GREEDY ALGORITHM ASSIGNMENT

Q1) 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.

Code:

```
def findWays(arr, N, K):
    arr.sort(reverse=True)  # Sort the array in descending order
    p, q = 0, 0

    for i in range(K):
        if arr[i] == arr[K - 1]:
            p += 1

    for i in range(N):
        if arr[i] == arr[K - 1]:
            q += 1

    def C(n, r):
        # Calculate nCr
        res = 1
        for i in range(2, n + 1):
            res *= i
        return res // (fact(r) * fact(n - r))

    ans = C(q, p)
    return ans

# Example usage
arr = [2, 3, 4, 5, 2, 2]
N = len(arr)
K = 4
print(findWays(arr, N, K))  # Output: 3
```

Q2) Given an integer array bills where bills[i] is the bill the ith customer pays, return true if you can provide every customer with the correct change, or false otherwise.

Code:

```
def lemonadeChange(bills):
    count5 = 0
    count10 = 0
```

```
    for bill in bills:
        if bill == 5:
            count5 += 1
        elif bill == 10:
            if count5 < 1:
                return False
            count5 -= 1
            count10 += 1
        elif bill == 20:
            if count10 >= 1 and count5 >= 1:
                count10 -= 1
                count5 -= 1
            elif count5 >= 3:
                count5 -= 3
            else:
                return False

    return True

# Example usage
bills1 = [5, 5, 5, 10, 20]
bills2 = [5, 5, 10, 10, 20]

print(lemonadeChange(bills1))  # Output: True
print(lemonadeChange(bills2))  # Output: False
```

Q3) Check if possible to survive on island.

Code:

```
def minimumDays(S, N, M):
    total_food_units = N * S
    if total_food_units >= M:
        return M
    else:
        return -1

# Example usage
S1, N1, M1 = 10, 16, 2
S2, N2, M2 = 10, 20, 30

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

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

Q4)Shortest job first.

Code:

```
def solve(bt):
    n = len(bt)
    bt.sort()  # Sort burst times in ascending order

    # Calculate waiting time for each process
    waiting_time = [0] * n
    for i in range(1, n):
        waiting_time[i] = bt[i - 1] + waiting_time[i - 1]

    # Compute average waiting time
    total_waiting_time = sum(waiting_time)
    average_waiting_time = total_waiting_time // n

    return average_waiting_time

# Example usage
bt1 = [4, 3, 7, 1, 2]
bt2 = [1, 2, 3, 4]

print(solve(bt1))  # Output: 4
print(solve(bt2))  # Output: 2
```

Q5)Fractional Knapsack.

Code:

```
def fractional_knapsack(n, w, value, weight):
    # Calculate value-to-weight ratio for each item
    ratio = [(value[i] / weight[i], i) for i in range(n)]
    ratio.sort(reverse=True)  # Sort by ratio in descending order

    max_value = 0.0
    for _, i in ratio:
        if w >= weight[i]:
            max_value += value[i]
            w -= weight[i]
        else:
            max_value += (w / weight[i]) * value[i]
```

```
            break

    return max_value

# Example usage
n1, w1, value1, weight1 = 3, 50, [60, 100, 120], [10, 20, 30]
n2, w2, value2, weight2 = 2, 50, [60, 100], [10, 20]

print(fractional_knapsack(n1, w1, value1, weight1))  # Output: 240.0
print(fractional_knapsack(n2, w2, value2, weight2))  # Output: 160.0
```

Q6)Maximum Units on a Truck.

Code:

```
def maximumUnits(boxTypes, truckSize):
    # Sort box types by units per box in descending order
    boxTypes.sort(key=lambda x: x[1], reverse=True)

    max_units = 0
    for boxes, units in boxTypes:
        if truckSize >= boxes:
            max_units += boxes * units
            truckSize -= boxes
        else:
            max_units += truckSize * units
            break

    return max_units

# Example usage
boxTypes1 = [[1, 3], [2, 2], [3, 1]]
boxTypes2 = [[5, 10], [2, 5], [4, 7], [3, 9]]
truckSize1 = 4
truckSize2 = 10

print(maximumUnits(boxTypes1, truckSize1))  # Output: 8
print(maximumUnits(boxTypes2, truckSize2))  # Output: 91
```

Q7)Chocolate Distribution.

Code:

```
def findMinDiff(arr, n, m):
    if m == 0 or n == 0:
        return 0

    arr.sort()  # Sort the array

    if n < m:
        return -1

    min_diff = float('inf')

    for i in range(len(arr) - m + 1):
        diff = arr[i + m - 1] - arr[i]
        min_diff = min(min_diff, diff)

    return min_diff

# Example usage
arr1 = [3, 4, 1, 9, 56, 7, 9, 12]
arr2 = [7, 3, 2, 4, 9, 12, 56]
m1, m2 = 5, 3

print(findMinDiff(arr1, len(arr1), m1))  # Output: 6
print(findMinDiff(arr2, len(arr2), m2))  # Output: 2
```

Q8)Shop in a Candy Store.

Code:

```
def candyStore(candies, N, K):
    candies.sort()  # Sort the candy prices in ascending order

    # Calculate minimum amount
    min_amount = sum(candies[:N // (K + 1)])

    # Calculate maximum amount
    max_amount = sum(candies[-(N // (K + 1)):])

    return min_amount, max_amount

# Example usage
candies1 = [3, 2, 1, 4]
```

```
candies2 = [3, 2, 1, 4, 5]
K1, K2 = 2, 4

min_cost1, max_cost1 = candyStore(candies1, len(candies1), K1)
min_cost2, max_cost2 = candyStore(candies2, len(candies2), K2)

print(min_cost1, max_cost1)  # Output: 3 7
print(min_cost2, max_cost2)  # Output: 1 5
```

Q9)Assign Cookies.

Code:

```
def findContentChildren(g, s):
    g.sort()  # Sort greed factors
    s.sort()  # Sort cookie sizes

    i, j = 0, 0
    content_children = 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, s1 = [1, 2, 3], [1, 1]
g2, s2 = [1, 2], [1, 2, 3]

print(findContentChildren(g1, s1))  # Output: 1
print(findContentChildren(g2, s2))  # Output: 2
```

Q10)N Meetings in one Room.

Code:

```
def maxMeetings(s, f, N):
    # Create a list of pairs (finish time, meeting index)
    a = [(f[i], i) for i in range(N)]
```

```
    a.sort()  # Sort by finish time

    time_limit = a[0][0]  # Initialize time limit with the finish time of the first meeting
    result = [a[0][1] + 1]  # Add the first meeting index to the result

    for i in range(1, N):
        if s[a[i][1]] > time_limit:
            result.append(a[i][1] + 1)
            time_limit = a[i][0]

    return result

# Example usage
s = [1, 3, 0, 5, 8, 5]
f = [2, 4, 6, 7, 9, 9]
N = len(s)
result = maxMeetings(s, f, N)
print(*result)  # Output: 1 2 4 5
```

Q11)Find Maximum Meetings in One room.

Code:

```
def maxMeetings(s, f, N):
    # Create a list of pairs (finish time, meeting index)
    a = [(f[i], i) for i in range(N)]
    a.sort()  # Sort by finish time

    time_limit = a[0][0]  # Initialize time limit with the finish time of the first meeting
    result = [a[0][1] + 1]  # Add the first meeting index to the result

    for i in range(1, N):
        if s[a[i][1]] > time_limit:
            result.append(a[i][1] + 1)
            time_limit = a[i][0]

    return result

# Example usage
s = [1, 3, 0, 5, 8, 5]
f = [2, 4, 6, 7, 9, 9]
N = len(s)
```

```
result = maxMeetings(s, f, N)
print(*result)  # Output: 1 2 4 5
```

Q12)Non Overlapping Intervals.

Code:

```
def minIntervalsToRemove(intervals):
    # Sort intervals by end time
    intervals.sort(key=lambda x: x[1])

    non_overlapping = []  # Stores non-overlapping intervals
    last_end = float('-inf')  # Initialize last end time

    for start, end in intervals:
        if start >= last_end:
            non_overlapping.append([start, end])
            last_end = end

    # Calculate the number of removed intervals
    return len(intervals) - len(non_overlapping)

# Example usage
intervals1 = [[1, 2], [2, 3], [3, 4], [1, 3]]
print(minIntervalsToRemove(intervals1))  # Output: 1

intervals2 = [[1, 2], [1, 2], [1, 2]]
print(minIntervalsToRemove(intervals2))  # Output: 2

intervals3 = [[1, 2], [2, 3]]
print(minIntervalsToRemove(intervals3))  # Output: 0
```

Q13)Insert Interval.

Code:

```
def insertInterval(intervals, newInterval):
    result = []
    i, n = 0, len(intervals)

    # Add intervals before newInterval
```

```
    while i < n and intervals[i][1] < newInterval[0]:
        result.append(intervals[i])
        i += 1

    # Merge overlapping intervals
    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)  # Add the merged newInterval

    # Add remaining intervals after newInterval
    while i < n:
        result.append(intervals[i])
        i += 1

    return result

# Example usage
intervals1 = [[1, 3], [6, 9]]
newInterval1 = [2, 5]
print(insertInterval(intervals1, newInterval1))  # Output: [[1, 5], [6, 9]]

intervals2 = [[1, 2], [3, 5], [6, 7], [8, 10], [12, 16]]
newInterval2 = [4, 8]
print(insertInterval(intervals2, newInterval2))  # Output: [[1, 2], [3, 10], [12, 16]]
```

Q14)Merge Intervals.

Code:

```
def mergeIntervals(intervals):
    intervals.sort(key=lambda x: x[0])  # Sort by start time
    result = []

    for interval in intervals:
        if not result or interval[0] > result[-1][1]:
            result.append(interval)
        else:
            result[-1][1] = max(result[-1][1], interval[1])

    return result
```

```
# Example usage
intervals1 = [[1, 3], [2, 6], [8, 10], [15, 18]]
print(mergeIntervals(intervals1))  # Output: [[1, 6], [8, 10], [15, 18]]

intervals2 = [[1, 4], [4, 5]]
print(mergeIntervals(intervals2))  # Output: [[1, 5]]
```

Q15)Job Sequencing Problem.

Code:

```
def JobScheduling(Jobs, N):
    # Sort jobs by profit in descending order
    Jobs.sort(key=lambda x: x[2], reverse=True)

    # Initialize slots and profit
    slots = [False] * N
    profit = 0

    for job in Jobs:
        deadline = job[1]
        for i in range(min(N, deadline) - 1, -1, -1):
            if not slots[i]:
                slots[i] = True
                profit += job[2]
                break

    return [sum(slots), profit]

# Example usage
Jobs1 = [(1, 4, 20), (2, 1, 10), (3, 1, 40), (4, 1, 30)]
N1 = 4
result1 = JobScheduling(Jobs1, N1)
print(*result1)  # Output: 2 60

Jobs2 = [(1, 2, 100), (2, 1, 19), (3, 2, 27), (4, 1, 25), (5, 1, 15)]
N2 = 5
result2 = JobScheduling(Jobs2, N2)
print(*result2)  # Output: 2 127
```

Q16)Minimum Cost to Cut Ropes.

Code:

```
import heapq

def minCost(arr, n):
    # Initialize the min heap
    heapq.heapify(arr)

    total_cost = 0
    while len(arr) > 1:
        # Extract the two smallest elements
        rope1 = heapq.heappop(arr)
        rope2 = heapq.heappop(arr)

        # Combine the ropes and add back to the heap
        combined_length = rope1 + rope2
        heapq.heappush(arr, combined_length)

        # Update the total cost
        total_cost += combined_length

    return total_cost

# Example usage
arr1 = [4, 3, 2, 6]
n1 = 4
print(minCost(arr1, n1))  # Output: 29

arr2 = [4, 2, 7, 6, 9]
n2 = 5
print(minCost(arr2, n2))  # Output: 62
```

Q17)Jump Game.

Code: