

Practical 6

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

int main() {
    int n;
    scanf("%d", &n);

    int keys[MAX];
    double p[MAX], q[MAX + 1];

    for (int i = 0; i < n; i++)
        scanf("%d", &keys[i]);

    for (int i = 0; i < n; i++)
```

```

    scanf("%lf", &p[i]);
for (int i = 0; i <= n; i++)
    scanf("%lf", &q[i]);

double e[MAX + 1][MAX + 1] = {0};
double w[MAX + 1][MAX + 1] = {0};
int root[MAX + 1][MAX + 1] = {0};

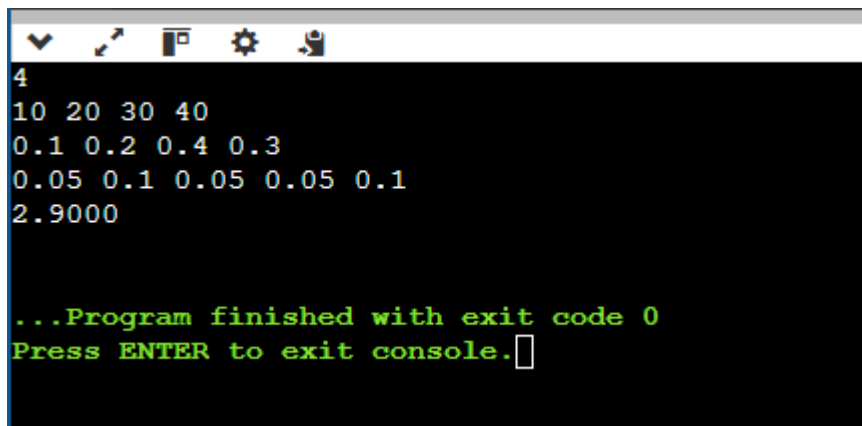
for (int i = 0; i <= n; i++) {
    e[i][i] = q[i];
    w[i][i] = q[i];
}

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

printf("%.4lf\n", e[0][n]);
return 0;
}

```

Output :

A screenshot of a console window with a dark background and a light gray title bar. The title bar contains several icons: a checkmark, a magnifying glass, a square, a gear, and a person. The console output is as follows:

```
4
10 20 30 40
0.1 0.2 0.4 0.3
0.05 0.1 0.05 0.05 0.1
2.9000

...Program finished with exit code 0
Press ENTER to exit console.
```

Task 2:

Code :

```
class Solution {
public:
    int optimalSearchTree(int keys[], int freq[], int n) {
        int cost[n][n];

        for (int i = 0; i < n; i++)
            cost[i][i] = freq[i];

        for (int L = 2; L <= n; L++) {
            for (int i = 0; i <= n - L; i++) {
                int j = i + L - 1;
                cost[i][j] = INT_MAX;

                int sum = 0;
                for (int k = i; k <= j; k++)
                    sum += freq[k];

                for (int r = i; r <= j; r++) {
                    int c = ((r > i) ? cost[i][r - 1] : 0) +
                        ((r < j) ? cost[r + 1][j] : 0) + sum;
                    if (c < cost[i][j])
                        cost[i][j] = c;
                }
            }
        }

        return cost[0][n - 1];
    }
};
```

OUTPUT:

The screenshot displays a coding platform interface with a dark theme. On the left, the 'Output Window' shows 'Compilation Results' for 'Custom Input' by 'Y.O.G.I. (AI Bot)'. It confirms the 'Problem Solved Successfully' with 104/104 test cases passed, 1/1 attempts, 100% accuracy, 8/8 points scored, and a time taken of 0.03. Below this, 'Solve Next' buttons for 'Fixing Two nodes of a BST', 'Strictly Increasing Array', and 'Word Wrap' are visible. The main editor on the right shows a C++ solution for an optimal BST problem. The code defines a class 'Solution' with a public method 'optimalSearchTree' that takes 'keys[]', 'freq[]', and 'n' as input and returns an integer. The algorithm uses dynamic programming to calculate the minimum cost of an optimal BST by iterating over all possible root nodes and their subtrees. The bottom of the interface includes a 'Stay Ahead With:' section, a 'Custom Input' field, and 'Compile & Run' and 'Submit' buttons. The system tray at the bottom shows the date and time as 20:26 on 03-11-2025.

```
1 class Solution {
2 public:
3     int optimalSearchTree(int keys[], int freq[], int n) {
4         int cost[n][n];
5         for (int i = 0; i < n; i++)
6             cost[i][i] = freq[i];
7         for (int L = 2; L <= n; L++) {
8             for (int i = 0; i <= n - L; i++) {
9                 int j = i + L - 1;
10                cost[i][j] = INT_MAX;
11                int sum = 0;
12                for (int k = i; k <= j; k++)
13                    sum += freq[k];
14                for (int r = i; r <= j; r++) {
15                    int c = ((r > i) ? cost[i][r - 1] : 0) +
16                        ((r < j) ? cost[r + 1][j] : 0) + sum;
17                    if (c < cost[i][j])
18                        cost[i][j] = c;
19                }
20            }
21        }
22        return cost[0][n - 1];
23    }
24 };
25
26
27
28
29
30
```