(1)
a) Describe the algorithms basic idea.

First, two hash sets are created. One to collect all minimal unique column combinations, and one for non-umique column combinations of size n-1 (last pass). As an initialization step, all single columns are checked for uinqueness. If they are unique, they are added to the result. If not, they are added to the last pass to mark that they need to be combined with other columns to be tested for uniqueness.
For the next part of the agorithm, the lastpass hashset contains all column combinations that are not unique on their own but could become so if combined with other columns.
Now, we iterate over every column combination in lastpass and and every single column, in order to test whether they become a unique column combination if combined. If the column is already contained in the combination, no new combination is formed so it is skipped. However, in every pass, a combination can be formed multiple times because the order of the columns does not make a difference. Therefore, a check whether this new combination has been tested already, is necessary. In order to do that, we keep a list of new combinations that have been generated in the current pass of checking UCCs of size n.
In order to only collect minimal unique column combinations, we have to test whether a known unique column combination is a subset of the combination to be tested. If so, this combination can not be minimal anymore and is therefore skipped.
Only if a combination passes these test, we perform a test on the data to determine the uniqueness. If the combination is in fact unique, it must be in fact a minimal unique column combination and is therefore added to the output.
In order to test the uniqueness of a column combination, the data tuples are inserted into a hashset. We iterate through every subrow in the data and if the subrow is already contained in the hashset, we know that the values are present at lest twice in the data so that the column combination can no be unique. If no duplicates are found, the combination is unique.

This approuch works well, if the minimal UCCs are small. The memory consumption to store all tuples of size n with large n and many columns is very high.

b) If you used an algorithm from literature, provide a reference to the according publication.

We implemented an apriori UCC algorithms with a different combination generation than in the slides in the lecture.

c) Provide one or two arguments why it is or could be better than related algorithms.

The algorithm uses pruning to reduce the number of checked combinations so it is better than a naive brute force approach. However, worse than an apriori UCC algorithm with prefix-based column combination generation

d) If your algorithm implements an adption of optimization of existing approaches, describe these briefly.

If one specific value is null, it is not inserted into the hashset during the uniqueness test. Because of this, our approach is optimized for the null unequals null assumption. A value that is null will evaluate to unequal to every other value. Therefore, the uniqueness of the column only depends on the non-null-values. If null values are present, the hashset will have a smaller size.

(2)

a) How many unique column combinations did your algorithm find on the provided datasets?

Input data set	Planets	Ncvoter 1k	Ncvoter
UCC count	7	113	Crashed with OutOfMemoryError

b) How long did it take?

Used hardware: 2,6 Ghz Intel Core i7, 16GB DDR 3, Mac OS 10.12

Input data set	Planets	Ncvoter 1k	Ncvoter
Time	9ms	38s	Crashed after 11 min

c) Did you discover any limitations of your approach?

Our approach is limited by memory. The hashset that contains every possible column combination for a specific level gets very big very easily and therefore exceeds the available memory on big datasets.
The execution on the large Ncvoter data set failed with "java.lang.OutOfMemoryError: Java heap space".

d) What is the conecptual difference between NULL != NULL and NULL = NULL and how does the choice influence the performance of your algorithm.

NULL != NULL means that if null values are detected in a column, the column can still be found to be unique because the uniqueness only depends on the non null values. Therefore, if NULL = NULL is assumed and there are at least two null values found and the non NULL values are not the same, we could instantly stop the test as this column combination is non unique. Because of this, NULL != NULL semantics produce smaller UCCs.
Therfore, the algorithm performs best, if it does not have to check large column combinations as is the case with NULL != NULL semantics.
We implemented the NULL = NULL sematic and the results on larger data sets clearly show the worsened performance.

Input data set	Planets	Ncvoter 1k	Ncvoter
Time	10ms	381s (6:21 min)	Crashed after 11 min