**Greedy\_Algorithms\_Assignment**

1). You are given an array of integers and an integer k. Your task is to select k elements from the array such that their sum is maximized:-

Solution :-

def max\_sum\_with\_k\_elements(array, k):

    array.sort(reverse=True)

    max\_sum = sum(array[:k])

    return max\_sum

# Example usage:

array = [10, 7, 2, 4, 6, 8, 9]

k = 3

result = max\_sum\_with\_k\_elements(array, k)

print(f"Maximum sum of {k} elements: {result}")

2). Lemonade change:-

Solution:-

class Solution:

    def lemonadeChange(self, bills: list[int]) -> bool:

        five\_dollar\_count = ten\_dollar\_count = 0

        for bill in bills:

            if bill == 5:

                five\_dollar\_count += 1

            elif bill == 10:

                if five\_dollar\_count > 0:

                    five\_dollar\_count -= 1

                    ten\_dollar\_count += 1

                else:

                    return False

            else:

                if ten\_dollar\_count > 0 and five\_dollar\_count > 0:

                    ten\_dollar\_count -= 1

                    five\_dollar\_count -= 1

                elif five\_dollar\_count >= 3:

                    five\_dollar\_count -= 3

                else:

                    return False

        return True

# Example usage:

bills1 = [5, 5, 5, 10, 20]

print(lemonadeChange(bills1))  # Output: True

bills2 = [5, 5, 10, 10, 20]

print(lemonadeChange(bills2))  # Output: False

3). Check if possible to survive on island:-

Solution:-

def minimumDays(S, N, M):

    if ((N \* 6) < (M \* 7)) and S > 6:

        return -1

    else:

        days = (M \* S) // N

        if (M \* S) % N != 0:

            days += 1

        return days

# Example usage:

S1, N1, M1 = 10, 16, 2

print(minimumDays(S1, N1, M1))  # Output: 2

S2, N2, M2 = 10, 20, 30

print(minimumDays(S2, N2, M2))  # Output: -1

4). Shortest job first:-

Solution:-

def solve(bt):

    n = len(bt)

    bt.sort()

    waiting\_time = 0

    total\_waiting\_time = 0

    for i in range(n):

        total\_waiting\_time += waiting\_time

        waiting\_time += bt[i]

    average\_waiting\_time = total\_waiting\_time // n

    return average\_waiting\_time

# Example usage:

bt1 = [4, 3, 7, 1, 2]

print(solve(bt1))  # Output: 4

bt2 = [1, 2, 3, 4]

print(solve(bt2))  # Output: 2

5). Fractional Knapsack:-

Solution:-

class Item:

    def \_\_init\_\_(self, weight, value):

        self.weight = weight

        self.value = value

        self.ratio = value / weight

def fractional\_knapsack(items, capacity):

    items.sort(key=lambda x: x.ratio, reverse=True)

    total\_value = 0

    remaining\_capacity = capacity

    for item in items:

        if remaining\_capacity <= 0:

            break

        fraction = min(1, remaining\_capacity / item.weight)

        total\_value += fraction \* item.value

        remaining\_capacity -= fraction \* item.weight

    return round(total\_value, 2)

# Example usage:

items = [

    Item(10, 60),

    Item(20, 100),

    Item(30, 120)

]

capacity = 50

print(f"Maximum value in knapsack: {fractional\_knapsack(items, capacity)}")

items = [

    Item(10, 60),

    Item(20, 100)

]

capacity = 50

print(f"Maximum value in knapsack: {fractional\_knapsack(items, capacity)}")

6). Maximum units on a truck:-

Solution:-

def maximumUnits(boxTypes, truckSize):

    boxTypes.sort(key=lambda x: x[1], reverse=True)

    total\_units = 0

    for num\_boxes, units\_per\_box in boxTypes:

        if truckSize >= num\_boxes:

            total\_units += num\_boxes \* units\_per\_box

            truckSize -= num\_boxes

        else:

            total\_units += truckSize \* units\_per\_box

            break

    return total\_units

# Example usage:

boxTypes1 = [[1, 3], [2, 2], [3, 1]]

truckSize1 = 4

print(maximumUnits(boxTypes1, truckSize1))  # Output: 8

boxTypes2 = [[5, 10], [2, 5], [4, 7], [3, 9]]

truckSize2 = 10

print(maximumUnits(boxTypes2, truckSize2))  # Output: 91

7). Chocolate distribution:-

Solution:-

def findMinDiff(A, N, M):

    if M == 0 or N == 0:

        return 0

    A.sort()

    if N < M:

        return -1

    min\_diff = float('inf')

    for i in range(N - M + 1):

        diff = A[i + M - 1] - A[i]

        min\_diff = min(min\_diff, diff)

    return min\_diff

# Example usage:

A1 = [3, 4, 1, 9, 56, 7, 9, 12]

N1, M1 = 8, 5

print(findMinDiff(A1, N1, M1))  # Output: 6

A2 = [7, 3, 2, 4, 9, 12, 56]

N2, M2 = 7, 3

print(findMinDiff(A2, N2, M2))  # Output: 2

8). Shop in candy store:-

Solution:-

def candyStore(candies, N, K):

    candies.sort()

    min\_cost = 0

    max\_cost = 0

    i = 0

    j = N - 1

    while i <= j:

        min\_cost += candies[i]

        i += K + 1

    i = N - 1

    j = 0

    while i >= j:

        max\_cost += candies[i]

        i -= K + 1

    return min\_cost, max\_cost

# Example usage:

candies1 = [3, 2, 1, 4]

N1, K1 = 4, 2

min\_cost1, max\_cost1 = candyStore(candies1, N1, K1)

print(min\_cost1, max\_cost1)  # Output: 5 5

candies2 = [3, 2, 1, 4, 5]

N2, K2 = 5, 4

min\_cost2, max\_cost2 = candyStore(candies2, N2, K2)

print(min\_cost2, max\_cost2)  # Output: 1 5

9). Assign cookies:-

Solution:-

def findContentChildren(g, s):

    g.sort()

    s.sort()

    content\_children = 0

    i, j = 0, 0

    while i < len(g) and j < len(s):

        if s[j] >= g[i]:

            content\_children += 1

            i += 1

        j += 1

    return content\_children

# Example usage:

g1 = [1, 2, 3]

s1 = [1, 1]

print(findContentChildren(g1, s1))  # Output: 1

g2 = [1, 2]

s2 = [1, 2, 3]

print(findContentChildren(g2, s2))  # Output: 2

10). N meetings in one room:-

Solution:-

def maxMeetings(n, start, end):

    meetings = list(zip(start, end))

    meetings.sort(key=lambda x: x[1])

    count = 0

    last\_end\_time = 0

    for meeting in meetings:

        start\_time, end\_time = meeting

        if start\_time >= last\_end\_time:

            count += 1

            last\_end\_time = end\_time

    return count

# Example usage:

n1 = 6

start1 = [1, 3, 0, 5, 8, 5]

end1 = [2, 4, 6, 7, 9, 9]

print(maxMeetings(n1, start1, end1))  # Output: 4

n2 = 3

start2 = [10, 12, 20]

end2 = [20, 25, 30]

print(maxMeetings(n2, start2, end2))  # Output: 2

11). find maximum meetings in one room:-

Solution:-

class meeting:

  def \_\_init\_\_(self, start, end, pos):

    self.start = start

    self.end = end

    self.pos = pos

def maxMeeting(l, N):

  ans = []

  l.sort(key=lambda x: x.end)

  ans.append(l[0].pos)

  time\_limit = l[0].end

  for i in range(1, N):

    if l[i].start > time\_limit:

      ans.append(l[i].pos)

      time\_limit = l[i].end

  for i in ans:

    print(i + 1, end=" ")

  print()

if \_\_name\_\_ == '\_\_main\_\_':

  s = [1, 3, 0, 5, 8, 5]

  f = [2, 4, 6, 7, 9, 9]

  N = len(s)

  l = []

  for i in range(N):

    l.append(meeting(s[i], f[i], i))

  maxMeeting(l, N)

12). Non overlapping intervals:-

Solution:-

def eraseOverlapIntervals(intervals):

    if not intervals:

        return 0

    intervals.sort(key=lambda x: x[1])

    count = 0

    last\_end = intervals[0][0] - 1

    for interval in intervals:

        start, end = interval

        if start < last\_end:

            count += 1

        else:

            last\_end = end

    return count

# Example usage:

intervals1 = [[1,2],[2,3],[3,4],[1,3]]

print(eraseOverlapIntervals(intervals1))  # Output: 1

intervals2 = [[1,2],[1,2],[1,2]]

print(eraseOverlapIntervals(intervals2))  # Output: 2

intervals3 = [[1,2],[2,3]]

print(eraseOverlapIntervals(intervals3))  # Output: 0

13). insert interval:-

Solution:-

class Solution:

    def insert(self, intervals, newInterval):

        result = []

        i = 0

        n = len(intervals)

        while i < n and intervals[i][1] < newInterval[0]:

            result.append(intervals[i])

            i += 1

        while i < n and intervals[i][0] <= newInterval[1]:

            newInterval[0] = min(newInterval[0], intervals[i][0])

            newInterval[1] = max(newInterval[1], intervals[i][1])

            i += 1

        result.append(newInterval)

        while i < n:

            result.append(intervals[i])

            i += 1

        return result

# Example usage:

intervals1 = [[1, 3], [6, 9]]

newInterval1 = [2, 5]

print(Solution().insert(intervals1, newInterval1))  # Output: [[1, 5], [6, 9]]

intervals2 = [[1, 2], [3, 5], [6, 7], [8, 10], [12, 16]]

newInterval2 = [4, 8]

print(Solution().insert(intervals2, newInterval2))  # Output: [[1, 2], [3, 10], [12, 16]]

14). Merge intervals:-

Solution:-

class Solution:

    def merge(self, intervals):

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

        merged = []

        for interval in intervals:

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

                merged.append(interval)

            else:

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

        return merged

# Example usage:

intervals1 = [[1, 3], [2, 6], [8, 10], [15, 18]]

print(Solution().merge(intervals1))  # Output: [[1, 6], [8, 10], [15, 18]]

intervals2 = [[1, 4], [4, 5]]

print(Solution().merge(intervals2))  # Output: [[1, 5]]

15). Job sequencing problem:-

Solution:-

class Solution:

    def JobScheduling(self, N, Jobs):

        Jobs.sort(key=lambda x: x[1])

        count = 0

        max\_profit = 0

        time\_slots = [False] \* (N + 1)

        for job in Jobs:

            job\_id, deadline, profit = job

            for i in range(deadline, 0, -1):

                if not time\_slots[i]:

                    time\_slots[i] = True

                    count += 1

                    max\_profit += profit

                    break

        return [count, max\_profit]

# Example usage:

N1 = 4

Jobs1 = [(1, 4, 20), (2, 1, 10), (3, 1, 40), (4, 1, 30)]

print(Solution().JobScheduling(N1, Jobs1))  # Output: [2, 30]

N2 = 5

Jobs2 = [(1, 2, 100), (2, 1, 19), (3, 2, 27), (4, 1, 25), (5, 1, 15)]

print(Solution().JobScheduling(N2, Jobs2))  # Output: [2, 119]

16). Minimum cost of ropes:-

Solution:-

class Solution:

    def minCost(self, arr, n):

        import heapq

        heapq.heapify(arr)

        total\_cost = 0

        while len(arr) > 1:

            smallest1 = heapq.heappop(arr)

            smallest2 = heapq.heappop(arr)

            combined\_length = smallest1 + smallest2

            total\_cost += combined\_length

            heapq.heappush(arr, combined\_length)

        return total\_cost

# Example usage:

arr1 = [4, 3, 2, 6]

n1 = len(arr1)

print(Solution().minCost(arr1, n1))  # Output: 29

arr2 = [4, 2, 7, 6, 9]

n2 = len(arr2)

print(Solution().minCost(arr2, n2))  # Output: 62

17). Jump game:-

Solution:-

class Solution:

    def canJump(self, nums):

        max\_reach = 0

        for i in range(len(nums)):

            if i > max\_reach:

                return False

            max\_reach = max(max\_reach, i + nums[i])

        return True

# Example usage:

nums1 = [2, 3, 1, 1, 4]

print(Solution().canJump(nums1))  # Output: True

nums2 = [3, 2, 1, 0, 4]

print(Solution().canJump(nums2))  # Output: False

18). Jump game 2:-

Solution:-

class Solution:

    def canJump(self, nums):

        max\_reach = 0

        jumps = 0

        for i in range(len(nums)):

            if i > max\_reach:

                return False

            max\_reach = max(max\_reach, i + nums[i])

            jumps += 1

        return True

# Example usage:

nums1 = [2, 3, 1, 1, 4]

print(Solution().canJump(nums1))  # Output: True

nums2 = [2, 3, 0, 1, 4]

print(Solution().canJump(nums2))  # Output: True

class Solution:

    def canJump(self, nums):

        max\_reach = 0

        jumps = 0

        for i in range(len(nums)):

            if i > max\_reach:

                return False

            max\_reach = max(max\_reach, i + nums[i])

            jumps += 1

        return True

# Example usage:

nums1 = [2, 3, 1, 1, 4]

print(Solution().canJump(nums1))  # Output: True

nums2 = [2, 3, 0, 1, 4]

print(Solution().canJump(nums2))  # Output: True

19). Minimum number of platforms required for a railway/bus:-

Solution:-

def findPlatform(arr, dep, n):

    plat\_needed = 1

    result = 1

    for i in range(n):

        plat\_needed = 1

        for j in range(n):

            if i != j:

                if (arr[i] >= arr[j] and dep[j] >= arr[i]):

                    plat\_needed += 1

        result = max(result, plat\_needed)

    return result

# Driver code

def main():

    arr = [100, 300, 500]

    dep = [900, 400, 600]

    n = len(arr)

    print("{}".format(

        findPlatform(arr, dep, n)))

if \_\_name\_\_ == '\_\_main\_\_':

    main()

20). Distribution Candy:-

Solution:-

class Solution:

    def minCandies(self, ratings):

        n = len(ratings)

        candies = [1] \* n

        for i in range(1, n):

            if ratings[i] > ratings[i - 1]:

                candies[i] = candies[i - 1] + 1

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

            if ratings[i] > ratings[i + 1] and candies[i] <= candies[i + 1]:

                candies[i] = candies[i + 1] + 1

        return sum(candies)

# Example usage:

ratings1 = [1, 0, 2]

print(Solution().minCandies(ratings1))  # Output: 5

ratings2 = [1, 2, 2]

print(Solution().minCandies(ratings2))  # Output: 4

21). Huffman coding:-  
Solution:-

import heapq

class Node:

    def \_\_init\_\_(self, symbol=None, frequency=None):

        self.symbol = symbol

        self.frequency = frequency

        self.left = None

        self.right = None

    def \_\_lt\_\_(self, other):

        return self.frequency < other.frequency

def build\_huffman\_tree(chars, freq):

    priority\_queue = [Node(char, f) for char, f in zip(chars, freq)]

    heapq.heapify(priority\_queue)

    while len(priority\_queue) > 1:

        left\_child = heapq.heappop(priority\_queue)

        right\_child = heapq.heappop(priority\_queue)

        merged\_node = Node(frequency=left\_child.frequency + right\_child.frequency)

        merged\_node.left = left\_child

        merged\_node.right = right\_child

        heapq.heappush(priority\_queue, merged\_node)

    return priority\_queue[0]

def generate\_huffman\_codes(node, code="", huffman\_codes={}):

    if node is not None:

        if node.symbol is not None:

            huffman\_codes[node.symbol] = code

        generate\_huffman\_codes(node.left, code + "0", huffman\_codes)

        generate\_huffman\_codes(node.right, code + "1", huffman\_codes)

    return huffman\_codes

def huffmanCodes(S, f, N):

    root = build\_huffman\_tree(S, f)

    huffman\_codes = generate\_huffman\_codes(root)

    return list(huffman\_codes.values())

# Example usage:

S = "abcdef"

f = [5, 9, 12, 13, 16, 45]

N = len(S)

print(huffmanCodes(S, f, N))  # Output: ['0', '100', '101', '1100', '1101', '111']

22). Assign mice to holes:-

Solution:-

class Solution:

    def assignMiceHoles(self, N, M, H):

        M.sort()

        H.sort()

        max\_time = 0

        for i in range(N):

            max\_time = max(max\_time, abs(M[i] - H[i]))

        return max\_time

# Example usage:

N1 = 3

M1 = [4, -4, 2]

H1 = [4, 0, 5]

print(Solution().assignMiceHoles(N1, M1, H1))  # Output: 4

N2 = 2

M2 = [4, 2]

H2 = [1, 7]

print(Solution().assignMiceHoles(N2, M2, H2))  # Output: 3

23). Policemen catchs thieves:-

Solution:-  
def policeThief(arr, n, k):

    i = 0

    l = 0

    r = 0

    res = 0

    thi = []

    pol = []

    while i < n:

        if arr[i] == 'P':

            pol.append(i)

        elif arr[i] == 'T':

            thi.append(i)

        i += 1

    while l < len(thi) and r < len(pol):

        if (abs(thi[l] - pol[r]) <= k):

            res += 1

            l += 1

            r += 1

        elif thi[l] < pol[r]:

            l += 1

        else:

            r += 1

    return res

# Driver program

if \_\_name\_\_ == '\_\_main\_\_':

    arr1 = ['P', 'T', 'T', 'P', 'T']

    k = 2

    n = len(arr1)

    print(("Maximum thieves caught: {}".

        format(policeThief(arr1, n, k))))

    arr2 = ['T', 'T', 'P', 'P', 'T', 'P']

    k = 2

    n = len(arr2)

    print(("Maximum thieves caught: {}".

        format(policeThief(arr2, n, k))))

    arr3 = ['P', 'T', 'P', 'T', 'T', 'P']

    k = 3

    n = len(arr3)

    print(("Maximum thieves caught: {}".

        format(policeThief(arr3, n, k))))

24). K items with the maximum sums:-  
Solution:-

class Solution:

    def kItemsWithMaximumSum(self, numOnes: int, numZeros: int, numNegOnes: int, k: int) -> int:

        return min(numOnes, k) + max(0, k - numOnes - numZeros) - max(0, k - numOnes)

# Example usage:

numOnes1, numZeros1, numNegOnes1, k1 = 3, 2, 0, 2

print(Solution().kItemsWithMaximumSum(numOnes1, numZeros1, numNegOnes1, k1))  # Output: 2

numOnes2, numZeros2, numNegOnes2, k2 = 3, 2, 0, 4

print(Solution().kItemsWithMaximumSum(numOnes2, numZeros2, numNegOnes2, k2))  # Output: 2

25). Valid parenthesis checker:-

Solution:-

class Solution:

    def isValid(self, s: str) -> bool:

        open\_stack = []

        asterisk\_stack = []

        for char in s:

            if char == '(':

                open\_stack.append(char)

            elif char == '\*':

                asterisk\_stack.append(char)

            elif char == ')':

                if open\_stack:

                    open\_stack.pop()

                elif asterisk\_stack:

                    asterisk\_stack.pop()

                else:

                    return False

        while open\_stack:

            if not asterisk\_stack:

                return False

            open\_stack.pop()

            asterisk\_stack.pop()

        return True

# Example usage:

s1 = "()"

print(Solution().isValid(s1))  # Output: True

s2 = "(\*)"

print(Solution().isValid(s2))  # Output: True

s3 = "(\*))"

print(Solution().isValid(s3))  # Output: True

26). Coin Change:-  
Solution:-

import sys

def minCoins(coins, m, V):

    if (V == 0):

        return 0

    res = sys.maxsize

    for i in range(0, m):

        if (coins[i] <= V):

            sub\_res = minCoins(coins, m, V-coins[i])

            if (sub\_res != sys.maxsize and sub\_res + 1 < res):

                res = sub\_res + 1

    return res

# Driver program to test above function

coins = [9, 6, 5, 1]

m = len(coins)

V = 11

print("Minimum coins required is",minCoins(coins, m, V))

27). Minimum platforms:-

Solution:-

class Solution:

    def minPlatforms(self, arr, dep, n):

        events = [(time, 1) for time in arr] + [(time, -1) for time in dep]

        events.sort()

        platforms\_needed = 0

        result = 0

        for \_, event\_type in events:

            platforms\_needed += event\_type

            result = max(result, platforms\_needed)

        return result

# Example usage:

n1 = 6

arr1 = [900, 940, 950, 1100, 1500, 1800]

dep1 = [910, 1200, 1120, 1130, 1900, 2000]

print(Solution().minPlatforms(arr1, dep1, n1))  # Output: 3

n2 = 3

arr2 = [900, 1235, 1100]

dep2 = [1000, 1240, 1200]

print(Solution().minPlatforms(arr2, dep2, n2))  # Output: 1

n3 = 3

arr3 = [1000, 935, 1100]

dep3 = [1200, 1240, 1130]

print(Solution().minPlatforms(arr3, dep3, n3))  # Output: 3

28). Page faults in LRU:-

Solution:-

class Solution:

    def pageFaults(self, N, C, pages):

        memory = set()

        recent\_pages = []

        page\_faults = 0

        for page in pages:

            if page not in memory:

                page\_faults += 1

                if len(memory) == C:

                    removed\_page = recent\_pages.pop(0)

                    memory.remove(removed\_page)

                memory.add(page)

            else:

                recent\_pages.remove(page)

            recent\_pages.append(page)

        return page\_faults

# Example usage:

N1, C1 = 9, 4

pages1 = [5, 0, 1, 3, 2, 4, 1, 0, 5]

print(Solution().pageFaults(N1, C1, pages1))  # Output: 8

N2, C2 = 3, 4

pages2 = [5, 0, 1]

print(Solution().pageFaults(N2, C2, pages2))  # Output: 3

29). Seats:-

Solution:-

MOD = 10\*\*9 + 7

def minJumps(seats):

    position = []

    count = 0

    lenn = len(seats)

    for i in range(lenn):

        if (seats[i] == 'x'):

            position.append(i - count)

            count += 1

    if (count == lenn or count == 0):

        return 0

    med\_index = (count - 1) // 2

    med\_val = position[med\_index]

    ans = 0

    for i in range(len(position)):

        ans = (ans % MOD +

            abs(position[i] - med\_val)

            % MOD) % MOD

    return ans % MOD

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

    S = "....x..xx...x.."

    print(minJumps(S))

30). Gas station:-

Solution:-

class Solution:

    def canCompleteCircuit(self, gas, cost):

        total\_gas = -1

        starting\_index = -1

        if sum(gas) - sum(cost) < 0:

            return -1

        for i in range(len(gas)):

            fuel = gas[i]

            isAmple = True

            for j in range(1, len(gas)):

                currStation = (i + j) % len(gas)

                nextStation = (currStation + 1) % len(gas)

                fuel -= cost[currStation]

                if fuel < 0:

                    isAmple = False

                    break

                fuel += gas[nextStation]

            if isAmple:

                return i

        return -1

# Example usage:

gas1 = [1, 2, 3, 4, 5]

cost1 = [3, 4, 5, 1, 2]

print(Solution().canCompleteCircuit(gas1, cost1))  # Output: 2

gas2 = [2, 3, 4]

cost2 = [3, 4, 3]

print(Solution().canCompleteCircuit(gas2, cost2))  # Output: -1