Time Spent: 1 hour

Collaborators and Resources:

Problem 2

For this problem, I decided that divide-and-conquer methods would be impractical. We are given two distinct sets R and S, one of which is a subset of the other. That is, R is a subset of S and we want to find the complement of S in R. That is, we want to be able to find the elements of the superset R which are not in S, and if there don't exist any we want to return "none" (that is, whenever R = S). The elements in the set are unordered and ordering them might take O(n·log(n)) time and yet without ordering the sets using divide-and-conquer becomes quite complicated. Thus I elected to do the simpler approach, which just involves concatenating the sets together and then determining which elements if any could have only come from one of the sets (which would then have to be R) ...

1. Given two unsorted arrays S and R such that $R \subseteq S$
2. Set M = R + S, that is, concatenate R and S into a single merged array (without sorting, so this should take O(1) time).
3. Numericize all the elements of M using a bijective function
4. Initialize an array A of arbitrarily large size consisting of all 0s
5. For i in (1:len(M)): This should take O(n) time
   (a) Add 1 to A[M[i]]
6. Iterate through the array A and find elements e which have a value of 1.
7. Return A[e] for all e. If e does not exist, return "none"

Claim 1. This algorithm runs in O(n) time and also finds elements, if they exist, which are in the set S but not in its subset R.

Proof. We can first claim that the algorithm runs in O(n) time because of the fact that all the operations except for the for-loop and iterating through A are done in linear time. The for-loop of course is going to run n times, where n is the sum of the sizes of R and S. Iterating through A likewise is going to be in O(n) time since there can only be n unique entries maximum (that is, if the subset R is the empty set then the union $R \cup S$ with cardinality n is just going to have all distinct elements).

Furthermore, the algorithm works because of the fact that the elements in R are mutually distinct as are the elements in S. When we merge them, we're going to get two copies of each element which is contained in the intersection $R \cap S$. The only elements which are going to have less than two copies (that is, be unique in the merged array M == R + S) are going to be the elements which are in exactly one of the sets R or S. Since everything in R is in S by definition of subsets, we know that elements in M are going to be distinct if and only if they are in the set $\{x \mid x \in S \setminus R\}$. So the algorithm is going to find all elements which are in the superset S but not in R, and if there are no such elements e then it will just return "none". □