Kyle Wolf

Problem Set #2

**1a.** Is not correct. If an item with a low weight but disproportionately high profit is at the n-1 index such that it could be combined with other items to form the highest total profit, choosing the earliest first could stop this item from being used.

Ex:

profit = [1, 1, 1, 1, 100]

weight = [2,1,2,3,1]

limit = 5

Here, best result is 102, but algorithm gives 3.

**1b.** Is not correct. It is possible that a heavier item still below or equal to limit would be disproportionately valuable and the result would not be the highest possible total profit.

Ex:

profit = [5, 10, 15, 5, 100]

weight = [1,2,2,3,3]

limit = 3

Here, best result is 100, but algorithm gives 15.

**1c.** It is not correct. If the lighter items are more valuable, or the heavier items are simply not valuable enough to offset their weight, the total profit would not be maximized.

Ex:

profit = [5, 10, 15, 5, 100]

weight = [3,3,2,2,1]

limit = 3

Here, best result is 115, but algorithm gives 5.

**1d.** It is not correct. If the highest value item is heavy enough to be surpassed by the combination of multiple profits of lower weights still below the limit, the profit will not be maximized.

Ex:

profit = [50, 100, 15, 60, 20]

weight = [1,3,3,2,3]

limit = 3

Here, best result is 110, but algorithm gives 100.

**1e.** It is not correct. The lowest value items can still weigh too much to maximize profit.

Ex:

profit = [5, 10, 5, 15, 100]

weight = [3,1,3,2,1]

limit = 3

Here, best result is 115, but algorithm gives 5.

**1f.**

Assume the algorithm is incorrect. Let S = {S1, S2, S3,…, Sn} be the profit[] and weights[] indices selected by our solution, and O = {O1, O2, O3,…,Or} be those in an optimal solution, both ordered from greatest to least profit-to-weight ratios, both with total weights not exceeding the limit. Note the total profit of all O > total profit of all S, as the total profit for an optimal solution must be greater to make our solution incorrect.

Let fTi be the profit-to-weight ratio of job Ti in one of the above sets.

Pf:

Base Case (i = 1): Algorithm selects array index of profit[] and weights[] to produce the highest possible profit-to-weight ratio with a weight value below the limit. Can’t pick any higher ratio.

Inductive Step: Assume fSt-1 <= fOt-1 for 1 < t < n.

I.H. All ratios available to the optimal algorithm are available to our algorithm. We choose one with the highest profit-to-weight ratio so the total weight is below the limit, implying fSt <= fOt

Any ratio selected as On+1 would still be available to our algorithm. This implies our algorithm’s total profit would not be less than the optimal’s, as previously stated. =><=

By contradiction, the algorithm is correct.

**1g.** This does not work. As the highest ratios of weight-to-profit come from a high weight with a low value, naturally selecting the highest ratio will lead to a relatively low profit.

Ex:

profit = [2,2,3,2,3]

weight = [8,2,6,10,9]

limit = 10

Here, best result is 5, but algorithm gives 2.

**2a.**

n = size of arrays

float[] PtoW\_ratio = new float[n];

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

PtoW\_ratio[i] = profit[i] / weights[i];

}

selectionSort PtoW\_ratio from greatest to least

for every swap in selectionSort, mimic swaps in profit[] and weights[]

int totalProfit = 0;

int weight = 0;

int i = 0;

while(true){

weight += weights[i];

if (weight == limit){

totalProfit += profit[i];

print totalProfit;

return;

}

if (weight > limit){

weight -= weights[i];

break;

}

totalProfit += profit[i];

i++;

}

int remainder = limit – weight;

totalProfit += profit[i] \* (remainder / weights[i]);

print totalProfit;

**2b.**

The worst case runtime is Ө(n^2). The time it takes to do the selection sort dominates the other operations, which runs in Ө(n^2) time in its worst case. Though three arrays are being adjusted during the selection sort, the sorting actions are simply mimicked. 3 Ө(n^2) = Ө(n^2).

**2c.**

Assume the algorithm is incorrect. Let S = {S1, S2, S3,…, Sn} be the profit[] and weights[] indices selected by our solution, and O = {O1, O2, O3,…,Or} be those in an optimal solution, both ordered from greatest to least profit-to-weight ratios, both with total weights either equal to the limit, or at most exceeding the limit by one value, Sn or Or. Note the total profit of all O > total profit of all S, as the total profit for an optimal solution must be greater to make our solution incorrect.

Let fTi be the profit-to-weight ratio of job Ti in one of the above sets.

Pf:

Base Case (i = 1): Algorithm selects array index of profit[] and weights[] to produce the highest possible profit-to-weight ratio. Can’t pick any higher ratio. If this ratio gives a weight above the limit, a portion k is taken so the weight of the ratio \* k = limit, and the total profit = profit of the ratio \* k.

Inductive Step: Assume fSt-1 <= fOt-1 for 1 < t < n.

I.H. All ratios available to the optimal algorithm are available to our algorithm. We choose one with the highest profit-to-weight ratio so the total weight is below or equal to the limit, or we can take a portion k of that weight and profit so the total weight is equal to the limit, implying fSt <= fOt

Any ratio selected as On+1 would still be available to our algorithm. This implies our algorithm’s total profit would not be less than the optimal’s, as previously stated. =><=

By contradiction, the algorithm is correct.