EECS 340, Breakout Session Notes, November 25, 2019

1. As usual, begin by handing out the quiz and collect it quiz at 3:40.

2. Place the students into groups and give them these problems: We have n items, and each item has a value. Item i has value v_i. Suppose we have two people and we want to divide the items up as follows:

   (a) Each person gets as close to the same amount of total value.

   (b) One person gets the $n/4$ of items that have the most value and the other gets $3n/4$ of items with least value.

   (c) No person ends up with more that twice the total value than the other person.

   (d) Each person gets to choose what items they want, and both people end up with the same number of desired items.

   Can they figure out the correct technique/algorithm for each? Do their answers change if instead of dividing between 2 people, we divide between m?

3. See if students can figure out the technique for each, figure out the actual algorithm for each, and have an idea how to prove each. Maybe you can ask the students to present their algorithm to the class.

4. **You will not have enough time to formally prove each, so don't go into those details.** I will provide the proofs below so that can understand the algorithms, and if students ask specific questions on how the proof might look, you can sketch it for them.

5. (a) is dymanic programming if no individual item has a lot of value. The recurrence is $T[i,k] = \text{true}$ if it is possible to get exactly k value from items 1 through i. The choices are we either include item i or we do not.

   $$T[i,k] = T[i-1,k] \vee T[i-1,k-v_i]$$

   Let $V = \sum_{i=1}^{n} v_i$ The solution is to look at $T[n,j]$ where j goes from $V/2$ to 1, and stop at the first value where $T[n.j]$ is true. The proof is straighforward because we either add the item in our group or we do not. The IH gives tells us whether, if we add the item, we can choose from the remaining items to get the needed total value. The running time is $O(nV)$. This is weakly polynomial so if V is not large compared to n it is fine.

   Otherwise, this problem is NP-hard via a reduction from Subset Sum. First we turn our problem into a decision problem: Can we divide amoung 2 people such that the difference in value between the two people receive is K'? Start with a Subset Sum problem: n values $v_1, \ldots, v_k$ and goal K. If $K < V/2$, we add a bogus item that has value $V - 2K$, and if $K > V/2$ add a bogus item with total value $2K - N$. Set $K' = 0$, and divide the items as evenly as possible, if we can divide them with $K' = 0$ difference between the two groups, and if $K < V/2$, then take the group that received the bogus item. the rest of its items sums to exactly K, as we solved Subset Sum. The same argument holds for when $K > N/2$.

   What happens if we have m people instead of 2? For the NP-hardness proof, nothing! Since we are now reducing to this problem, we can set $m = 2$ and show that with $m = 2$ is NP-hard, so it will still be NP-hard for arbitrary m. For the dynamic programming solution the recurrence gets a lot bigger: $T[i, k_1, \ldots, k_m] = \text{true}$ if it is possible to partition items $1, \ldots, i$ into m sets such that set j has value k_j, and

   $$T[i,k_1,\ldots,k_m] = T[i-1,k_1-v_i,k_2,\ldots,k_m] \vee T[i-1,k_1,k_2-v_i,\ldots,k_m] \vee \cdots \vee T[i-1,k_1,k_2,\ldots,k_m-v_i]$$

   So the runtime is now $O(nV^{m-1})$, and it is no longer a weakly polynomial algorithm.

6. (b) is Divide and Conquer. If the students come up with a sort, it is $O(n \log n)$, and we can do better. The hint might help them remember some divide and purge ideas. The algorithm set $K = n/4$ and try to find the K largest items. Take a random item, and sort the set of items into those with smaller value

and those with larger than or equal value. If the set with larger value has more than K items, repeat on this set again looking forthe K largest value items. If it has less than K, recurse on the set with lesser value, but now look for $K-$ (size of set with greater value) and add the result of the recursive call to the other set. The worse case time is $O(n^2)$ but the average case time is $T(n) = T(3n/4) + O(n)$ which is $O(n)$. The proof is again straightfoward because we assume the algorithm correctly finds the largest K values, for any $K < n$, on sets with fewer than n elements.

What happens if we have m people instead of 2? Where each person gets some fraction of items, on person gets a small number of the most valuable, the next gets a somewhat larger number of the next most valuable, and so on? We can just repeat this process and get $O(nm)$, or if $m > \log n$, just sort first and then divide so it is $O(n \log n)$.

7. (c) Is greedy, but the proof is trickier. Sort all the items from largest to smallest. Give the largest item to the first person, and the second person gets the next largest items until the second person has more value than the first. If the second person gets all the remaining items, and still has less value than the first person, we can do a quick check to see if the first person has more than twice the value than the second. If so, then a "fair" division is impossible. If not, we do a loop. Considering the remaining items from largest to smallest value, give the next item to the person with less total value.

   The standard greedy "swap" proof does not work well because the point is not to minimize the difference between the two people. We saw in (a) that this is NP-hard. So, instead we will use a loop invariant proof. The invariant is simply that after each iteration, no person has more than twice the value of the other. Before the first iteration, the second person has more value, but has at least 2 items. If the second person has more than twice the value, than one of these items must be worth more than the item the first person has. That is impossible. Now, assume we start the loop with no person having more than twice the value of the other. We give the next item to the person with less. If that person still has less, the invaraint holds. If that person has more than twice the first, then the item added must be worth more than any item the first person had, but that is impossible.

   The runtime is $O(n \log n)$ because we had to sort.

   What if we increase the number of people to m? Now we want the person with the most to have no more than twice the person with the least. The algorithm is roughly the same, but the division of the first items is a little trickier. We again go through the items from most to least, and give each person one item. Then we loop handing out items to the person with the least until the first person now has the least. Here we stop and see if the first person has more than twice the least. If so, then a fair division is impossible. Otherwise, we continue to give the person with the least value the next item, and the same loop invariant above holds.

8. (d) This is flow, but with only two people there is also a very simple greedy algorithm. For flow, have the source connect to each person with caoacity $k = \min\{d, n/2\}$ where d is the smallest number of items any one person chose. Have each person connect to an item they desire with capacity 1. Connect each item to a sink with capacity 1. Run flow. If we get a flow of $2k$ then each person sent one flow to k items they desire, and those are the items they get. If not? Then we reduce the capacity of each source to person vertex by 1 and now run flow to see if we can get $2(k-2)$. You can be more clever about reducing the capacities, but that does not significantly change the running time. This is $O(n^3)$ (number of edges is $O(n)$ and the max flow is $\leq n/2$, and we may need to run the flow algorithm up to $n/2$ times).

   Here is the simple greedy algorithm. Give the person who wants the least all the items they want. Call that person, person 1. Give person 2 all the items left over that they want (up to the number person 1 has). Until the two people have the same number of items, take an item from person 1 that person 2 wants and give it to person 2. This is $O(n)$.

   What happens when we have m people? The greedy algorithm no longer works, but the flow algorithm can be easily adjusted by adding another person vertex and edge from source to that vertex with capacity k. Now, we see of the total flow is mk.