Zijian Gao
26319560
4/7/2016
CS311 HW4

16.1-2.)

The given algorithm is greedy because instead of filling activities from earliest finish time to latest, it chooses the latest start time and fills in the reverse direction towards the earliest start time. As this is just the greedy algorithm that chooses the earliest finish time choosing activities in reverse, it likewise will maximize the set of compatible activities.

IH: the first i activities chosen by the greedy algorithm yields an optimal solution S, the i'th activity is S_i

BC: i = 1, this selects the last activity to start since no other activities have been chosen. Let k be some activity in some other optimal solution that starts before S_1 and ends during S_1. No other activity may start after k in that other optimal solution since S_1 is by definition the last activity to start. Since this is true, they are both optimal choices for that particular timespan and k can be replaced with S_1, yielding still an optimal solution.

IS: Assume the hypothesis for i. Proof for i+1. For S_{i+1} take a similar arbitrary k activity in some other optimal solution that starts before S_{i+1} and ends during. Because the IH gives an optimal solution for i, there can exist no activity that starts after k ends but before S_i begins. If this were not the case, that activity would be a part of the optimal solution, yielding a contradiction. By the same logic as in the base case, k can be swapped with S_{i+1}, yielding another optimal set of activities.

16.2-3.)

Greedy Algorithm:

1. Sort items by increasing weight
2. Pick the item with the least weight, add to knapsack.
3. Continue to add the next item with the least weight until the knapsack runs out of space

Proof of correctness:
This algorithm is correct because of the given premise that the order of the items when sorted in increasing weight is the same as the order in decreasing value. This means that the item with the minimum weight is always the item with the maximum value. Take two solutions G(greedy) and Y(some other optimal solution). the weight of G_i will always be less than or equal to the weight of Y_i because the algorithm always chooses the item with the least weight to add to the knapsack. Because of this, the value of G_i will always be greater than or equal to the value of Y_i and the greedy algorithm yields an optimal solution.

Runtime:
Given n items. The sort runs in $\theta(n \log n)$ as this is the fastest the items can be sorted. Step 2-3 takes $\theta(n)$ time as the most items that could be added to the knapsack would be all of the n items. $\theta(n \log n)$ dominates and is the runtime of this algorithm.