**PRACTICAL 6**

**Aim :**

**Construction of OBST.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**TASK 1 :**

**CODE :**

def optimal\_bst(p, q, n):

# Initializing matrices

E = [[0 for \_ in range(n + 1)] for \_ in range(n + 1)] # Expected cost

W = [[0 for \_ in range(n + 1)] for \_ in range(n + 1)] # Weights

R = [[0 for \_ in range(n + 1)] for \_ in range(n + 1)] # Roots ( No root for dummy-only interval already intialised 0 )

# Initialization for empty subtrees

for i in range(n + 1):

E[i][i] = q[i] # Cost of empty subtree

W[i][i] = q[i] # Weight of interval with no key

# Compute for all intervals of increasing length

for d in range(1, n + 1): # d = number of keys in interval

for i in range(n - d + 1): # i = start index

j = i + d # j = end index

E[i][j] = float('inf') # E[i][j] <- infinity

W[i][j] = W[i][j - 1] + p[j - 1] + q[j]

# Try each key k as root

for k in range(i + 1, j + 1):

cost = E[i][k - 1] + E[k][j] + W[i][j]

if cost < E[i][j]:

E[i][j] = cost

R[i][j] = k

return E, W, R

n = int(input("Total Number : ").strip())

keys = list(map(int, input("Keys : ").strip().split()))

p = list(map(float, input("Probabilities of Successful Searches").strip().split()))

q = list(map(float, input("Probabilities of Unsuccessful Searches").strip().split()))

E, W, R = optimal\_bst(p, q, n)

# Printing Minimum Expected Cost ( rounded to 4 decimal places )

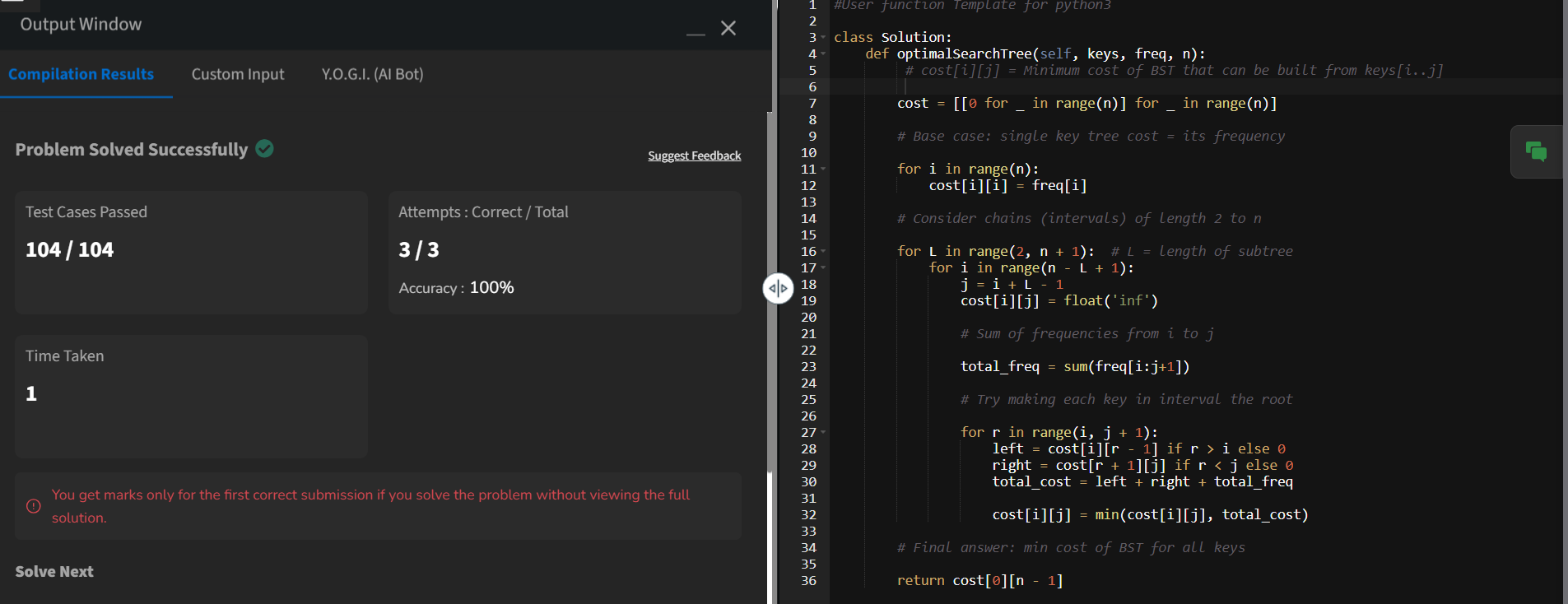
print(f"\n\nMinimum Expected Cost : {E[0][n]:.4f}")

**OUTPUT :**

****

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**TASK 2 :**

**OUTPUT :  
  
**