Knapsack Problem 0/1

Given a set of items where each item has a **weight** and a **value**. And given a knapsack with max weight **capacity**, determine the maximum *value* that can be placed into the knapsack without going over the capacity.

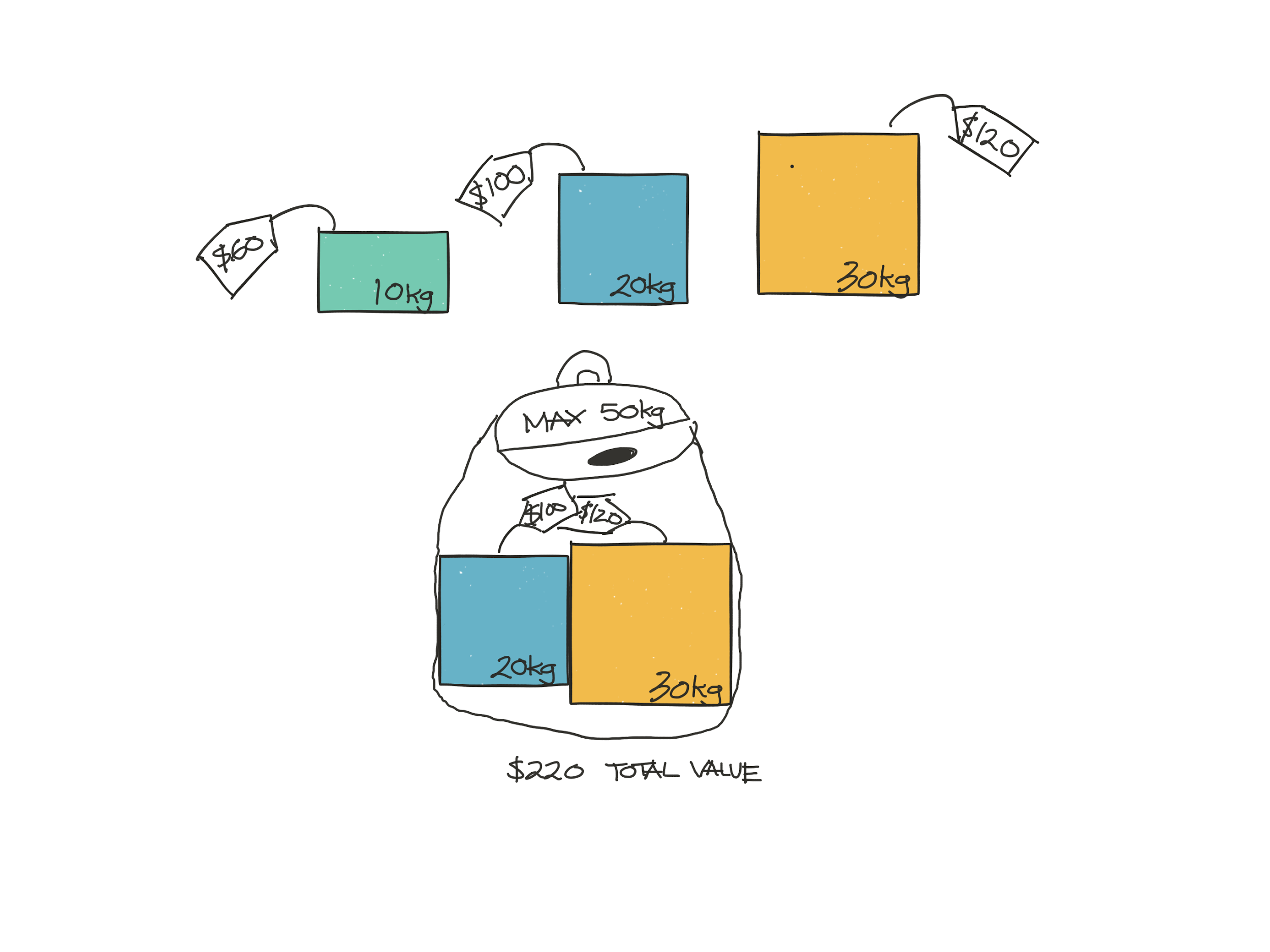
Input: An integer array of weights

An integer array of values

The ith item is weights[i] and values[i].

Output: Integer of maximum total value

# Example

Input:   
 value = [60, 100, 120]

weight = [10, 20, 30]

capacity = 50

Output: 220

# 

# Constraints

Intermediate Advanced

Time Complexity: O(2N) O(KN)  
Auxiliary Space Complexity: O(N) O(K)

K is the capacity of the knapsack, N is the number of items

# Solution

Utilize recursion for the intermediate solution.

Use dynamic programming w/ memoization of the recursive solution or use tabulation

Advanced Tabulation Approach:

1. There will be two loops, the outer loop selects an item to to be the current item
2. The inner loop iterates from 0 to the knapsack capacity and we try to determine the maximum value at each incremental capacity.
   1. If adding the current item results in a higher value, we update our max value at that capacity.

Detailed Pseudocode for Tabulation Approach:

1. Instantiate an array called ‘*finalMax*’ with a length equal to one plus the capacity
2. Loop through the number of items
   1. Inside this first loop, make a copy of *‘finalMax’* called *‘tempMax’*
   2. Create an inner loop and increment a variable *‘cap’* from 0 to the input capacity
   3. If the *weight* of the current item is less than or equal to the current *cap,*
      1. Check to see if the value at tempMax[cap - weight] plus the value of the current item leads to a greater value than what is currently in tempMax[cap].
         1. If greater, replace the value at tempMax[cap]
3. After performing both loops, the *finalMax* should contain the max values obtainable for each capacity level. Since we only care about the max value at our input capacity, return finalMax[capacity].

# Code Solution

function knapsack(values, weights, capacity) {

var finalMax = new Int32Array(capacity + 1); // create a new array of 0’s

for(var i = 0; i < weights.length; i++) {

var weight = weights[i];

var value = values[i];

var tempMax = finalMax.slice(); // make a copy of the finalMax array

for(var cap = 0; cap <= capacity; cap++) {

if(cap - weight >= 0) {

tempMax[cap] = Math.max(tempMax[cap], finalMax[cap - weight] + value);

}

}

finalMax = tempMax;

}

return finalMax[capacity];

}

# Resources

<http://www.geeksforgeeks.org/dynamic-programming-set-10-0-1-knapsack-problem/>

<https://en.wikipedia.org/wiki/Knapsack_problem>