PRACTICAL 6

Name: Isha Raut
Section: A4
Batch: B2
Roll no: 23

Aim: Construction of OBST

Problem Statement: Smart Library Search Optimization.

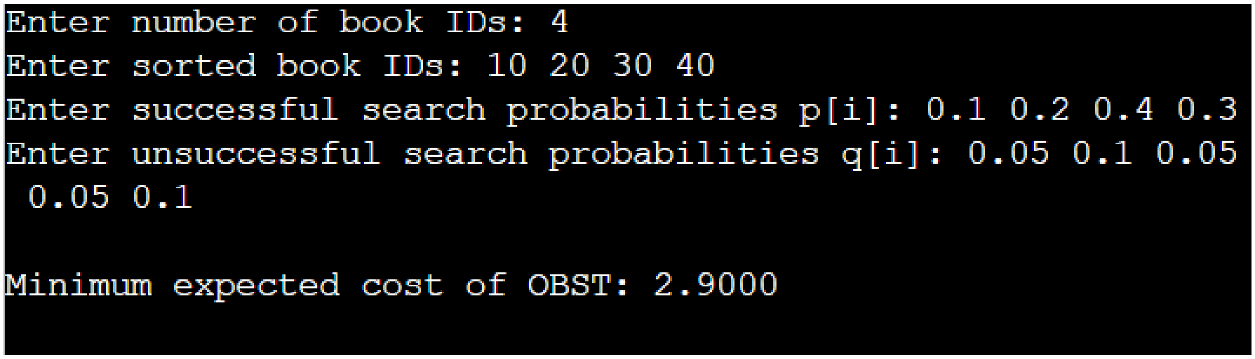
Task 1:

Code:

```
#include <stdio.h>
#define MAX 20

int main()
{
   int n;
   double p[MAX], q[MAX], e[MAX][MAX], w[MAX][MAX];

   printf("Enter number of book IDs: ");
   scanf("%d", &n);

   int keys[MAX];
   printf("Enter sorted book IDs: ");
   for (int i = 1; i <= n; i++)
   {
      scanf("%d", &keys[i]);
   }

   printf("Enter successful search probabilities p[i]: ");
```

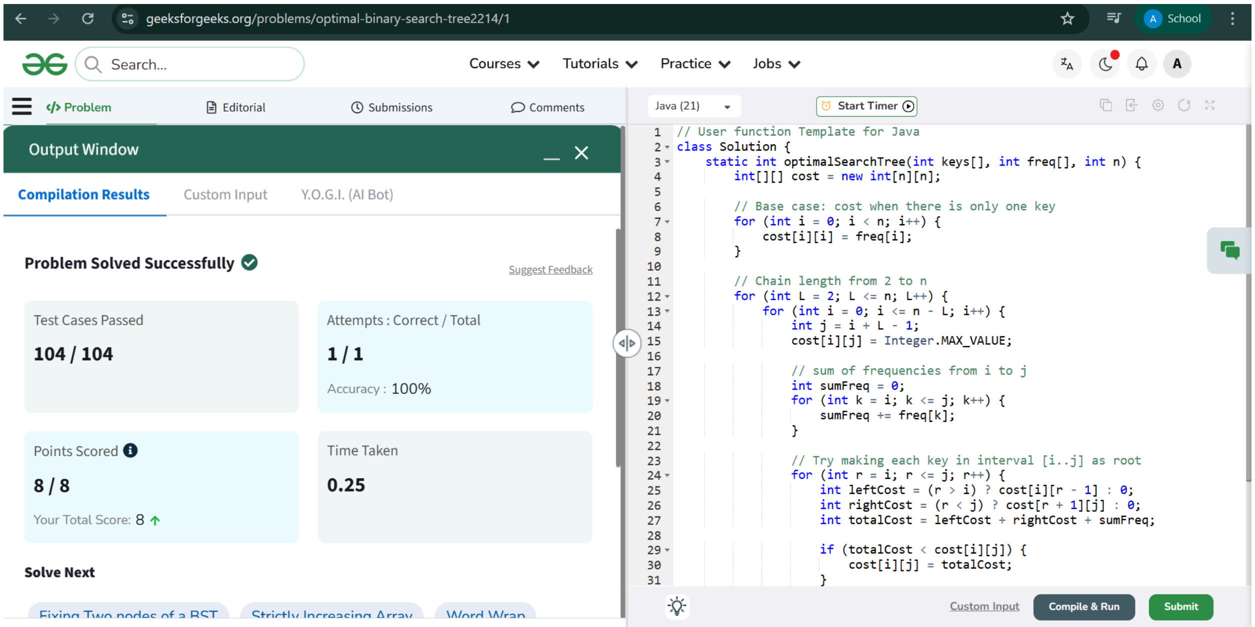
```
for (int i = 1; i <= n; i++)
{
    scanf("%lf", &p[i]);
}

printf("Enter unsuccessful search probabilities q[i]: ");
for (int i = 0; i <= n; i++)
{
    scanf("%lf", &q[i]);
}

for (int i = 1; i <= n + 1; i++)
{
    e[i][i - 1] = q[i - 1];
    w[i][i - 1] = q[i - 1];
}

for (int length = 1; length <= n; length++)
{
    for (int i = 1; i <= n - length + 1; i++)
    {
        int j = i + length - 1;
        e[i][j] = 9999999;
        w[i][j] = w[i][j - 1] + p[j] + q[j];


        for (int r = i; r <= j; r++)
        {
            double cost = e[i][r - 1] + e[r + 1][j] + w[i][j];
            if (cost < e[i][j])
            {
                e[i][j] = cost;
            }
        }
    }
    }


    printf("\nMinimum expected cost of OBST: %.4lf\n", e[1][n]);

    return 0;
}
```

Output:



```
Enter number of book IDs: 4
Enter sorted book IDs: 10 20 30 40
Enter successful search probabilities p[i]: 0.1 0.2 0.4 0.3
Enter unsuccessful search probabilities q[i]: 0.05 0.1 0.05
 0.05 0.1

Minimum expected cost of OBST: 2.9000
```

Task 2:



Problem Solved Successfully

Test Cases Passed 104 / 104	Attempts : Correct / Total 1 / 1 Accuracy : 100%
Points Scored 8 / 8 Your Total Score: 8	Time Taken 0.25

```java
// User function Template for Java
class Solution {
    static int optimalSearchTree(int keys[], int freq[], int n) {
        int[][] cost = new int[n][n];

        // Base case: cost when there is only one key
        for (int i = 0; i < n; i++) {
            cost[i][i] = freq[i];
        }

        // Chain length from 2 to n
        for (int L = 2; L <= n; L++) {
            for (int i = 0; i <= n - L; i++) {
                int j = i + L - 1;
                cost[i][j] = Integer.MAX_VALUE;

                // sum of frequencies from i to j
                int sumFreq = 0;
                for (int k = i; k <= j; k++) {
                    sumFreq += freq[k];
                }

                // Try making each key in interval [i..j] as root
                for (int r = i; r <= j; r++) {
                    int leftCost = (r > i) ? cost[i][r - 1] : 0;
                    int rightCost = (r < j) ? cost[r + 1][j] : 0;
                    int totalCost = leftCost + rightCost + sumFreq;

                    if (totalCost < cost[i][j]) {
                        cost[i][j] = totalCost;
                    }
```

Link:

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