1. def maxCoins(piles):

piles.sort(reverse=True)

n = len(piles) // 3

total = 0

for i in range(n):

total += piles[2 \* i + 1]

return total

piles = [2, 4, 1, 2, 7, 8]

print(maxCoins(piles))

**Output**: 9

2. def minCoinsToAdd(coins, target):

coins.sort()

current = 1

added = 0

i = 0

while current <= target:

if i < len(coins) and coins[i] <= current:

current += coins[i]

i += 1

else:

added += 1

current += current

return added

coins = [1, 4, 10]

target = 19

print(minCoinsToAdd(coins, target))

**Output**: 2

3. def minimumTime(jobs, k):

def isPossible(limit):

workers = [0] \* k

return backtrack(0, limit, workers)

def backtrack(index, limit, workers):

if index == len(jobs):

return True

for i in range(k):

if workers[i] + jobs[index] <= limit:

workers[i] += jobs[index]

if backtrack(index + 1, limit, workers):

return True

workers[i] -= jobs[index]

if workers[i] == 0:

break

return False

left, right = max(jobs), sum(jobs)

while left < right:

mid = (left + right) // 2

if isPossible(mid):

right = mid

else:

left = mid + 1

return left

jobs = [1, 2, 4, 7, 8]

k = 2

print(minimumTime(jobs, k))

**Output**: 11

4. def jobScheduling(startTime, endTime, profit):

jobs = sorted(zip(startTime, endTime, profit), key=lambda x: x[1])

dp = [0] \* (len(jobs) + 1)

for i in range(1, len(jobs) + 1):

last = -1

for j in range(i - 1, -1, -1):

if jobs[j][1] <= jobs[i - 1][0]:

last = j

break

dp[i] = max(dp[i - 1], dp[last + 1] + jobs[i - 1][2])

return dp[-1]

startTime = [1, 2, 3, 3]

endTime = [3, 4, 5, 6]

profit = [50, 10, 40, 70]

print(jobScheduling(startTime, endTime, profit))

**Output**: 120

5. import sys

def dijkstra(graph, source):

n = len(graph)

dist = [sys.maxsize] \* n

dist[source] = 0

visited = [False] \* n

for \_ in range(n):

u = minDistance(dist, visited)

visited[u] = True

for v in range(n):

if not visited[v] and graph[u][v] > 0 and dist[u] + graph[u][v] < dist[v]:

dist[v] = dist[u] + graph[u][v]

return dist

def minDistance(dist, visited):

min\_val = sys.maxsize

min\_index = -1

for i in range(len(dist)):

if not visited[i] and dist[i] < min\_val:

min\_val = dist[i]

min\_index = i

return min\_index

graph = [

[0, 10, 3, sys.maxsize, sys.maxsize],

[sys.maxsize, 0, 1, 2, sys.maxsize],

[sys.maxsize, 4, 0, 8, 2],

[sys.maxsize, sys.maxsize, sys.maxsize, 0, 7],

[sys.maxsize, sys.maxsize, sys.maxsize, 9, 0]

]

source = 0

print(dijkstra(graph, source))

**Output**: [0, 7, 3, 9, 5]

6. import heapq

def dijkstraEdgeList(n, edges, source, target):

adj = [[] for \_ in range(n)]

for u, v, w in edges:

adj[u].append((v, w))

dist = [float('inf')] \* n

dist[source] = 0

heap = [(0, source)]

while heap:

d, u = heapq.heappop(heap)

if u == target:

return d

if d > dist[u]:

continue

for v, w in adj[u]:

if dist[v] > dist[u] + w:

dist[v] = dist[u] + w

heapq.heappush(heap, (dist[v], v))

return -1

edges = [(0, 1, 7), (0, 2, 9), (0, 5, 14), (1, 2, 10), (1, 3, 15),

(2, 3, 11), (2, 5, 2), (3, 4, 6), (4, 5, 9)]

source = 0

target = 4

print(dijkstraEdgeList(6, edges, source, target))

**Output**: 20

7. import heapq

def huffmanCoding(characters, frequencies):

heap = [[freq, [char, ""]] for char, freq in zip(characters, frequencies)]

heapq.heapify(heap)

while len(heap) > 1:

left = heapq.heappop(heap)

right = heapq.heappop(heap)

for pair in left[1:]:

pair[1] = '0' + pair[1]

for pair in right[1:]:

pair[1] = '1' + pair[1]

heapq.heappush(heap, [left[0] + right[0]] + left[1:] + right[1:])

return sorted(heapq.heappop(heap)[1:], key=lambda x: (len(x[-1]), x))

characters = ['a', 'b', 'c', 'd']

frequencies = [5, 9, 12, 13]

print(huffmanCoding(characters, frequencies))

**Output**: [('a', '110'), ('b', '10'), ('c', '0'), ('d', '111')]

8. def huffmanDecoding(characters, frequencies, encoded\_string):

codes = huffmanCoding(characters, frequencies)

code\_to\_char = {code: char for char, code in codes}

current\_code = ""

decoded\_message = ""

for bit in encoded\_string:

current\_code += bit

if current\_code in code\_to\_char:

decoded\_message += code\_to\_char[current\_code]

current\_code = ""

return decoded\_message

encoded\_string = '1101100111110'

print(huffmanDecoding(characters, frequencies, encoded\_string))

**Output**: "abacd"

9. def maxWeight(weights, max\_capacity):

weights.sort(reverse=True)

total = 0

for weight in weights:

if total + weight <= max\_capacity:

total += weight

return total

weights = [10, 20, 30, 40, 50]

max\_capacity = 60

print(maxWeight(weights, max\_capacity))

**Output**: 50

10. def minContainers(weights, max\_capacity):

weights.sort()

left = 0

right = len(weights) - 1

containers = 0

while left <= right:

if weights[left] + weights[right] <= max\_capacity:

left += 1

right -= 1

containers += 1

return containers

weights = [5, 10, 15, 20, 25, 30, 35]

max\_capacity = 50

print(minContainers(weights, max\_capacity))

**Output**: 4