Matlab Code

clear all; close all;
% DEFINITIONS
% needs value and weight matrix
% track_value = [value;weight]

value = [ 1 7 20 5 2 ]
weight = [8 5 2 10 6]
MaximumWeight = 20

n = length(value)
track_value = []

% remember_temps = [0]

%% Binomial

for i=2:n % all ways to choose 2,3,4,...,n items from weight vector
    bin_choices = nchoosek(weight,i) % creates all choices of nchoosek objects from weight

    for j=1:length(bin_choices(:,1)) % cycle through each combination
        % code here
    end
end

for i=2:n % all ways to choose 2,3,4,...,n items from weight vector
    bin_choices = nchoosek(weight,i) % creates all choices of nchoosek objects from weight

    for j=1:length(bin_choices(:,1)) % cycle through each combination
        % code here
    end
end
for m=1:i % takes one combo and goes through each of its weights
    temp = find(weight==bin_choices(j,m));

    for p=1:length(temp)
    % if temp == find(remember_temps==temp) % make sure don't add up weights already added
    % break
    %
        pack_weight(1,m) = weight(temp(1,p))
        Total_Weight = sum(pack_weight)
        pack_value(1,m) = value(temp(1,p))
        Total_Value = sum(pack_value)

        % remember_temps(length(remember_temps+1)) = temp(1,p)
    end
    end

if Total_Weight <= MaximumWeight % if weight of combination is under max weight
    % add it to track_value
    % which keeps track of
    % all good combinations
    track_value(:,length(track_value)+1) = [Total_Value;Total_Weight]
else
    track_value = track_value % else don't add it
end

%reset pack_weight and pack_value
pack_weight = zeros(1,n);
pack_value = zeros(1,n);

end

%%%%%%%%%%%%%%%%%%%%% Bridge Druken
% 04/12/08
% Senior Project:
% "A Walk Through the Forest: The Knapsack Problem in Graph Theory"
% Investigating the Knapsack Problem
%
clear all; close all;

% potentialItems = [value;weight]
% MaximumWeight = maximum weight backpack can hold

value = [1 7 20 5 2];
weight = [8 5 2 10 5];
MaximumWeight = 20;

n = length(value);

%% Greedy Algorithm

[sorted_value, initial_index] = sort(value,'descend'); % sorts value largest to smallest
% initial_index records
% where value came from

total_W = 0; % weight of backpack
for i = 1:n
    diff = MaximumWeight - total_W; % difference between the maximum weight and weight of backpack
    new_value = sorted_value(i); % the ith largest (sorted) value
    W = weight(initial_index(i)); % weight of the ith largest (sorted) value
    if W <= diff % check to see if the weight of the ith value can fit in the backpack
        backpack(i) = new_value; % add value into backpack
        Weight(i) = W; % add weight into weight vector
        total_W = sum(Weight); % sum weight of backpack thus far
        last_n = i; % remembers index for last item added
    end
end

Greedy_Backpack = backpack % backpack values
total_Value = sum(backpack) % print out total value
Greedy_Backpack_Weight = Weight % backpack weight
total_Weight = total_W % print out total weight

track_trial = [total_Value; total_Weight]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Senior Project:
% "A Walk Through the Forest: The Knapsack Problem in Graph Theory"
% Investigating the Knapsack Problem
%
clear all; close all;

% potentialItems = [value;weight]
% MaximumWeight = maximum weight backpack can hold

value = [ 1 7 20 5 2 ];
weight = [8 5 2 10 5];
MaximumWeight = 20;

n = length(value);

%% "Not so greedy" Algorithm
% starts with Greedy Algorithm

[sorted_value, initial_index] = sort(value,'descend'); % sorts value largest to smallest
% initial_index
% records where
% value came from

total_W = 0; % weight of backpack
for i = 1:n
    diff = MaximumWeight - total_W; % difference between the maximum weight and weight of backpack
    new_value = sorted_value(i); % the ith largest (sorted) value
    W = weight(initial_index(i)); % weight of the ith largest (sorted) value
    if W <= diff % check to see if the weight of the ith value can fit in the backpack
        backpack(i) = new_value; % add value into backpack
        Weight(i) = W; % add weight into weight vector
        total_W = total_W + W; % sum weight of backpack thus far
        last_n = i; % remembers index for last item added
    end
end

Greedy_Backpack = backpack % backpack values
total_Value = sum(Greedy_Backpack) % print out total value
Greedy_Backpack_Weight = Weight % backpack weight

track_trials = [total_Value;total_W] % then goes into Not-so-greedy algorithm

for j=1:n-1
    [heaviest_item, heaviest_index] = max(Greedy_Backpack_Weight); % finds heaviest item and
    % records where it came from
    Greedy_Backpack(heaviest_index) = 0; % takes heaviest item out of backpack vector
    Greedy_Backpack_Weight(heaviest_index) = 0; % takes the weight of the heaviest item out of weight vector
end
total_Weight = total_Weight - heaviest_item; \% subtracts the weight of heaviest item out of backpack
for i = last_n+1:n \% for loop cycles through rest of sorted value items
    diff = MaximumWeight - total_Weight; \% diff keeps track of difference between the max weight and current weight
    new_value = sorted_value(i); \% repeat greedy algorithm
    new_weight = weight(initial_index(i));
    if new_weight <= diff \% if weight can fit, add it to backpack
        Greedy_Backpack(i) = new_value;
        Greedy_Backpack_Weight(i) = new_weight;
        total_Weight = sum(Greedy_Backpack_Weight);
        last_n = i;
    end
end

Greedy_Backpack
total_Value = sum(Greedy_Backpack)
Greedy_Backpack_Weight
total_Weight

track_trials(:,length(track_trials)+1) = [total_Value;total_Weight]
end