**C-7.4**

We can inserting our objects into a hashmap with object as key and the number of objects as value, with some loop like “for each object do (if value == null then value = 0; value++ put(key, value))” we can get a map with unique keys and value as counts. Then it only takes O(n) time to get a unique set.

**A-7.2**

We would insert the sequence σ with n elements to disjoint trees, then we mark any elements in between two set of continuous extractions by Set B. Now for the n elements we create index i between 1 and n and j is index in range of Set B, where Set B is within range of 1 to n. To do the extractions we loop i from 1 to n and when i is within B , set i = j, now check if j is the last element in B. If so ignore j, else give value of j to Extraction array. After the extraction we remove the extracted J and one extraction function then union the Set B with size -1 with the rest elements. We keep doing this under recursive function and stop when finished all m extraction calls. According to theorem A.15 in ppt page 22, the running time for all nodes is . Thus the total time for m union-find operations with n singleton sets would be O((n+m)α(n)).

**C-8.3**

We can merge the sorted sequences A and B to C. Then traverse C and if the next element is equal to the current one, then we remove the next element. Since C is also sorted this should be done in O(n) with one scan through.

**C-8.7**

Sort the sequence S for O(n log n). Then like what I did in C-8.3 we can just delete the next element if it’s equal to the current one for O(n) time. Overall this is O(n log n).

Since we are give the number of elements in S we can also create a hashmap with length of n. Then like in C-7.4 we can do “for each object do (if value == null then value = 1 else return true)”. This should take O(n).

**A-8.4**

We could use the in-place, randomized quick-sort algorithm. We pick a nut from A then split B into (B1 < Ai)(Bk = Ai)(Bn>Ai). Then recursively sort B1and Bn until all matched. This Quick-sort should take O(n log n).

**C-9.5**

Since the range is from 1 to n^3, which means the elements could be triple tuples. Thus, we should use radix-sort which runs in O(n) time for this case. Then we scan through the sequence to return true if next element equals to current one, this should take another O(n). As a result, the total running time is O(n).

**A-9.5**

Since it’s randomly ordered we can use quick-sort for rank n/3 and 2n/3. Then we shall know the number of elements at that two ranks then we can know which two or which one student get more than n/3 votes. The quick sort should take O(n) time for this, and we only need to scan through the list once.