1. **Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy. (DAA)**

#include <iostream>

using namespace std;

int knapSack(int W, int weights[], int values[], int items) {

int table[items + 1][W + 1];

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

for (int w = 0; w <= W; w++) {

if (i == 0 || w == 0) {

table[i][w] = 0;

} else if (weights[i - 1] <= w) {

table[i][w] = max(values[i - 1] + table[i - 1][w - weights[i - 1]], table[i - 1][w]);

} else {

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

}

}

}

return table[items][W];

}

int main() {

int n;

cout << "Enter number of items: ";

cin >> n;

int values[n];

int weights[n];

cout << "Enter values of items (space-separated): ";

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

cin >> values[i];

}

cout << "Enter weights of items (space-separated): ";

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

cin >> weights[i];

}

int W;

cout << "Enter Knapsack Capacity: ";

cin >> W;

int maxProfit = knapSack(W, weights, values, n);

cout << "Maximum profit that can be achieved: " << maxProfit << endl;

return 0;

}