Practical 8

```
#include <iostream>
#include <limits.h>
using namespace std;

struct Node {
    int key;
    Node *left, *right;
    Node(int k) {
        key = k;
        left = right = NULL;
    }
};

// Function to calculate the optimal cost using dynamic programming
int findOptimalCost(int keys[], int freq[], int n, int cost[][100], int root[][100]) {
    // Initialize the cost and root arrays
    for (int i = 0; i < n; i++) {
        cost[i][i] = freq[i];
        root[i][i] = i;
    }

    // Fill the cost and root arrays for subproblems of length 2 to n
    for (int length = 2; length <= n; length++) {
        for (int i = 0; i <= n - length; i++) {
            int j = i + length - 1;
            cost[i][j] = INT_MAX;
            int totalFreq = 0;
            for (int k = i; k <= j; k++) {
                totalFreq += freq[k];
            }

            // Calculate the minimum cost for the subproblem [i, j]
            for (int k = i; k <= j; k++) {
                int currentCost = (k > i ? cost[i][k - 1] : 0) + (k < j ? cost[k + 1][j] : 0) + totalFreq;
                if (currentCost < cost[i][j]) {
                    cost[i][j] = currentCost;
                    root[i][j] = k;
                }
            }
        }
    }
    return cost[0][n - 1];
```

```
}

// Function to build the optimal BST from the root array
Node* buildOptimalBST(int keys[], int freq[], int start, int end, int root[][100]) {
    if (start > end) {
        return NULL;
    }

    int r = root[start][end];
    Node* node = new Node(keys[r]);
    node->left = buildOptimalBST(keys, freq, start, r - 1, root);
    node->right = buildOptimalBST(keys, freq, r + 1, end, root);
    return node;
}

// Inorder traversal to print the BST
void inorderTraversal(Node* root) {
    if (root == NULL) {
        return;
    }
    inorderTraversal(root->left);
    cout << root->key << " ";
    inorderTraversal(root->right);
}

int main() {
    int ks;
    cout << "Enter number of keys: ";
    cin >> ks;

    cout << "Enter keys: ";
    int keys[ks];
    for (int i = 0; i < ks; i++) {
        cin >> keys[i];
    }

    int freq[ks];
    cout << "Enter frequencies of keys: ";
    for (int i = 0; i < ks; i++) {
        cin >> freq[i];
    }

    // Define the maximum size (100, assuming it won't exceed 100 keys)
    int cost[100][100] = {0};  // cost[i][j] will store the minimum cost for keys[i..j]
```

```
    int root[100][100] = {0};  // root[i][j] will store the root index of the optimal BST for keys[i..j]

    // Calculate optimal cost and fill the root table
    int n = ks;
    int optimalCost = findOptimalCost(keys, freq, n, cost, root);
    cout << "Optimal cost of the BST: " << optimalCost << endl;

    // Build the optimal BST
    Node* rootNode = buildOptimalBST(keys, freq, 0, n - 1, root);

    // Inorder traversal of the optimal binary search tree
    cout << "Inorder traversal of the optimal binary search tree: ";
    inorderTraversal(rootNode);

    return 0;
}
```