/\*Problem Statement: Given sequence k = k1 <k2 < … <kn of n sorted keys, with a search probability pi for each key ki . Build the Binary search tree that has the least search cost given the access probability for each key?\*/

#include <bits/stdc++.h>

using namespace std;

// Function to construct Optimal BST and return the minimum cost

float optimalBST(const vector<float>& p, int n) {

// cost[i][j] = minimum cost of BST that contains keys ki to kj

vector<vector<float>> cost(n + 2, vector<float>(n + 1, 0));

vector<vector<float>> sum(n + 2, vector<float>(n + 1, 0));

// Initialize cost[i][i - 1] = 0 for all i (empty tree cost)

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

cost[i][i - 1] = 0;

}

// Calculate prefix sums of probabilities for easy sum lookup

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

sum[i][i] = p[i];

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

sum[i][j] = sum[i][j - 1] + p[j];

}

// l = chain length (number of keys in subproblem)

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

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

int j = i + l - 1;

cost[i][j] = numeric\_limits<float>::max();

// Try making each key in [i..j] the root

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

float c = cost[i][r - 1] + cost[r + 1][j];

if (c < cost[i][j]) {

cost[i][j] = c;

}

}

// Add sum of probabilities from i to j

cost[i][j] += sum[i][j];

}

}

return cost[1][n];

}

int main() {

int n;

cout << "Enter the number of keys: ";

cin >> n;

vector<float> p(n + 1); // p[1] to p[n]

cout << "Enter the probabilities for each key (sorted keys assumed):\n";

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

cin >> p[i];

}

float minCost = optimalBST(p, n);

cout << "Minimum expected search cost: " << minCost << endl;

return 0;

}