[41. First Missing Positive](https://leetcode.com/problems/first-missing-positive/)

class Solution:

def firstMissingPositive(self, nums: List[int]) -> int:

nums\_set = set(nums)

i = 1

while i in nums\_set:

i += 1

return i

class Solution:

def firstMissingPositive(self, nums: List[int]) -> int:

if not nums:

return 1

n = len(nums)

for i in range(n):

while 1<=nums[i]<=n and nums[nums[i]-1]!=nums[i]:

nums[nums[i]-1], nums[i] = nums[i], nums[nums[i]-1]

for i in range(n):

if nums[i]!=i+1:

return i+1

return n+1

[42. Trapping Rain Water](https://leetcode.com/problems/trapping-rain-water/)

class Solution:

def trap(self, height: List[int]) -> int:

l = 0

r = len(height)-1

lMax = 0

rMax = 0

trapedWater = 0

while l<r:

if height[l]<height[r]:

if height[l]>=lMax:

lMax = height[l]

else:

trapedWater+=lMax-height[l]

l = l+1

else:

if height[r]>=rMax:

rMax = height[r]

else:

trapedWater+=rMax-height[r]

r = r-1

return trapedWater

[43. Multiply Strings](https://leetcode.com/problems/multiply-strings/)

class Solution:

def multiply(self, num1: str, num2: str) -> str:

def str2int(s):

res = 0

for n in s:

res = res\*10 + ord(n)-ord('0')

return res

val1 = str2int(num1)

val2 = str2int(num2)

ans = str(val1\*val2)

return ans

[55. Jump Game](https://leetcode.com/problems/jump-game/)

class Solution:

def canJump(self, nums: List[int]) -> bool:

goal = len(nums)-1

for i in range(len(nums)-1, -1, -1):

if i+nums[i]>=goal:

goal = i

return goal==0

[45. Jump Game II](https://leetcode.com/problems/jump-game-ii/)

class Solution:

def jump(self, nums: List[int]) -> int:

l = 0

r = 0

count =0

while r<len(nums)-1:

farthest = 0

for i in range(l, r+1):

farthest = max(farthest, nums[i]+i)

l = r+1

r = farthest

count = count+1

return count

[46. Permutations](https://leetcode.com/problems/permutations/)

class Solution:

def permute(self, nums: List[int]) -> List[List[int]]:

res = []

def backtrack(nums, path):

if not nums:

res.append(path)

return

for i in range(len(nums)):

backtrack(nums[:i]+nums[i+1:], path+[nums[i]])

backtrack(nums,[])

return res

[47. Permutations II](https://leetcode.com/problems/permutations-ii/)

class Solution:

def permuteUnique(self, nums: List[int]) -> List[List[int]]:

res = []

nums.sort()

def backtrack(nums, path):

if not nums:

res.append(path)

return

for i in range(len(nums)):

if i>0 and nums[i]==nums[i-1]:

continue

backtrack(nums[:i]+nums[i+1:], path+[nums[i]])

backtrack(nums, [])

return res

[48. Rotate Image](https://leetcode.com/problems/rotate-image/)

class Solution:

def rotate(self, matrix: List[List[int]]) -> None:

"""

Do not return anything, modify matrix in-place instead.

"""

l = 0

r = len(matrix)-1

while l<r:

matrix[l], matrix[r] = matrix[r], matrix[l]

l = l+1

r = r-1

for i in range(len(matrix)):

for j in range(i):

matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

[49. Group Anagrams](https://leetcode.com/problems/group-anagrams/)

class Solution:

def groupAnagrams(self, strs: List[str]) -> List[List[str]]:

from collections import defaultdict

group\_anagrams = defaultdict(list)

for word in strs:

key = ''.join(sorted(word))

group\_anagrams[key].append(word)

return list(group\_anagrams.values())

[50. Pow(x, n)](https://leetcode.com/problems/powx-n/)

class Solution:

def myPow(self, x: float, n: int) -> float:

if n==0:

return 1

if n<0:

x= 1/x

n = -n

if n%2==0:

return self.myPow(x\*x, n//2)

else:

return x\*self.myPow(x\*x, n//2)

[53. Maximum Subarray](https://leetcode.com/problems/maximum-subarray/)

class Solution:

def maxSubArray(self, nums: List[int]) -> int:

maxSum = nums[0]

currentSum = 0

for n in nums:

currentSum+=n

maxSum = max(maxSum, currentSum)

if currentSum<0:

currentSum = 0

return maxSum

[54. Spiral Matrix](https://leetcode.com/problems/spiral-matrix/)

class Solution:

def spiralOrder(self, matrix: List[List[int]]) -> List[int]:

l = 0

r = len(matrix[0])

top = 0

bottom = len(matrix)

res = []

while l<r and top<bottom:

for i in range(l,r):

res.append(matrix[top][i])

top = top+1

for i in range(top, bottom):

res.append(matrix[i][r-1])

r = r-1

if not (l<r and top<bottom):

break

for i in range(r-1, l-1,-1):

res.append(matrix[bottom-1][i])

bottom = bottom-1

for i in range(bottom-1,top-1,-1):

res.append(matrix[i][l])

l = l+1

return res

[56. Merge Intervals](https://leetcode.com/problems/merge-intervals/)

class Solution:

def merge(self, intervals: List[List[int]]) -> List[List[int]]:

intervals.sort(key = lambda x:x[0])

merged = []

for n in intervals:

if not merged or merged[-1][-1]<n[0]:

merged.append(n)

else:

merged[-1][-1] = max(merged[-1][-1], n[1])

return merged

[57. Insert Interval](https://leetcode.com/problems/insert-interval/)

class Solution:

def insert(self, intervals: List[List[int]], newInterval: List[int]) -> List[List[int]]:

intervals.append(newInterval)

intervals.sort()

merged = []

for n in intervals:

if not merged or merged[-1][-1]<n[0]:

merged.append(n)

else:

merged[-1][-1] = max(merged[-1][-1], n[1])

return merged

class Solution:

def insert(self, intervals: List[List[int]], newInterval: List[int]) -> List[List[int]]:

i = 0

while (i<len(intervals) and intervals[i][0]<newInterval[0]):

i = i+1

intervals.insert(i, newInterval)

print(intervals)

merged = []

for n in intervals:

if not merged or merged[-1][-1]<n[0]:

merged.append(n)

else:

merged[-1][-1] = max(merged[-1][-1], n[1])

return merged

[58. Length of Last Word](https://leetcode.com/problems/length-of-last-word/)

class Solution:

def lengthOfLastWord(self, s: str) -> int:

m = s.strip().split()

return len(m[-1])

[59. Spiral Matrix II](https://leetcode.com/problems/spiral-matrix-ii/)

class Solution:

def generateMatrix(self, n: int) -> List[List[int]]:

matrics = [[0]\*n for \_ in range(n)]

l=0

r = n

top = 0

bottom = n

val = 1

while l<r and top<bottom:

for i in range(l,r):

matrics[top][i]=val

val = val+1

top = top+1

for i in range(top,bottom):

matrics[i][r-1]=val

val = val+1

r = r-1

for i in range(r-1, l-1,-1):

matrics[bottom-1][i]=val

val = val+1

bottom = bottom-1

for i in range(bottom-1, top-1, -1):

matrics[i][l]=val

val = val+1

l = l+1

return matrics