Assignment 8

Given sequence k = k_1 < k_2 < ... < k_n of n sorted keys, with a search probability p_i for each key k_i. Build the Binary Search Tree that has the least search cost given the access probability for each key.

```cpp
#include <iostream>

#include <vector>

#include <climits>

using namespace std;

// Function to compute the sum of probabilities for a subarray from i to j

int sum(const vector<int>& prob, int i, int j) {

    int s = 0;

    for (int k = i; k <= j; k++) {

        s += prob[k];

    }

    return s;

}



// Function to find the minimum search cost and build the optimal BST

int optimalBST(const vector<int>& keys, const vector<int>& prob, int n) {

    // dp[i][j] will store the minimum cost for the subtree from i to j

    vector<vector<int>> dp(n, vector<int>(n, 0));


    // w[i][j] will store the sum of probabilities from i to j

    vector<vector<int>> w(n, vector<int>(n, 0));


    // Calculate the sum of probabilities for each subarray

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

        w[i][i] = prob[i];

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

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

        }

    }
```

```cpp
    // Fill the dp table using bottom-up dynamic programming
    for (int len = 1; len <= n; len++) {
        for (int i = 0; i <= n - len; i++) {
            int j = i + len - 1;
            if (len == 1) {
                dp[i][j] = prob[i];
            } else {
                dp[i][j] = INT_MAX;
                for (int r = i; r <= j; r++) {
                    int cost = (r > i ? dp[i][r - 1] : 0) + (r < j ? dp[r + 1][j] : 0) + w[i][j];
                    dp[i][j] = min(dp[i][j], cost);
                }
            }
        }
    }

    return dp[0][n - 1];
}

int main() {
    int n;

    // Take input for number of keys
    cout << "Enter the number of keys: ";
    cin >> n;

    vector<int> keys(n), prob(n);

    // Take input for the keys
    cout << "Enter the keys (in sorted order): ";
```

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

        cin >> keys[i];

    }


    // Take input for the probabilities

    cout << "Enter the frequency for each key: ";

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

        cin >> prob[i];

    }


    // Call the optimalBST function to get the minimum cost

    int minCost = optimalBST(keys, prob, n);


    cout << "The minimum search cost is: " << minCost << endl;


    return 0;

}
```
//OUTPUT