DAA Practical 6

Name : Ansha Lanjewar

Section: A3_B1_13

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).

Input Format

First line: integer n — number of book IDs.

Second line: n integers representing the sorted book IDs (keys).

Third line: n real numbers — probabilities of successful searches (p[i]).

Fourth line: n+1 real numbers — probabilities of unsuccessful searches (q[i]).

Keys: 10 20 30 40

P[i]: 0.1 0.2 0.4 0.3

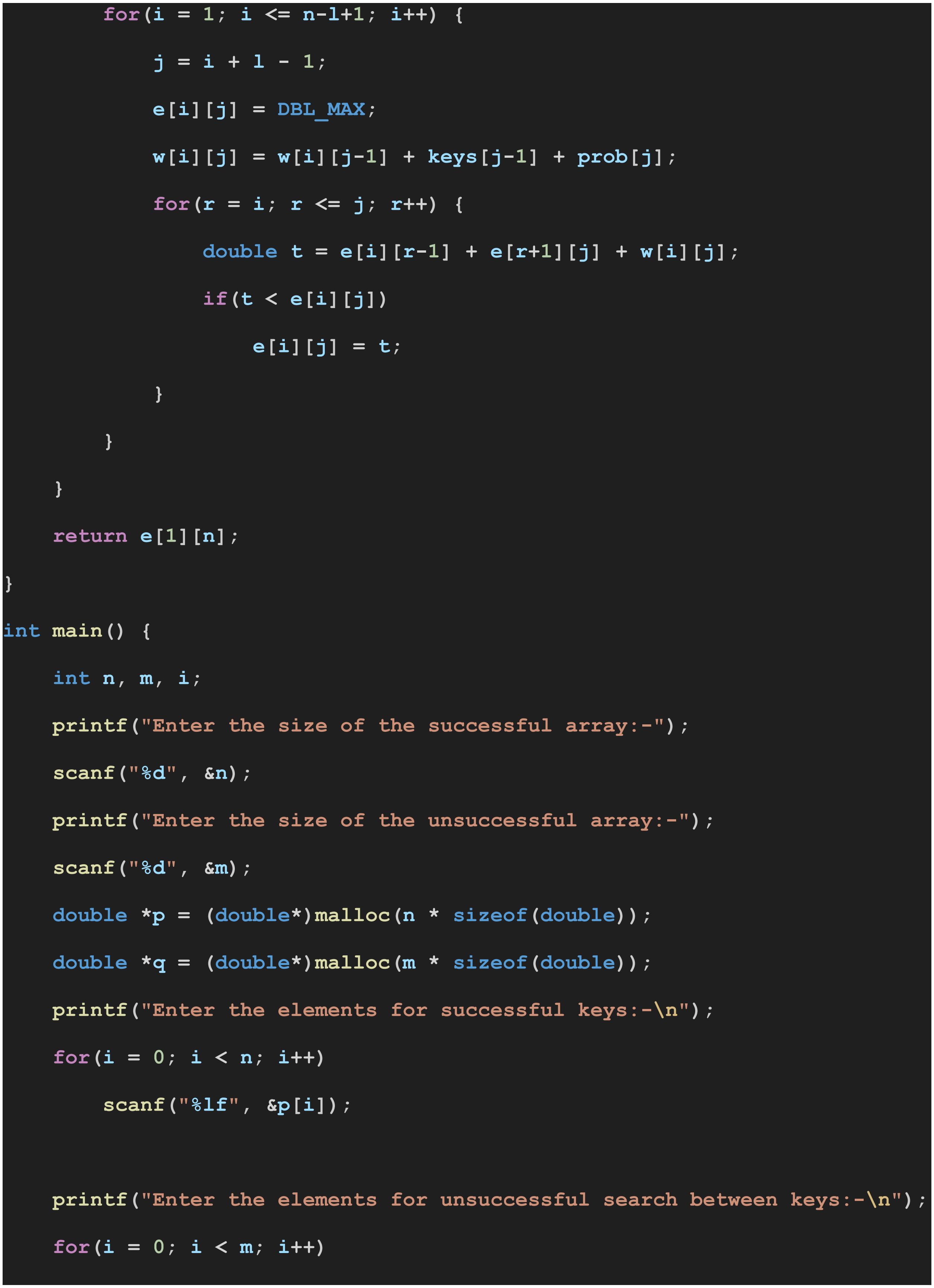
Q[i]: 0.05 0.1 0.05 0.05 0.1

Output Format:-Print the minimum expected cost of the Optimal Binary Search Tree, rounded to 4 decimal

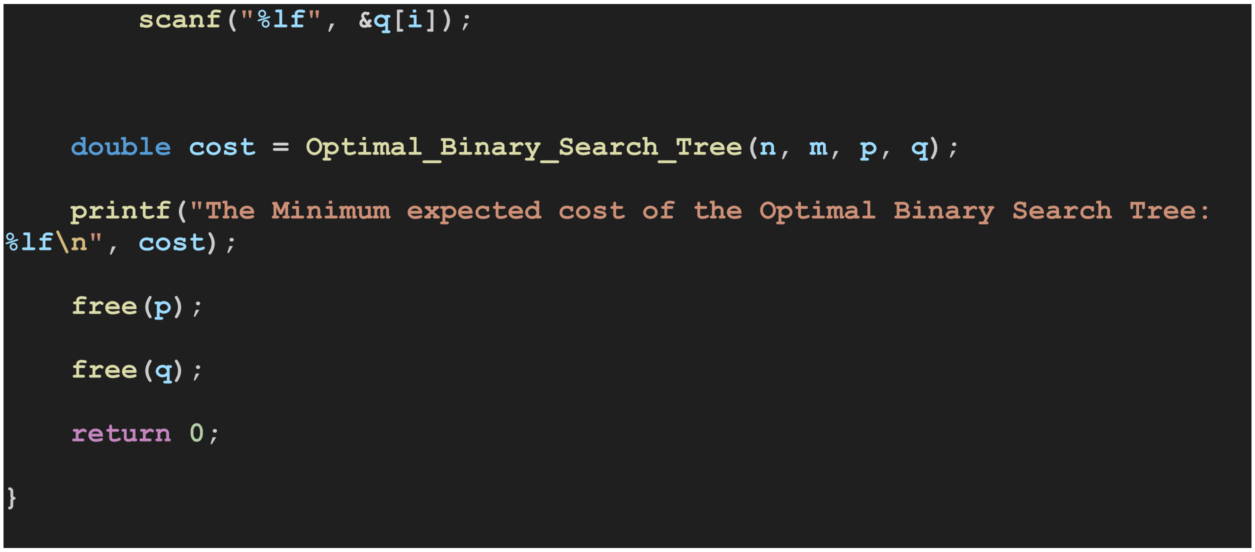
Places.

Code:-

```
#include <stdio.h>

#include <stdlib.h>

#include <float.h>

double Optimal_Binary_Search_Tree(int n, int m, double keys[], double
prob[]) {

    double e[n+2][n+2], w[n+2][n+2];

    int i, j, l, r;

    for(i = 0; i <= n+1; i++) {

        for(j = 0; j <= n+1; j++) {

            e[i][j] = 0;

            w[i][j] = 0;

        }

    }

    for(i = 1; i <= n+1; i++) {

        e[i][i-1] = prob[i-1];

        w[i][i-1] = prob[i-1];

    }

    for(l = 1; l <= n; l++) {
```

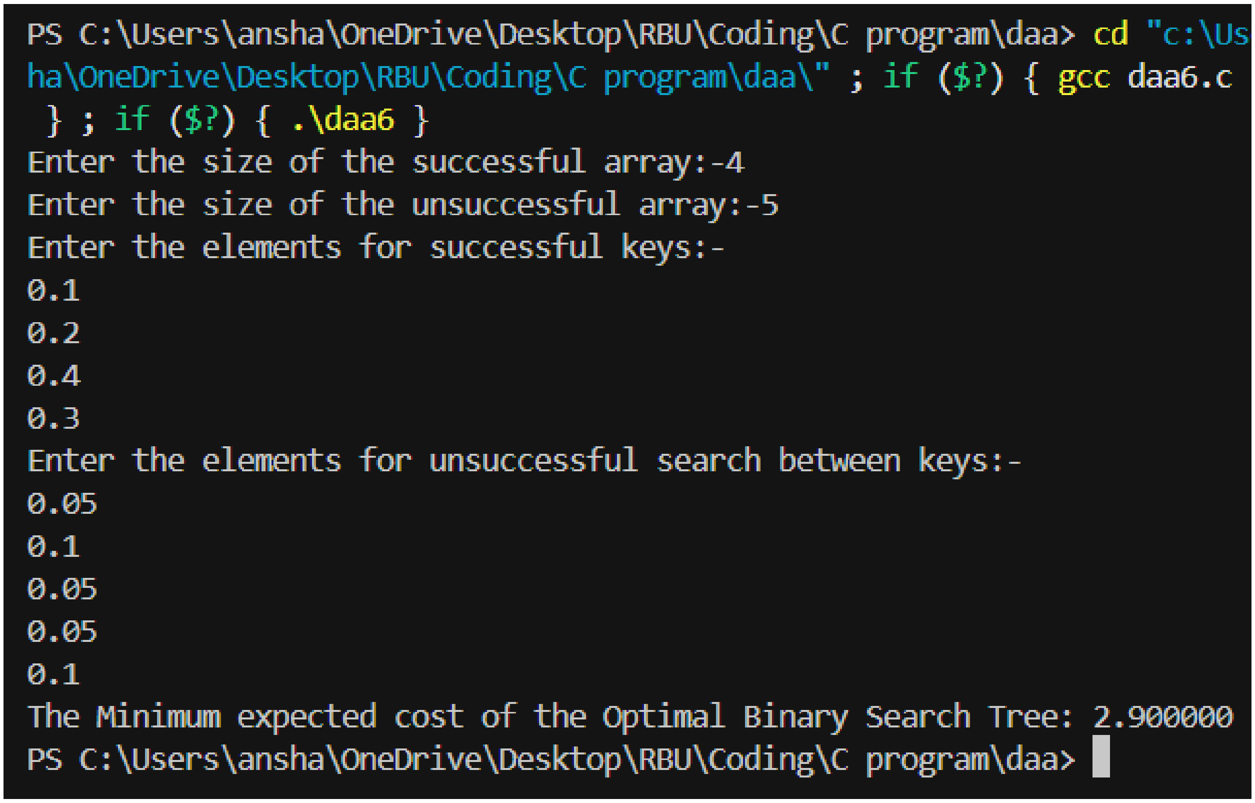


```
        for(i = 1; i <= n-l+1; i++) {
            j = i + l - 1;
            e[i][j] = DBL_MAX;
            w[i][j] = w[i][j-1] + keys[j-1] + prob[j];
            for(r = i; r <= j; r++) {
                double t = e[i][r-1] + e[r+1][j] + w[i][j];
                if(t < e[i][j])
                    e[i][j] = t;
            }
        }
    }
    return e[1][n];
}
int main() {
    int n, m, i;
    printf("Enter the size of the successful array:-");
    scanf("%d", &n);
    printf("Enter the size of the unsuccessful array:-");
    scanf("%d", &m);
    double *p = (double*)malloc(n * sizeof(double));
    double *q = (double*)malloc(m * sizeof(double));
    printf("Enter the elements for successful keys:-\n");
    for(i = 0; i < n; i++)
        scanf("%lf", &p[i]);

    printf("Enter the elements for unsuccessful search between keys:-\n");
    for(i = 0; i < m; i++)
```



```
        scanf("%lf", &q[i]);


    double cost = Optimal_Binary_Search_Tree(n, m, p, q);

    printf("The Minimum expected cost of the Optimal Binary Search Tree:
%lf\n", cost);

    free(p);

    free(q);

    return 0;

}
```

Output:-

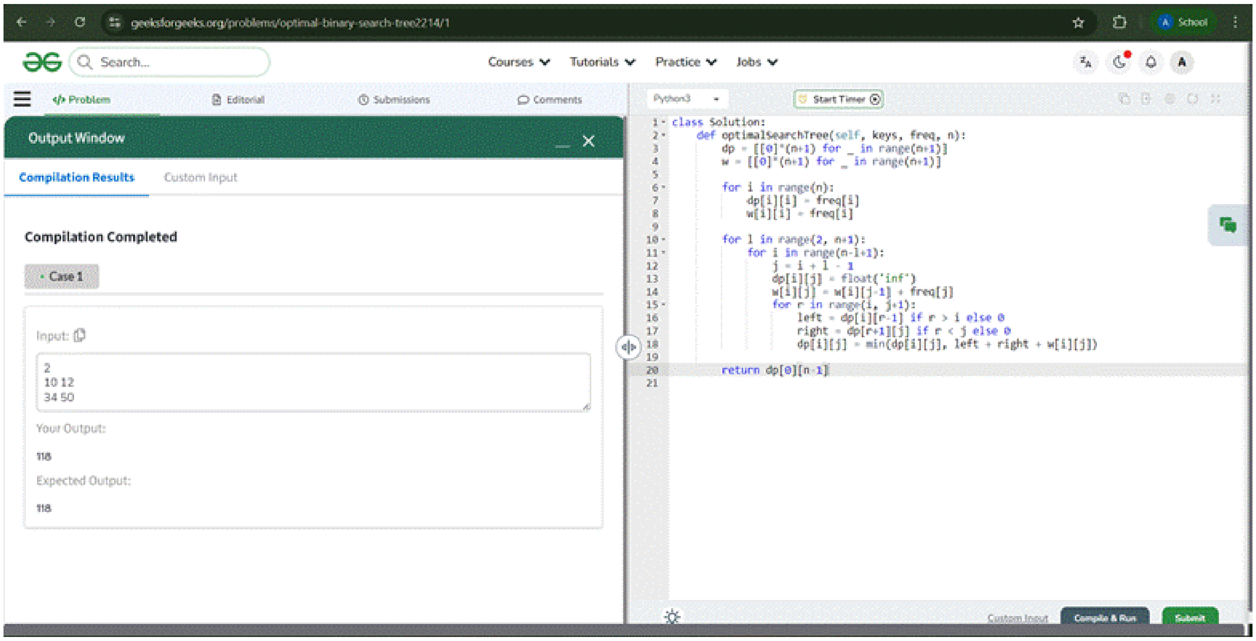


```
PS C:\Users\ansha\OneDrive\Desktop\RBU\Coding\C program\daa> cd "c:\Us
ha\OneDrive\Desktop\RBU\Coding\C program\daa\" ; if ($?) { gcc daa6.c
 } ; if ($?) { .\daa6 }
Enter the size of the successful array:-4
Enter the size of the unsuccessful array:-5
Enter the elements for successful keys:-
0.1
0.2
0.4
0.3
Enter the elements for unsuccessful search between keys:-
0.05
0.1
0.05
0.05
0.1
The Minimum expected cost of the Optimal Binary Search Tree: 2.900000
PS C:\Users\ansha\OneDrive\Desktop\RBU\Coding\C program\daa> 
```

Task2:

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

Output:



Compilation Completed

Case 1

Input:

```
2
10 12
34 50
```

Your Output:

```
118
```

Expected Output:

```
118
```

```python
class Solution:
    def optimalSearchTree(self, keys, freq, n):
        dp = [[0]*(n+1) for _ in range(n+1)]
        w = [[0]*(n+1) for _ in range(n+1)]

        for i in range(n):
            dp[i][i] = freq[i]
            w[i][i] = freq[i]

        for l in range(2, n+1):
            for i in range(n-l+1):
                j = i + l - 1
                dp[i][j] = float('inf')
                w[i][j] = w[i][j-1] + freq[j]
                for r in range(i, j+1):
                    left = dp[i][r-1] if r > i else 0
                    right = dp[r+1][j] if r < j else 0
                    dp[i][j] = min(dp[i][j], left + right + w[i][j])

        return dp[0][n-1]
```