1.SEARCH INSERT POSITION

class Solution(object):

def searchInsert(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: int

"""

left, right = 0, len(nums) - 1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

return mid

elif nums[mid] < target:

left = mid + 1

else:

right = mid - 1

return left

2.SUBSETS

class Solution(object):

def subsets(self, nums):

nums.sort()

result = [[]]

for num in nums:

result += [i + [num] for i in result]

return result

3.COMBINATIONS

class Solution:

def \_\_init\_\_(self):

self.res = []

def solve1(self, num, tot, k, ans):

if num == tot + 1:

if len(ans) == k:

self.res.append(ans[:])

return

ans.append(num)

self.solve1(num + 1, tot, k, ans)

ans.pop()

self.solve1(num + 1, tot, k, ans)

def solve2(self, num, tot, k, ans):

if len(ans) == k:

self.res.append(ans[:])

return

for i in range(num, tot + 1):

ans.append(i)

self.solve2(i + 1, tot, k, ans)

ans.pop()

def combine(self, n, k):

ans = []

self.solve2(1, n, k, ans)

return self.res

4.MAXIMUM SUBARRAY

class Solution:

# @param A, a list of integers

# @return an integer

# 6:57

def maxSubArray(self, A):

if not A:

return 0

curSum = maxSum = A[0]

for num in A[1:]:

curSum = max(num, curSum + num)

maxSum = max(maxSum, curSum)

return maxSum

5.SEARCH IN ROTATED SORTED ARRAY

class Solution(object):

def search(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: int

"""

low = 0

high = len(nums) - 1

while low <= high:

mid = (low + high) // 2

if nums[mid] == target:

return mid

if nums[low] <= nums[mid]:

if nums[low] <= target < nums[mid]:

high = mid - 1

else:

low = mid + 1

else:

if nums[mid] < target <= nums[high]:

low = mid + 1

else:

high = mid - 1

return -1

6. GENERATE PARENTHESES

class Solution(object):

def generateParenthesis(self, n):

def backtrack(S='',left=0,right=0):

if len(S)==2\*n:

result.append(S)

return

if left<n:

backtrack(S+'(',left+1,right)

if right<left:

backtrack(S+')',left,right+1)

result=[]

backtrack()

return result

7. COMBINATION SUM

class Solution(object):

def combinationSum(self, can, target):

"""

:type candidates: List[int]

:type target: int

:rtype: List[List[int]]

"""

res=[]

def backtrace(st,com,curr):

if(curr==target):

res.append(list(com))

return

if curr>target:

return

for i in range(st, len(can)):

com.append(can[i])

backtrace(i,com,curr+can[i])

com.pop()

backtrace(0, [],0)

return res

8. LETTER COMBINATIONS OF A PHONE NUMBER

class Solution(object):

def letterCombinations(self,digits):

dict\_1 = {"2":"abc","3":"def","4":"ghi","5":"jkl","6":"mno","7":"pqrs","8":"tuv","9":"wxyz"}

if not digits:

return []

result = [""]

for x in digits:

temp = []

for a in result:

for j in dict\_1[x]:

temp.append(a + j)

result = temp

return result

9. CONTAINER WITH MOST WATER

class Solution:

def maxArea(self, v):

i = 0

n = len(v)

j = n - 1

area = 0

while j > i:

area = max(area, min(v[i], v[j]) \* abs(i - j))

if v[i] < v[j]:

i += 1

else:

j -= 1

return area

10. LONGEST SUBSTRING WITHOUT REPEATING CHARACTERS

class Solution(object):

def lengthOfLongestSubstring(self, s):

if not s:

return 0

if len(s) == 1:

return 1

char\_index = [-1] \* 128

left = 0

max\_len = 0

for right in range(len(s)):

curr\_char = ord(s[right])

left = max(left, char\_index[curr\_char] + 1)

curr\_len = right - left + 1

if curr\_len > max\_len:

max\_len = curr\_len

char\_index[curr\_char] = right

return max\_len