#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// State class to represent (weight, value, chosen items)

class State {

public:

int weight, value;

vector<int> chosen; // 0/1 vector indicating taken items

State(int w, int v, int n) : weight(w), value(v), chosen(n, 0) {}

State(int w, int v, vector<int> c) : weight(w), value(v), chosen(c) {}

};

// Bubble sort for states (by weight ascending, then value descending if equal)

void sortStates(vector<State> &states) {

int n = states.size();

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (states[j].weight > states[j + 1].weight ||

(states[j].weight == states[j + 1].weight && states[j].value < states[j + 1].value)) {

swap(states[j], states[j + 1]);

}

}

}

}

// Purge dominated states (remove worse ones)

void purge(vector<State> &states, int W) {

sortStates(states);

vector<State> filtered;

int maxValue = -1;

for (auto &s : states) {

if (s.weight <= W && s.value > maxValue) {

filtered.push\_back(s);

maxValue = s.value;

}

}

states = filtered;

}

// Merge & Purge knapsack

pair<int, vector<int>> knapsackSetMergePurge(vector<int> &weights, vector<int> &values, int W) {

int n = weights.size();

vector<State> states;

states.push\_back(State(0, 0, n)); // initial state: no items chosen

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

vector<State> newStates;

// Try including current item

for (auto &s : states) {

int newW = s.weight + weights[i];

int newV = s.value + values[i];

if (newW <= W) {

vector<int> newChosen = s.chosen;

newChosen[i] = 1; // mark item as taken

newStates.push\_back(State(newW, newV, newChosen));

}

}

// Merge old and new states

states.insert(states.end(), newStates.begin(), newStates.end());

// Purge dominated states

purge(states, W);

}

// Find best value and corresponding chosen array

int bestValue = 0;

vector<int> bestChosen(n, 0);

for (auto &s : states) {

if (s.value > bestValue) {

bestValue = s.value;

bestChosen = s.chosen;

}

}

return {bestValue, bestChosen};

}

int main() {

int n, W;

cout << "Enter number of items: ";

cin >> n;

cout << "Enter knapsack capacity: ";

cin >> W;

vector<int> weights(n), values(n);

cout << "Enter weights of items:\n";

for (int i = 0; i < n; i++) cin >> weights[i];

cout << "Enter values of items:\n";

for (int i = 0; i < n; i++) cin >> values[i];

auto result = knapsackSetMergePurge(weights, values, W);

cout << "Maximum value in Knapsack = " << result.first << endl;

cout << "Items taken (0=not taken, 1=taken): ";

for (int x : result.second) cout << x << " ";

cout << endl;

return 0;

}

**Output:**

E:\PVG\TY\sem5\DAA\programs>cd "e:\PVG\TY\sem5\DAA\programs\" && g++ knap01.cpp -o knap01 && "e:\PVG\TY\sem5\DAA\programs\"knap01

Enter number of items: 3

Enter knapsack capacity: 5

Enter weights of items:

3

2

4

Enter values of items:

5

10

2

Maximum value in Knapsack = 15

Items taken (0=not taken, 1=taken): 1 1 0