```
#include <iostream>
using namespace std;

// global variables
// problem instance globals (note: I waste index 0 to match textbook )
float w[] = {-1, 4, 7, 5, 3 };      // item weights - previously SORTED in ascending order of profit per   ↙
    weight
float p[] = {-1, 40, 63, 25, 12 };  // item profits
int n = 4;  // number of items
float W = 10.0; // knapsack weight limit

// best solution found (note: I waste index 0 to match textbook )
float maxprofit = 0.0;
bool bestset[] = {false, false, false, false, false };

// global node state
bool includes[] = {false, false, false, false, false };

bool promising(int i, float weight, float profit) {
    float totweight, boundOnProfit;
    int k;

    if (weight > W)
        return false;

    totweight = weight;
    boundOnProfit = profit;
    for (int j=i+1; j<=n; j++) {
        if (totweight + w[j] <= W) {
            totweight += w[j];
            boundOnProfit += p[j];
        } else {
            k = j;
            boundOnProfit += p[k] * (W - totweight) / w[k];
            break;
        } // end if
    } // end for

    return boundOnProfit > maxprofit;
} // end promising

void knapsack(int i, float weight, float profit) {
    if (weight <= W && profit > maxprofit) {
        maxprofit = profit;
        for (int index=1; index <= n; index++) {
            bestset[index] = includes[index];
        } // end for
    } // end if

    if (promising(i, weight, profit)) {
        includes[i+1] = true;
        knapsack(i+1, weight + w[i+1], profit + p[i+1]);
        includes[i+1] = false;
        knapsack(i+1, weight, profit);
    } // end if

} // end knapsack

int main() {
    knapsack(0, 0.0, 0.0);  // inital call to kick off the backtracking

    // print results
    cout << "Best profit: " << maxprofit << endl;
    cout << "Items stolen should be:" << endl;
    for (int index = 1; index <= n; index++) {
        if (bestset[index]) {
            cout << "item " << index << endl;
        } // end if
    } // end for

} // end main
```