PRACTICAL NO : 06

Name : Soham Adgokar

Roll No : A2-B2-28

Date : 14/10/25

Aim: Construction of OBST

Problem Statement: Smart Library Search Optimization

Task 1:

Scenario:

A university digital library system stores frequently accessed books using a binary search mechanism. The library admin wants to minimize the average search time for book lookups by arranging the book IDs optimally in a binary search tree. Each book ID has a probability of being searched successfully and an associated probability for unsuccessful searches (when a book ID does not exist between two keys).

Your task is to determine the minimum expected cost of searching using an Optimal Binary Search Tree (OBST).

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#define MAX 100
double e[MAX][MAX];
double w[MAX][MAX];
int root[MAX][MAX];
double min(double a, double b) {
 return a < b ? a : b;
}
int main() {
 int n;
 printf("Enter the number of book IDs (n): ");
 scanf("%d", &n);
 int keys[n + 1];
 double p[n + 1];
```

```
 double q[n + 2];
 printf("Enter %d space-separated book IDs:\n", n);
 for (int i = 1; i <= n; i++) {
 scanf("%d", &keys[i]);
 }
 printf("Enter %d space-separated probabilities of
successfu searches (p[1] to p[%d]):\n", n, n);
 for (int i = 1; i <= n; i++) {
 scanf("%lf", &p[i]);
 }
 printf("Enter %d space-separated probabilities of
unsuccessful searches (q[0] to q[%d]):\n", n + 1, n);
 for (int i = 0; i <= n; i++) {
 scanf("%lf", &q[i]);
 }
 printf("\n Book IDs and Probabilities:\n");
 for (int i = 1; i <= n; i++) {
 printf("Key: %d, Success Probability (p[%d]): %.2lf\n",
keys[i], i, p[i]);
 }
 for (int i = 0; i <= n; i++) {
 printf("Unsuccessful Probability (q[%d]): %.2lf\n", i, q[i]);
```

```
}

for (int i = 0; i <= n; i++) {
e[i][i] = q[i];
w[i][i] = q[i];
}

for (int l = 1; l <= n; l++) {
for (int i = 0; i <= n - l; i++) {
int j = i + l;
e[i][j] = DBL_MAX;
w[i][j] = w[i][j - 1] + p[j] + q[j];
for (int r = i + 1; r <= j; r++) {
double cost = e[i][r - 1] + e[r][j] + w[i][j];
if (cost < e[i][j]) {
e[i][j] = cost;
root[i][j] = r;
  }
 }
}
}
```

printf("\n Minimum expected cost of the Optimal Binary Search Tree:%.4lf\n", e[0][n]);

return 0;

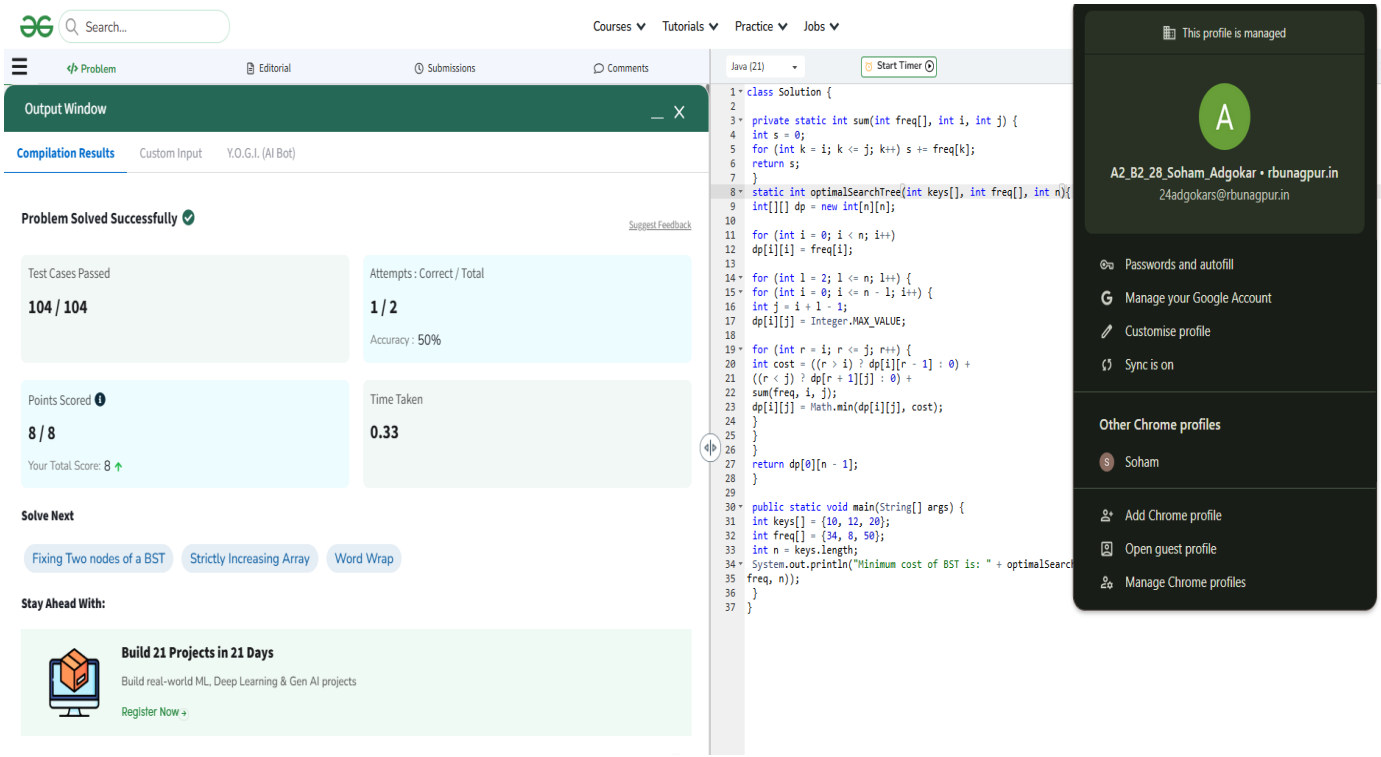
}

Output :

```
Enter the number of book IDs (n): 4
Enter 4 space-separated book IDs:
10
20
30
40
Enter 4 space-separated probabilities of successfu searches (p[1] to p[4]):
0.1
0.2
0.4
0.3
Enter 5 space-separated probabilities of unsuccessful searches (q[0] to q[4]):
0.05
0.1
0.05
0.05
0.1

 Book IDs and Probabilities:
Key: 10, Success Probability (p[1]): 0.10
Key: 20, Success Probability (p[2]): 0.20
Key: 30, Success Probability (p[3]): 0.40
Key: 40, Success Probability (p[4]): 0.30
Unsuccessful Probability (q[0]): 0.05
Unsuccessful Probability (q[1]): 0.10
Unsuccessful Probability (q[2]): 0.05
Unsuccessful Probability (q[3]): 0.05
Unsuccessful Probability (q[4]): 0.10

 Minimum expected cost of the Optimal Binary Search Tree:2.9000


=== Code Execution Successful ===
```

Task 2:

https://www.geeksforgeeks.org/problems/optimal-binary-search-tree2214/1



Code :

```
class Solution {
 private static int sum(int freq[], int i, int j) {
 int s = 0;
 for (int k = i; k <= j; k++) s += freq[k];
 return s;
 }
 static int optimalSearchTree(int keys[], int freq[], int n){
 int[][] dp = new int[n][n];
```

```
for (int i = 0; i < n; i++)
dp[i][i] = freq[i];
for (int l = 2; l <= n; l++) {
for (int i = 0; i <= n - l; i++) {
int j = i + l - 1;
dp[i][j] = Integer.MAX_VALUE;
for (int r = i; r <= j; r++) {
int cost = ((r > i) ? dp[i][r - 1] : 0) +
((r < j) ? dp[r + 1][j] : 0) +
sum(freq, i, j);
dp[i][j] = Math.min(dp[i][j], cost);
}
}
}
return dp[0][n - 1];
}

public static void main(String[] args) {
int keys[] = {10, 12, 20};
int freq[] = {34, 8, 50};
```

```
 int n = keys.length;
 System.out.println("Minimum cost of BST is: " +
optimalSearchTree(keys,
freq, n));
 }
}
```