class Knapsack {

static int solveKnapsack(int[] profits, int[] weights, int capacity) {

//TODO: Write - Your - Code

return -1;

}

}

SOLUTION:

class Knapsack {

static int solveKnapsack(int[] profits, int[] weights, int capacity) {

// basic checks

if (capacity <= 0 || profits.length == 0 || weights.length != profits.length)

return 0;

int n = profits.length;

int[] dp = new int[capacity + 1];

// if we have only one weight, we will take it if it is not more than the

// capacity

for (int c = 0; c <= capacity; c++) {

if (weights[0] <= c)

dp[c] = profits[0];

}

// process all sub-arrays for all the capacities

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

for (int c = capacity; c >= 0; c--) {

int profit1 = 0, profit2 = 0;

// include the item, if it is not more than the capacity

if (weights[i] <= c)

profit1 = profits[i] + dp[c - weights[i]];

// exclude the item

profit2 = dp[c];

// take maximum

dp[c] = Math.max(profit1, profit2);

}

}

return dp[capacity];

}

}