1: A

2: A

3: A

4: B

5: A

6: A

7: B

8: A

9: B

10: D

11: B

12 : B

13: A

14: B

15: D

16: B

17: A

18: B

19: C

20: B

21 : A

22 : B

23 : A

24: A

25: C

26; A

27 : D

--------------------------------------------------------------------------------------------------------------------------------------

PROGRAMS –

Ans 1)

def evaluate\_performance(employees):

evaluated\_employees = []

for employee in employees:

name = employee["name"]

scores = employee["scores"]

weights = employee["weights"]

# Calculate the performance score for the employee

performance\_score = sum(scores[criterion] \* weights[criterion] for criterion in scores)

# Create a dictionary for the evaluated employee

evaluated\_employee = {

"name": name,

"performance\_score": performance\_score

}

evaluated\_employees.append(evaluated\_employee)

return evaluated\_employees

# Example usage:

employees = [

{

"name": "John",

"scores": {"Quality of Work": 90, "Team Collaboration": 80},

"weights": {"Quality of Work": 0.6, "Team Collaboration": 0.4}

},

{

"name": "Alice",

"scores": {"Quality of Work": 95, "Team Collaboration": 85},

"weights": {"Quality of Work": 0.6, "Team Collaboration": 0.4}

}

]

evaluated\_employees = evaluate\_performance(employees)

print(evaluated\_employees)

Ans 3)

def sort\_employees\_by\_id(employees):

# Sort employees by id in ascending order

sorted\_employees = sorted(employees, key=lambda x: x["id"])

return sorted\_employees

def sort\_employees\_by\_salary(employees):

# Sort employees by salary in descending order

sorted\_employees = sorted(employees, key=lambda x: x["salary"], reverse=True)

return sorted\_employees

# Example usage:

employees = [

{"id": 101, "name": "John", "salary": 60000},

{"id": 103, "name": "Alice", "salary": 75000},

{"id": 102, "name": "Bob", "salary": 65000},

# Sort employees by id

sorted\_by\_id = sort\_employees\_by\_id(employees)

print("Sorted by id:")

print(sorted\_by\_id)

# Sort employees by salary

sorted\_by\_salary = sort\_employees\_by\_salary(employees)

print("\nSorted by salary:")

print(sorted\_by\_salary)

Ans 4)

import heapq

def find\_kth\_number(N, prime\_numbers, K):

# Initialize a min-heap with the prime numbers

heap = prime\_numbers.copy()

heapq.heapify(heap)

# Generate the Kth number by multiplying the smallest number with previously generated numbers

for \_ in range(K - 1):

smallest = heapq.heappop(heap)

for prime in prime\_numbers:

heapq.heappush(heap, smallest \* prime)

return heapq.heappop(heap)

Ans 6)

def max\_subarray\_sum(arr, n):

max\_sum = arr[0] # Initialize max\_sum to the first element of the array

current\_sum = arr[0] # Initialize current\_sum to the first element

for i in range(1, n):

# Choose the maximum between the current element and the current sum plus the current element

current\_sum = max(arr[i], current\_sum + arr[i])

# Update max\_sum if the current\_sum is greater

max\_sum = max(max\_sum, current\_sum)

return max\_sum

Ans7)

def longest\_common\_subsequence(s1, s2):

m, n = len(s1), len(s2)

# Create a 2D table to store the length of LCS for subproblems

dp = [[0] \* (n + 1) for \_ in range(m + 1)]

# Fill in the dp table

for i in range(1, m + 1):

for j in range(1, n + 1):

if s1[i - 1] == s2[j - 1]:

dp[i][j] = dp[i - 1][j - 1] + 1

else:

dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

return dp[m][n]