**1.3 SUM CLOSEST**

class Solution(object):

    def threeSumClosest(self, nums, target):

        """

        :type nums: List[int]

        :type target: int

        :rtype: int

        """

        nums.sort()

        closest\_sum = float('inf')

        min\_diff = float('inf')

        for i in range(len(nums) - 2):

            left, right = i + 1, len(nums) - 1

            while left < right:

                current\_sum = nums[i] + nums[left] + nums[right]

                current\_diff = abs(current\_sum - target)

                if current\_diff < min\_diff:

                    min\_diff = current\_diff

                    closest\_sum = current\_sum

               if current\_sum < target:

                    left += 1

                else:

                    right -= 1

        return closest\_sum

**2.SORT LIST:**

class Solution(object):

    def sortList(self, head):

        if not head or not head.next:

            return head

        fast, slow = head.next, head

        while fast and fast.next:

            fast = fast.next.next

            slow = slow.next

        start = slow.next

        slow.next = None

        l, r = self.sortList(head), self.sortList(start)

        return self.merge(l, r)

    def merge(self, l, r):

        if not l or not r:

            return l or r

        dummy = p = ListNode(0)

        while l and r:

            if l.val < r.val:

                p.next = l

                l = l.next

            else:

                p.next = r

                r = r.next

            p = p.next

        p.next = l or r

        return dummy.next

3.**REVERSE WORD IN STRING:**

class Solution(object):

    def reverseWords(self, s):

        """

        :type s: str

        :rtype: str

        """

        return ' '.join(reversed(s.split()))

**4.RECTANGLE AREA:**

class Solution(object):

    def computeArea(self, ax1, ay1, ax2, ay2, bx1, by1, bx2, by2):

        area1 = (ax2 - ax1) \* (ay2 - ay1)

        area2 = (bx2 - bx1) \* (by2 - by1)

        overlap\_width = max(0, min(ax2, bx2) - max(ax1, bx1))

        overlap\_height = max(0, min(ay2, by2) - max(ay1, by1))

        overlap\_area = overlap\_width \* overlap\_height

        total\_area = area1 + area2 - overlap\_area

        return total\_area

**5.STREAM OF CHARACTERS:**

 def nextPermutation(self, nums):

        for i in range(len(nums)-1, 0, -1):

            # find the index of the last peak

            if nums[i - 1] < nums[i]:

                nums[i:] = sorted(nums[i:])

                # get the index before the last peak

                j = i - 1

                # swap the pre-last peak index with the value just large than it

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

                    if nums[j] < nums[k]:

                        nums[k], nums[j] = nums[j], nums[k]

                        return nums

        return nums.reverse()

**6.EDIT DISTANCE:**

class Solution(object):

def minDistance(self, word1, word2):

"""

:type word1: str

:type word2: str

:rtype: int

"""

m, n = len(word1), len(word2)

dp = [[0] \* (n + 1) for \_ in range(m + 1)]

for i in range(m + 1):

dp[i][0] = i

for j in range(n + 1):

dp[0][j] = j

for i in range(1, m + 1):

for j in range(1, n + 1):

if word1[i - 1] == word2[j - 1]:

dp[i][j] = dp[i - 1][j - 1]

else:

dp[i][j] = min(dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1]) + 1

return dp[m][n]

**7.GRAY CODE:**

class Solution {

public:

vector<int> grayCode(int n) {

vector<int> result;

result.push\_back(0);

for (int i = 1; i <= n; i++) {

int prevLength = result.size();

int mask = 1 << (i - 1);

for (int j = prevLength - 1; j >= 0; j--) {

result.push\_back(mask + result[j]);

}

}

return result;

}

};

**8.RECURSIVE APPROACH WITH EASY STEPS:**

class Solution:

def isSameTree(self, p, q):

# If both nodes are None, they are identical

if p is None and q is None:

return True

# If only one of the nodes is None, they are not identical

if p is None or q is None:

return False

# Check if values are equal and recursively check left and right subtrees

if p.val == q.val:

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

# Values are not equal, they are not identical

return False

**9.SYMMENTRIC TREE:**

class Solution(object):

def isMirror(self, left, right):

if not left and not right:

return True

if not left or not right:

return False

return left.val == right.val and self.isMirror(left.left, right.right) and self.isMirror(left.right, right.left)

def isSymmetric(self, root):

if not root:

return True

return self.isMirror(root.left, root.right)

**10.WORD LADDER:**

class Solution:

# @param {string} beginWord

# @param {string} endWord

# @param {set<string>} wordDict

# @return {integer}

def ladderLength(self, beginWord, endWord, wordDict):

length = 2

front, back = set([beginWord]), set([endWord])

wordDict.discard(beginWord)

while front:

# generate all valid transformations

front = wordDict & (set(word[:index] + ch + word[index+1:] for word in front

for index in range(len(beginWord)) for ch in 'abcdefghijklmnopqrstuvwxyz'))

if front & back:

# there are common elements in front and back, done

return length

length += 1

if len(front) > len(back):

# swap front and back for better performance (fewer choices in generating nextSet)

front, back = back, front

# remove transformations from wordDict to avoid cycle

wordDict -= front

return 0