**ASSIGNMENT -2**

**1**.def maxArea(A, Len) :

area = 0

for i in range(Len) :

for j in range(i + 1, Len) :

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

return area

a = [ 1, 5, 4, 3 ]

b = [ 3, 1, 2, 4, 5 ]

len1 = len(a)

print(maxArea(a, len1))

len2 = len(b)

print(maxArea(b, len2))

**2**.def value(r):

if (r == 'I'):

return 1

if (r == 'V'):

return 5

if (r == 'X'):

return 10

if (r == 'L'):

return 50

if (r == 'C'):

return 100

if (r == 'D'):

return 500

if (r == 'M'):

return 1000

return -1

def romanToDecimal(str):

res = 0

i = 0

while (i < len(str)):

s1 = value(str[i])

if (i + 1 < len(str)):

s2 = value(str[i + 1])

if (s1 >= s2):

res = res + s1

i = i + 1

else:

res = res + s2 - s1

i = i + 2

else:

res = res + s1

i = i + 1

return res

print("Integer form of Roman Numeral is")

print(romanToDecimal("MCMIV"))

**3**.def romanToInt(s):

roman = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}

total = 0

prev\_value = 0

for char in s:

value = roman[char]

if value > prev\_value:

total += value - 2 \* prev\_value

else:

total += value

prev\_value = value

return total

s = "MCMXCIV"

print(romanToInt(s))

**4**.def longestCommonPrefix(strs):

if not strs:

return ""

shortest = min(strs, key=len)

for i, char in enumerate(shortest):

for other in strs:

if other[i] != char:

return shortest[:i]

return shortest

strs = ["flower","flow","flight"]

print(longestCommonPrefix(strs))

**5**.def threeSum(nums):

nums.sort()

res = []

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

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

continue

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

while left < right:

s = nums[i] + nums[left] + nums[right]

if s < 0:

left += 1

elif s > 0:

right -= 1

else:

res.append((nums[i], nums[left], nums[right]))

while left < right and nums[left] == nums[left + 1]:

left += 1

while left < right and nums[right] == nums[right - 1]:

right -= 1

left += 1

right -= 1

return res

nums = [-1,0,1,2,-1,-4]

print(threeSum(nums))

**6**.def three\_sum\_closest(nums, target):

nums.sort()

closest\_sum = float('inf')

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

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

while left < right:

total = nums[i] + nums[left] + nums[right]

if abs(target - total) < abs(target - closest\_sum):

closest\_sum = total

if total < target:

left += 1

elif total > target:

right -= 1

else:

return total

return closest\_sum

**7**.def threeSumClosest(nums, target):

nums.sort()

closest\_sum = 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]

if abs(current\_sum - target) < abs(closest\_sum - target):

closest\_sum = current\_sum

if current\_sum < target:

left += 1

elif current\_sum > target:

right -= 1

else:

return current\_sum

return closest\_sum

nums = [-1, 2, 1, -4]

target = 1

print(threeSumClosest(nums, target))

**8**.def letterCombinations(digits):

if not digits:

return []

phone = {

'2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',

'6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'

}

def backtrack(index, path):

if len(path) == len(digits):

combinations.append("".join(path))

return

for letter in phone[digits[index]]:

path.append(letter)

backtrack(index + 1, path)

path.pop()

combinations = []

backtrack(0, [])

return combinations

digits = "23"

print(letterCombinations(digits))

class ListNode:

def \_init\_(self, val=0, next=None):

self.val = val

self.next = next

**9**.class ListNode:

def \_init\_(self, val=0, next=None):

self.val = val

self.next = next

def removeNthFromEnd(head, n):

dummy = ListNode(0)

dummy.next = head

first = dummy

second = dummy

for \_ in range(n + 1):

first = first.next

while first is not None:

first = first.next

second = second.next

second.next = second.next.next

return dummy.next

def create\_linked\_list(arr):

head = ListNode(arr[0])

current = head

for val in arr[1:]:

current.next = ListNode(val)

current = current.next

return head

head = create\_linked\_list([1, 2, 3, 4, 5])

n = 2

new\_head = removeNthFromEnd(head, n)

def linked\_list\_to\_list(node):

result = []

while node:

result.append(node.val)

node = node.next

return result

print(linked\_list\_to\_list(new\_head))

**10**.def isValid(s):

stack = []

mapping = {")": "(", "}": "{", "]": "["}

for char in s:

if char in mapping:

top\_element = stack.pop() if stack else '#'

if mapping[char] != top\_element:

return False

else:

stack.append(char)

return not stack

s = "()[]{}"

print(isValid(s))