// 1

def gale\_shapley(men\_preferences, women\_preferences):  
 n = len(men\_preferences)  
 engaged = {}  
 men\_free = set(range(n))  
  
 while men\_free:  
 m = men\_free.pop()  
 w = men\_preferences[m].pop(0)  
  
 if w not in engaged:  
 engaged[w] = m  
 else:  
 m1 = engaged[w]  
 w\_prefs = women\_preferences[w]  
 if w\_prefs.index(m) < w\_prefs.index(m1):  
 engaged[w] = m  
 men\_free.add(m1)  
 else:  
 men\_free.add(m)  
  
 return engaged  
  
# Example preferences for men and women  
men\_preferences = {  
 0: [1, 0, 2, 3],  
 1: [0, 1, 2, 3],  
 2: [0, 1, 2, 3],  
 3: [1, 0, 2, 3]  
}  
  
women\_preferences = {  
 0: [1, 0, 2, 3],  
 1: [0, 1, 2, 3],  
 2: [1, 0, 2, 3],  
 3: [1, 0, 2, 3]  
}  
  
# Perform Gale-Shapley algorithm  
stable\_matching = gale\_shapley(men\_preferences, women\_preferences)  
  
# Print the stable matching result  
for woman, man in stable\_matching.items():  
 print(f"Man {man} is engaged to Woman {woman}.")

Man 0 is engaged to Woman 1.

Man 1 is engaged to Woman 0.

Man 2 is engaged to Woman 2.

Man 3 is engaged to Woman 3.

2.

def merge\_sort(arr):  
 if len(arr) <= 1:  
 return arr  
  
 mid = len(arr) // 2  
 left = arr[:mid]  
 right = arr[mid:]  
  
 left = merge\_sort(left)  
 right = merge\_sort(right)  
  
 return merge(left, right)  
  
  
def merge(left, right):  
 result = []  
 i = j = 0  
  
 while i < len(left) and j < len(right):  
 if left[i] < right[j]:  
 result.append(left[i])  
 i += 1  
 else:  
 result.append(right[j])  
 j += 1  
  
 result.extend(left[i:])  
 result.extend(right[j:])  
 return result  
  
  
# Example usage  
arr = [8, 4, 2, 1, 6, 9, 3, 5, 7]  
sorted\_arr = merge\_sort(arr)  
print("Sorted array:", sorted\_arr)

original array [8, 4, 2, 1, 6, 9, 3, 5, 7]

Sorted array: [1, 2, 3, 4, 5, 6, 7, 8, 9]

3.

import heapq  
def dijkstra(graph, start):  
 distances = {node: float('inf') for node in graph}  
 distances[start] = 0  
  
 priority\_queue = [(0, start)]  
  
 while priority\_queue:  
 current\_distance, current\_node = heapq.heappop(priority\_queue)  
  
 if current\_distance > distances[current\_node]:  
 continue  
  
 for neighbor, weight in graph[current\_node].items():  
 distance = current\_distance + weight  
 if distance < distances[neighbor]:  
 distances[neighbor] = distance  
 heapq.heappush(priority\_queue, (distance, neighbor))  
  
 return distances  
  
  
# Example graph as an adjacency list  
graph = {  
 'A': {'B': 1, 'C': 4},  
 'B': {'A': 1, 'C': 3, 'D': 2},  
 'C': {'A': 4, 'B': 3, 'D': 5},  
 'D': {'B': 2, 'C': 5}  
}  
  
start\_node = 'A'  
shortest\_distances = dijkstra(graph, start\_node)  
print("Shortest distances from node", start\_node + ":", shortest\_distances)

5.

def drama\_venue\_allocation(requests):  
 requests.sort(key=lambda x: x[1]) # Sort requests based on finish times  
 prev\_profit = curr\_profit = 0  
  
 for start\_time, finish\_time, profit in requests:  
 max\_profit = max(prev\_profit + profit, curr\_profit)  
 prev\_profit, curr\_profit = curr\_profit, max\_profit  
  
 return curr\_profit  
  
# Example requests: (start\_time, finish\_time, profit)  
requests = [(1, 2, 100), (2, 5, 200), (3, 6,300 ), (4, 8, 400),(4,9,500),(6,10,100)]  
  
# Calculate the maximum profit using dynamic programming  
max\_profit = drama\_venue\_allocation(requests)  
  
print("Maximum Profit:", max\_profit)

Maximum Profit: 900

6.

def knapsack(values, weights, capacity):  
 n = len(values)  
 dp = [0] \* (capacity + 1)  
  
 for i in range(n):  
 for w in range(capacity, weights[i] - 1, -1):  
 dp[w] = max(dp[w], values[i] + dp[w - weights[i]])  
  
 return dp[capacity]  
  
  
# Example values and weights for items  
values = [10,4,9,11]  
weights = [3,5,6,2]  
capacity = 7  
  
max\_value = knapsack(values, weights, capacity)  
print("Maximum value:", max\_value)

Maximum value: 21

7.

def bellman\_ford(vertices, edges, start):  
 distances = {v: float('inf') for v in vertices}  
 distances[start] = 0  
  
 for \_ in vertices:  
 for u, v, weight in edges:  
 if distances[u] + weight < distances[v]:  
 distances[v] = distances[u] + weight  
  
 return distances  
  
  
# Example usage  
vertices = ['A', 'B', 'C', 'D', 'E','F']  
edges = [('A', 'B', -4), ('B', 'E', -2), ('B', 'D', -1), ('D', 'A', 6),  
 ('A', 'F', -3), ('D', 'F', 4), ('E', 'F', 2),('C','B',8),('C','F',3)]  
  
start\_vertex = 'A'  
shortest\_distances = bellman\_ford(vertices, edges, start\_vertex)  
print("Shortest distances from vertex", start\_vertex + ":", shortest\_distances)

Shortest distances from vertex A: {'A': 0, 'B': -4, 'C': inf, 'D': -5, 'E': -6, 'F': -4}

8.