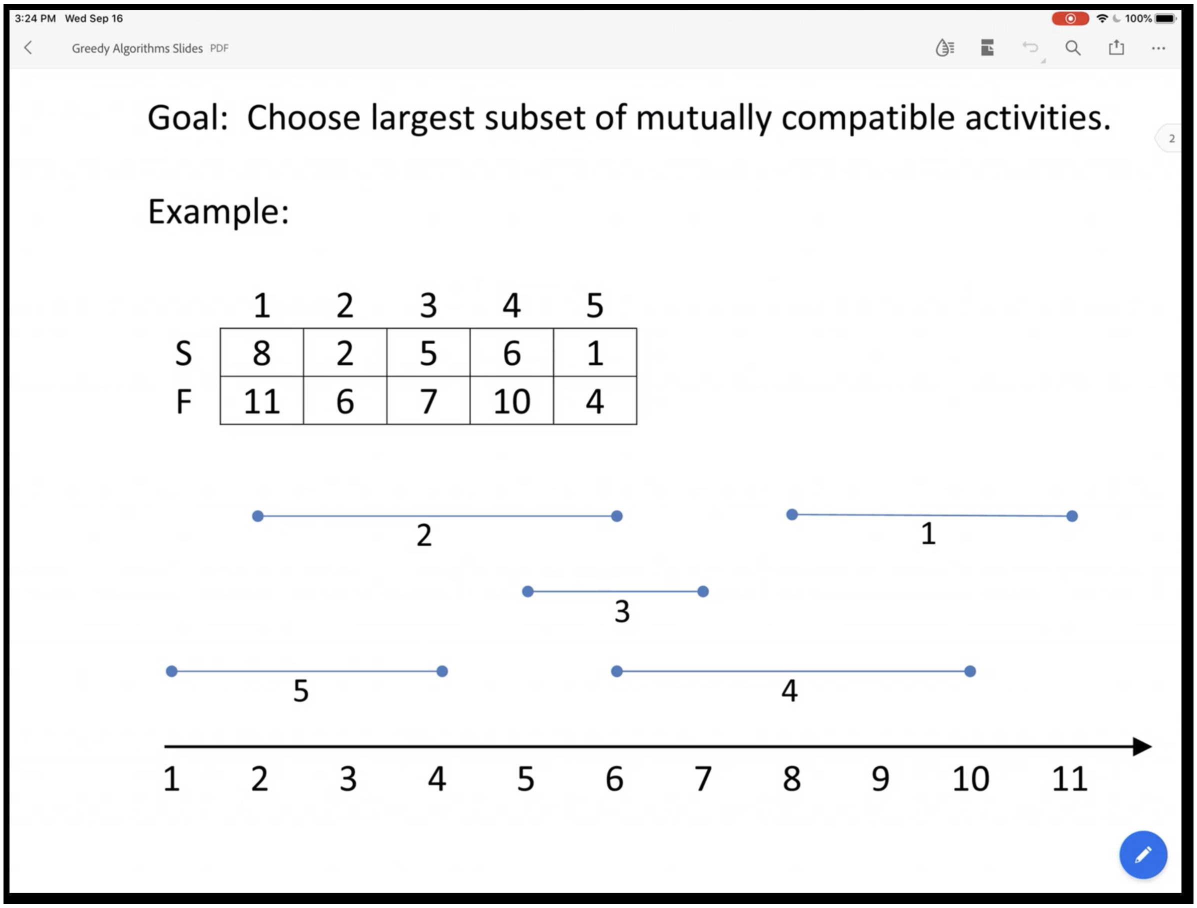
Greedy Algorithms - Part 1

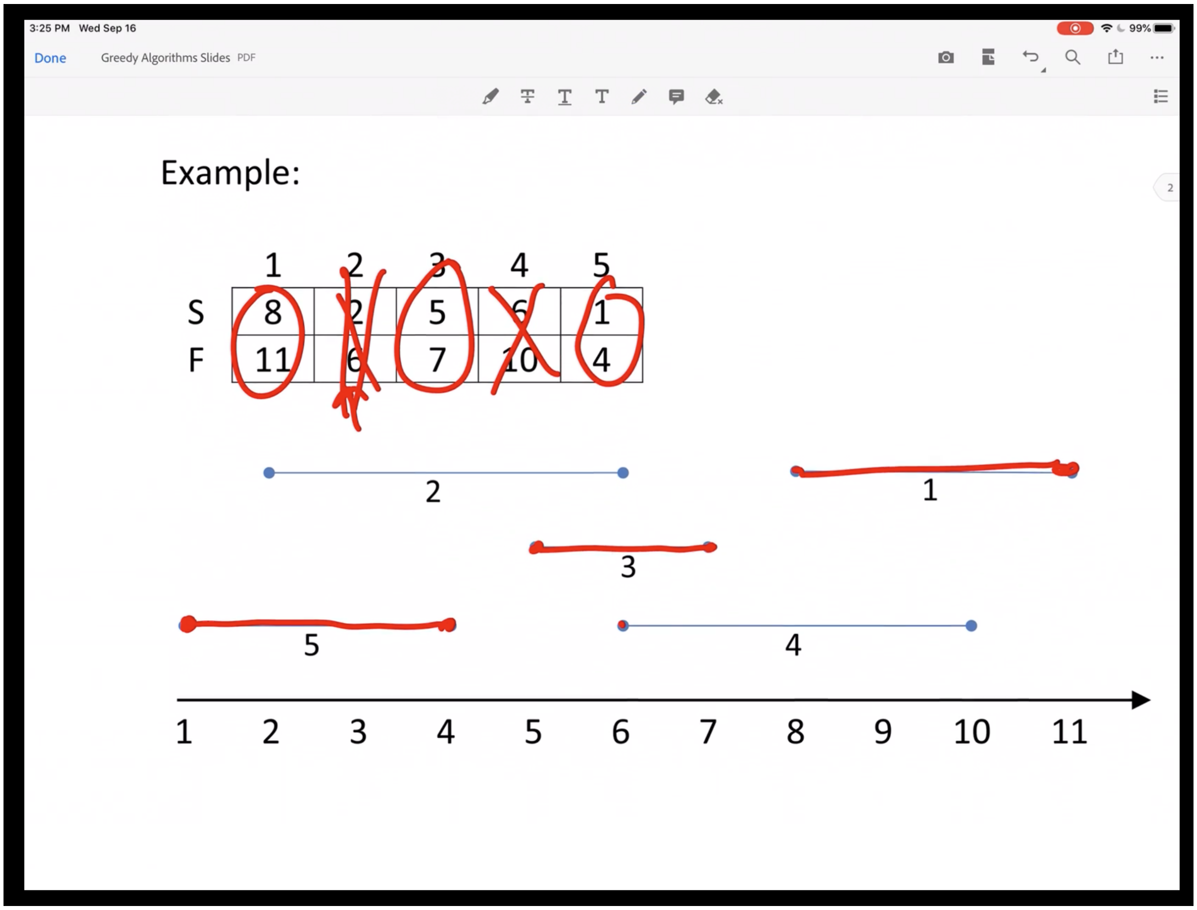
- What are greedy algorithms?
  - Algorithms where we go over each item X in some order and decide whether or not to include item X in the solution.
  - We make decisions as we encounter each item, we're *greedy* about how we take/add stuff.

- What are some examples of greedy algorithms
  - Sorting: insertion sort, selection sort
  - Minimum spanning tree: Kruskal, Prim
  - Single-source shortest paths: Dikjstra's Algorithm

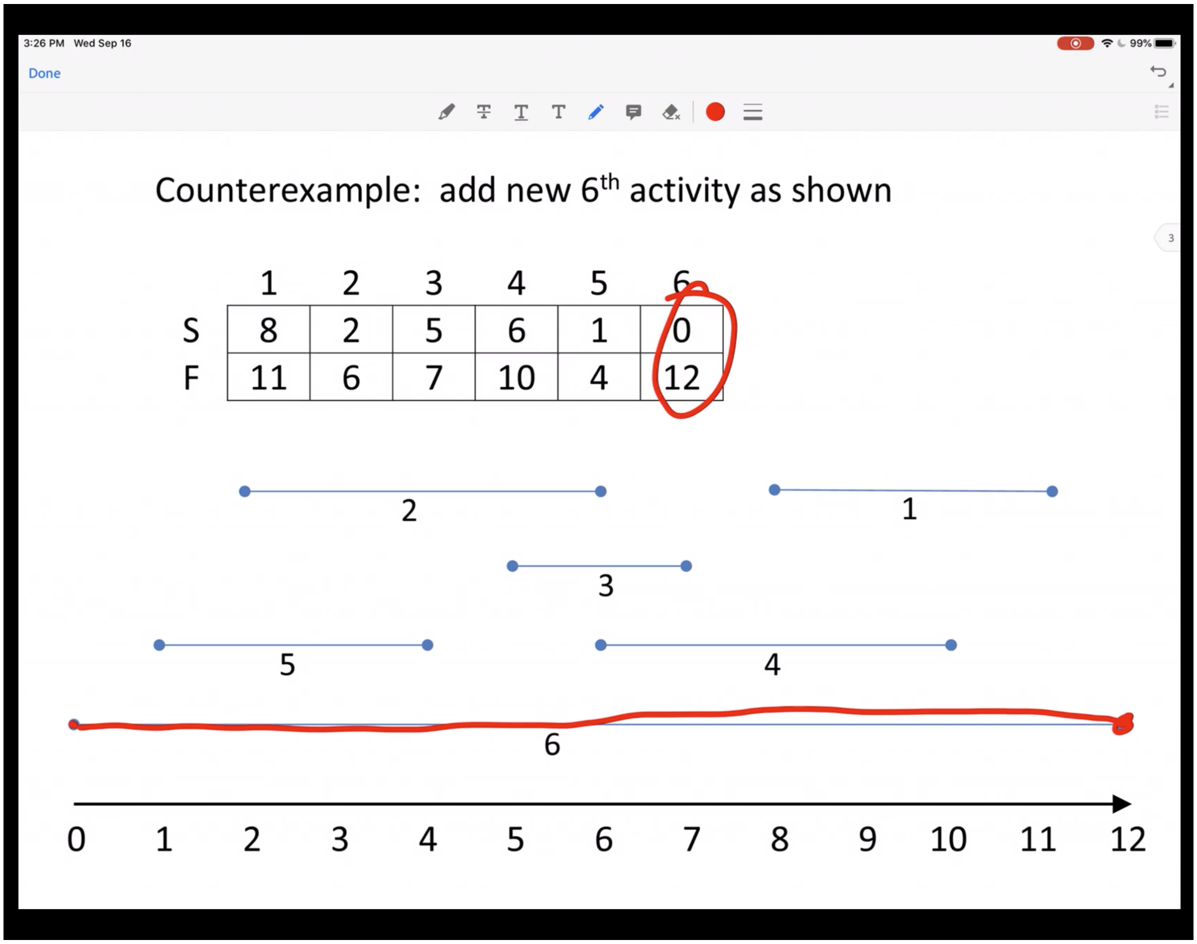
- Do greedy algorithms always work?
  - **No.** There's a lot of 'gotchas' with greedy algorithms that might seem like they work but don't actually work.
- What's the definition of the activity selection problem?
  - You have a list of Activities {1…n}, their Start times S[1…n], and the finish times F[1…n]
  - We'll say that activities j and k are compatible if either F[j] ≤ S[k] or F[k] ≤ S[j].
  - Compatible jobs fit with out overlap, conflicting activities don't.
  - Goal: Choose the largest subset of mutually compatible activities.z



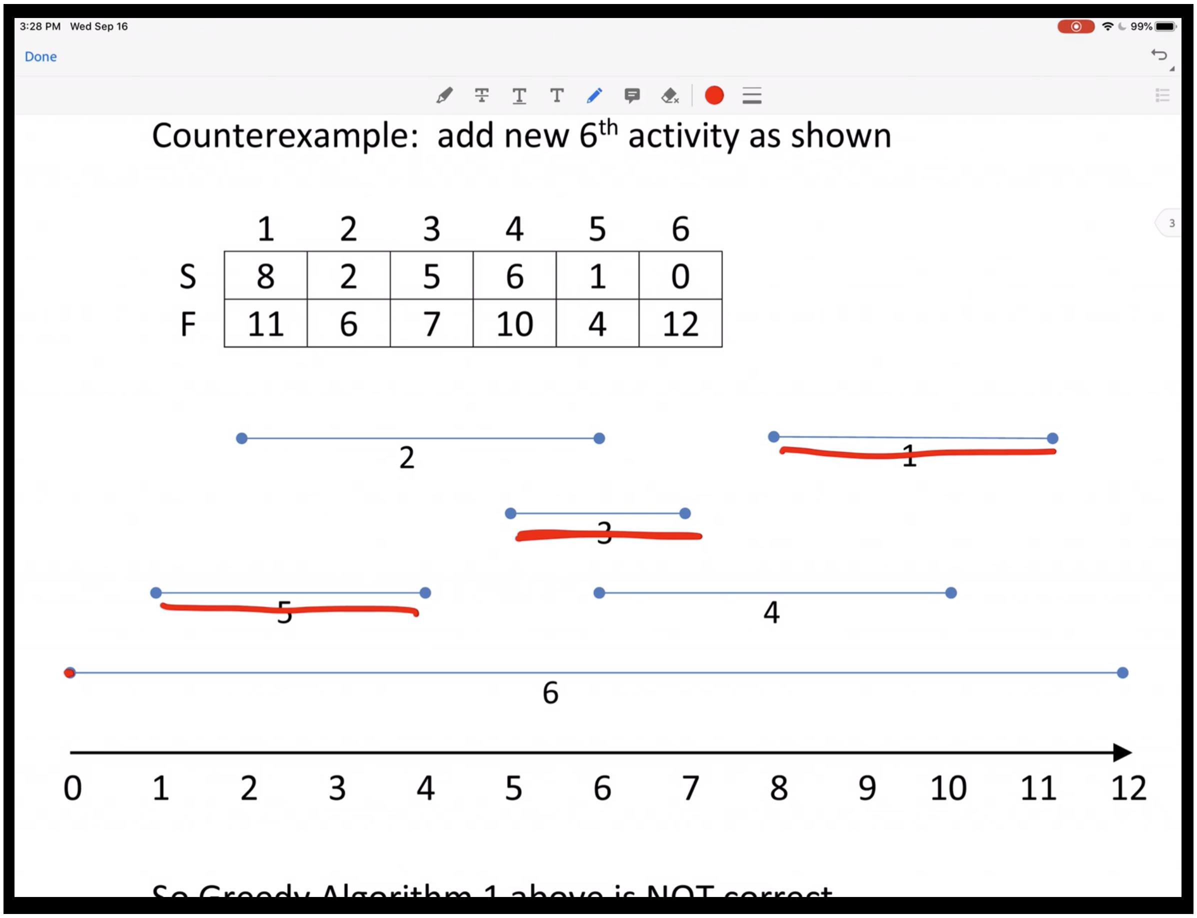
- First greedy approach to activity selection
    - Let's go over each item in order of earliest start time first. If it fits with the rest of the items, we add it. If it doesn't, then it gets left out.
    - This actually works on our first example! This algorithm would select jobs 5, 3, and 1, which is the greatest set of compatible activities.



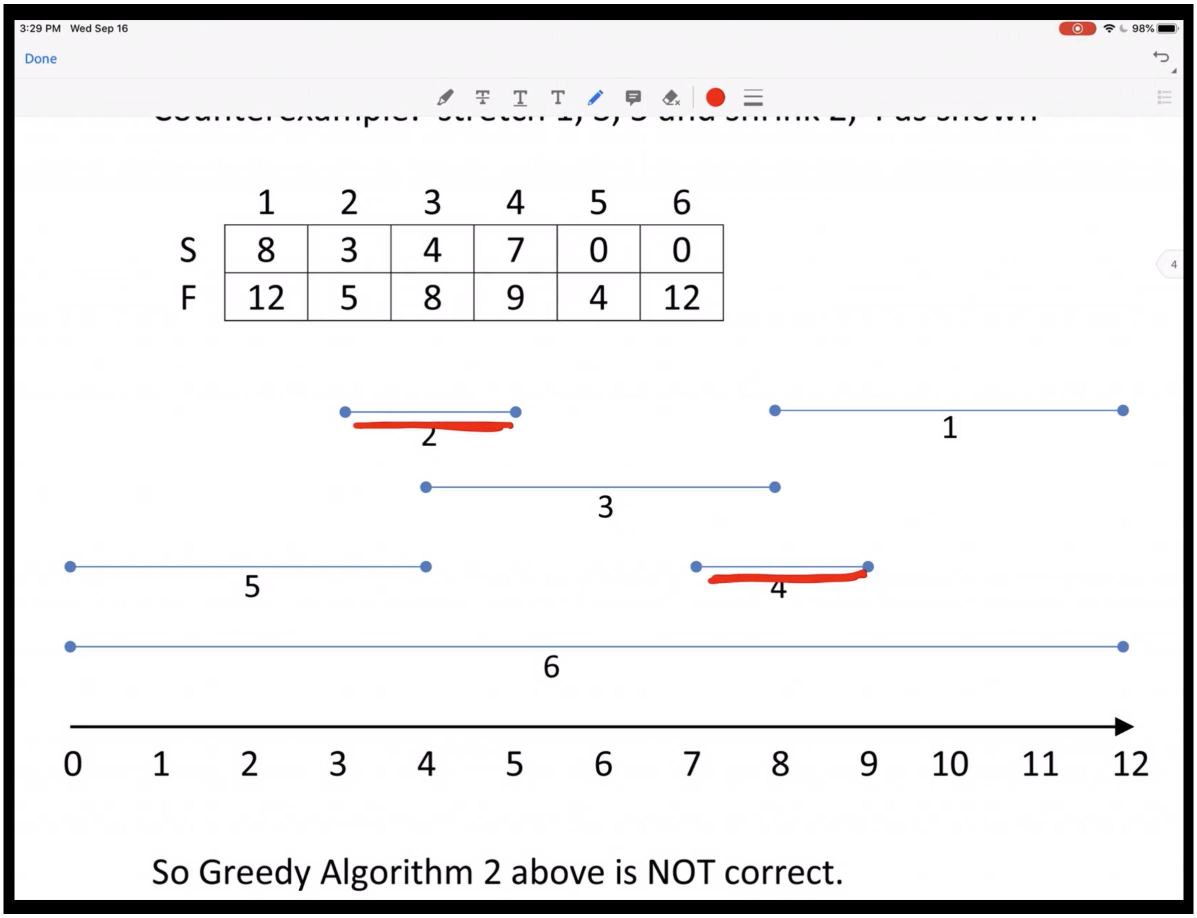
- However, there's a good counterexample to this problem. This wouldn't work right here (remember that the length of the job doesn't matter)



- Second greedy algorithm
  - Let's go over each item in order of *shortest job first* (F[k] - S[k]).
  - This would work for our little counterexample actually! It would select items, 3, 5, and 1.



- However, like the other approaches, there's a counterexample:



- So this greedy algorithm is not correct.

▼ Third greedy algorithm

- The third idea is to select items in order of their finish time. We want to choose the job with the earliest finish time first.
- This is the greedy strategy that works!
- We can't just say that 'there's no counter example lol'

▼ Contradiction argument proof

- Suppose this algorithm isn't correct
- Consider its first incorrect choice of some activity k.
- So there must be a better next choice, say some activity k' which leads to a larger solution subset A'.

- By compatibility, every future k'' in A' has F[k'] ≤ S[k'']. Also F[k] ≤ S[k''].
- Therefore solution A'-{k'} union {k} is an equally good solution, thus contradicting that K' is a better choice than k.

- Therefore, this algorithm is indeed correct.

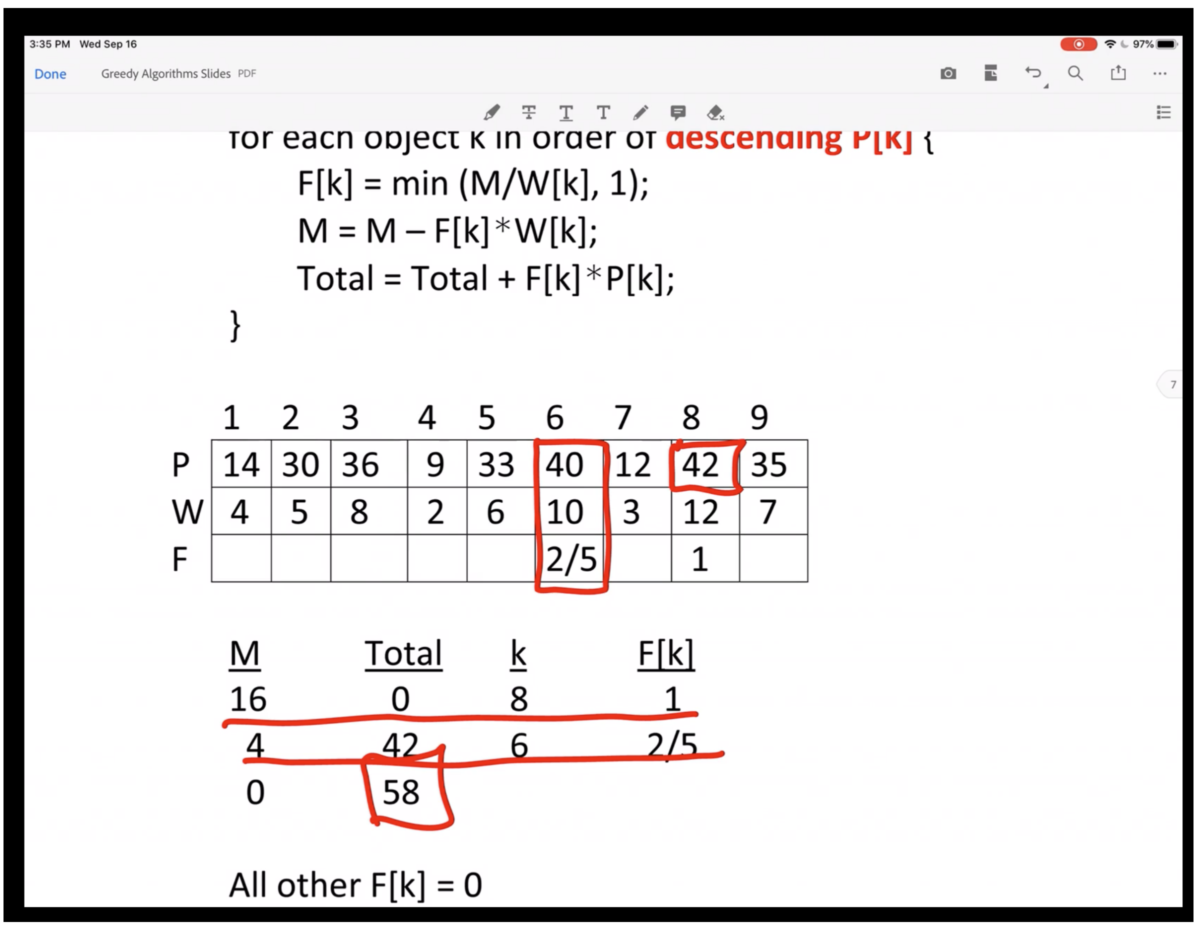
▼ What's the fractional knapsack problem?

- It's the 0-1 knapsack problem but without the 'we have to take the whole item' constraint.

▼ Algorithm one

- For each item in order of descending P[k], let's add each item as a whole and then add the last item as a fraction of the weight.

Counterexample:



- So this algorithm is not optimal. We put some of the heaviest items in first, which weren't the most profitable.

▼ Algorithm two

- Let's add items to the knapsack in order of increasing weight.
- This is not optimal either.

▼ Algorithm three

- Let's add things in order of descending P[k] / W[k] (highest profit to weight ratio).
- We're adding items in order of the most profit per unit of weight.
- This is actually the solution! We compute the value R (profit over weight value) and then just sort by that.