EECS 340, Breakout Session Notes, October 7, 2019

1. As usual, we start with the quiz. Please end the quiz at 3:40.

2. Place the students into groups and give them this problem:

   You have to divide up n items between you and your sibling. Each item i has two values. There is the value a_i that you give the item and the value b_i that your sibling gives the item. You sibling will be unhappy if he/she does not get items that total to at least B total value (as summed by their b_i values).

   You want an algorithm that will choose the items for yourself to maximize the sum of the a_i values of the items you are getting such that the sum of the b_i values for the items you leave for your sibling is at least B.

   We want to solve this problem by *reducing* it to another similar problem.

3. Let the groups take just a little time figuring out what problem is close to this one. You will probably have to give hints because we just started reductions and they will not remember all the problems from class.

4. The problem that is closest to this problem is Knapsack. The hint is that we choosing values to take and we have a limit imposed on how many items we can choose. You should give them the formal definition of Knapsack since most students will have forgotten the details:
   We have n items. Each item has a weight w_i and a value v_i, and you have a bag with capacity C. You want to maximize the value of the items you take such that the sum of their weight is at most C.

5. Now, you should help them create the reduction. The key points are that we have to take the n items, values a_i and b_i and turn it into n items with value v_i weight w_i and a bag with capacity C.

6. The easier part is to see that the n items for the Knapsack problem will be the same n items for the sibling problem. The next easiest part is to see that for each item, we can set the value $v_i = a_i$ because that is what we want to maximize, and we can set the weight $w_i = b_i$ since that is what we want to limit. The trickiest part is to get the capacity correct. If we want to leave total b value (or w weight) B behind, then we want to take at most $\sum_{i=1}^{n} b_i - B$ value. So we set $C = \sum_{i=1}^{n} b_i - B$.

7. This reduction is clearly worst case linear time since we just run through the objects (actually we don't have to change the objects at all) and do a single summation of n values. Even if we allow the values to be really big, the summation is still linear in terms of the number of bits in the values.

8. Now we have to do an "if-and-only-if" proof that the reduction is correct. Let the groups try to form the proof statement before you do.

9. We need to prove:
   The sibling problem has a solution that gives me total value at least X if and only if there is a solution to the knapsack problem with total value at least X.
   You can remind them that this implies that a maximizing the sibling problem is the same as maximizing the knapsack, but we write it this way because it is easier to prove for a specific value than to prove the value is the maximum.

10. Proof: Assume the knapsack problem returns total value X. Then it selected items that sum up to X, and we choose the same items from the sibling problem to get value X. The items weigh at most $C = \sum_{i=1}^{n} b_i - B$ so we leave our sibling at least $\sum_{i=1}^{n} b_i - C = B$ value.

    Assume there is a way to choose total value X items for you while leaving your sibling at least B value. Fill a knapsack with those same items and you get value X in your knapsack. The sibling value of the items you take is at most $\sum_{i=1}^{n} b_i - B$. Since this value is also C, the values you take fit in the knapsack.