Question #1: Interval Scheduling

Given n jobs, each an interval with a starting time s_i and finishing time t_i choose the maximum number of them that don't intersect. I have already gone over the problem and its solution which is the following:

Input: Integer n, set of intervals $J = \{j_1 = [s_1, f_1], j_2 = [s_2, f_2], \ldots, j_n = [s_n, f_n]\}$.
Output: Schedule with maximum number of non-intersecting intervals.

```
1 Chosen ← ∅
2 Sort(J, by increasing finishing time)
3 for i ← 1 to n do
4     if j_i doesn't intersect with any job in Chosen then
5         Chosen ← Chosen ∪ {j_i}
6     end
7 end
8 return Chosen
```

We have already shown in class that this greedy job scheduling is optimal using a "promising argument" that argued by induction that for all i after the ith step of the loop the choices we have made so far (the set *Chosen*) can be completed to some optimal solution.
What I want you to do is to go over the problem and the algorithm again and prove its correctness with the following charging argument:
Consider a set G output by the greedy algorithm and a set O which is optimal. We will show how to match each job in G with at most one job in O which shows that the set G can't be smaller than O. Look at jobs in the set G in the order greedy has put them there, i.e. in increasing order of finishing time. Whenever greedy puts a job in G match it with all the jobs in O that intersect it (including itself if it is common between G and O) and delete them from O. There are two important observations:

1. When a job j_i is considered all the jobs it is matched to intersect with it at its finishing time. This is because the job j_i that greedy considers is always the job with the smallest finishing time among those that don't intersect any job previously selected by greedy. What this implies is that all these jobs intersect each other too at the point f_i so not more than one of them can be in O.

2. Each job in O has to be matched to some job in G because if it is not when it was considered by greedy it didn't intersect any of the previously selected jobs in G (otherwise it would have been deleted from O by this point and matched to something) so greedy would select it!

These two things together give us a one-to-one and on-to mapping from G to O showing that they have the same size. In other words G is optimal.

Question #2: Fractional Knapsack (only if you had extra time)

You have a knapsack of capacity S and n items the ith of which weights w_i and is worth v_i. But all these items can be broken into pieces, they are valuable minerals lets say. You want to pack your knapsack with the maximum value. So if $S = 10, w_1 = 6, v_1 = 1, w_2 = 5, v_1 = 2, w_3 = 10, v_1 = 2$ then we can take 5 units of the second item (worth \$2) and 5 units of the 3rd (worth \$1) for a total value of \$3.
The algorithm is of course is to sort according to decreasing unit value, and the optimality is more or less trivial.