# **PYTHON –LEETCODE**

**1.Subsets:**

class Solution(object):

def subsetsWithDup(self, nums):

res = []

nums.sort()

self.backtrack(nums, 0, [],res)

return res

def backtrack(self, nums, index,path, res):

res.append(path[:])

for i in range(index, len(nums)):

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

continue

path.append(nums[i])

self.backtrack(nums, i + 1, path, res)

path.pop()

2.**Decode Ways:**

class Solution(object):

def numDecodings(self, s):

"""

:type s: str

:rtype: int

"""

if not s or s[0] == '0':

return 0

n = len(s)

v = [0] \*(n+1)

v[0] =1

v[1] =1

for i in range(2,n + 1):

if s[i-1] != '0':

v[i] += v[i - 1]

two\_digits = int(s[i - 2:i])

if 10 <= two\_digits <= 26:

v[i] +=v[i-2]

return v[n]

**3.Binary tree in order traversal:**

# Definition for a binary tree node.

# class TreeNode(object):

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution(object):

def inorderTraversal(self, root):

"""

:type root: TreeNode

:rtype: List[int]

"""

result = []

def inorder(node):

if node is None:

return

inorder(node.left)

result.append(node.val)

inorder(node.right)

inorder(root)

return result

**4.Merge sorted Array:**

class Solution(object):

def merge(self, nums1, m, nums2, n):

# Remove the extra elements in nums1

n = len(nums1) - m

for i in range(n):

nums1.pop()

# Extend nums1 with nums2 and sort

nums1.extend(nums2)

nums1.sort()

**5. Decode Ways:**

class Solution(object):

def numDecodings(self, s):

"""

:type s: str

:rtype: int

"""

if not s or s[0] == '0':

return 0

n = len(s)

v = [0] \*(n+1)

v[0] =1

v[1] =1

for i in range(2,n + 1):

if s[i-1] != '0':

v[i] += v[i - 1]

two\_digits = int(s[i - 2:i])

if 10 <= two\_digits <= 26:

v[i] +=v[i-2]

return v[n]

**6. Same tree:**

# Definition for a binary tree node.

# class TreeNode(object):

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution(object):

def isSameTree(self, p, q):

if p == None and q == None:

return True

if p == None or q == None or p.val != q.val:

return False

return self.isSameTree(p.left, q.left) and self.isSameTree(p.right, q.right)

**7.Two sum:**

class Solution(object):

def twoSum(self, nums, target):

for i in range(len(nums)):

for j in range(i+1,len(nums)):

if nums[i]+nums[j]==target:

return [i,j]

**8.Reverse integer:**

class Solution:

def reverse(self, x):

if -2\*\*31 <= x <= 2\*\*31:

rem = 0

if x < 0:

sign = -1

x = abs(x)

else:

sign = 1

while x > 0:

a = x % 10

rem = rem \* 10 + a

x = x // 10

res=rem\*sign

if -2\*\*31 <= res <= 2\*\*31:

return rem \* sign

else:

return 0

**9. palindrome Number:**

class Solution(object):

def isPalindrome(self, x):

"""

:type x: int

:rtype: bool

"""

if x < 0:

return False

original, reversed\_num = x, 0

while x > 0:

reversed\_num = reversed\_num \* 10 + x % 10

x //= 10

return original == reversed\_num

**10.Longest Palindromic Substring:**

class Solution(object):

def longestPalindrome(self, s):

"""

:type s: str

:rtype: str

"""

res = ""

for i in xrange(len(s)):

tmp = self.helper(s, i, i)

if len(tmp) > len(res):

res = tmp

tmp = self.helper(s, i, i+1)

if len(tmp) > len(res):

res = tmp

return res

def helper(self, s, l, r):

while l>=0 and r<len(s) and s[l]==s[r]:

l-=1;r+=1

return s[l+1:r]