                            k+=1
                        res = max(res, k)
        return res
```