Question 1

Let's say that $A$ and $B$ are the two databases and $A(i)$ and $B(i)$ are the $i^{th}$ smallest elements of $A$ and $B$ respectively.

If we let $k = \left\lceil \frac{n}{2} \right\rceil$ then $A(k)$ and $B(k)$ are the medians of the two databases. Suppose $A(k) < B(k)$ (the other case will be symmetric, and equality can not happen because we are assuming all values are distinct). Then $B(k)$ is greater than the first $k$ elements of $A$. Furthermore, $B(k)$ is also greater than the first $k - 1$ elements of $B$. Thus $B(k)$ is at least the $2k^{th}$ element in the combined database. Since $2k \geq n$, all elements greater than $B(k)$ are greater than the median, and therefore we can eliminate the second part of $B$ from consideration. Let $B'$ be the first $k$ elements of $B$.

Similarly, we can argue that $A(k)$ is smaller than the last $\left\lceil \frac{n}{2} \right\rceil$ elements of $A$, as well as the last $\left\lceil \frac{n}{2} \right\rceil + 1$ elements of $B$. Thus we can eliminate the first half of $A$. More precisely, we can define $A'$ to be $A$ from element $\left\lceil \frac{n}{2} \right\rceil + 1$ to $n$.

Since we have eliminated $\left\lceil \frac{n}{2} \right\rceil$ elements less than the median and $\left\lceil \frac{n}{2} \right\rceil$ elements greater than the median, the median of our new problem is the same as the median of our old problem. We can now recursively find the median of $A'$ and $B'$. Note that we can’t actually delete the elements from our database, and even if we could, this would be too time consuming. However, the $i^{th}$ smallest element of $A'$ is the $i + \left\lceil \frac{n}{2} \right\rceil^{th}$ smallest element of $A$, and the $i^{th}$ smallest element of $B'$ is the $i^{th}$ smallest element of $B$.

Formally, we define the algorithm median$(m, a, b)$ which takes integers $m$, $a$ and $b$ and finds the median of the union of the two segments $A[a+1 : a+m]$ and $B[b+1 : b+m]$.

\[
\text{median}(m, a, b) \\
\quad \text{if } m = 1 \text{ then return min}(A(a + 1), B(b + 1)) // \text{ base case} \\
\quad k = \left\lceil \frac{m}{2} \right\rceil \\
\quad \text{if } A(a + k) < B(b + k) \text{ then return median}(k, a + \left\lceil \frac{m}{2} \right\rceil, b) \\
\quad \text{else return median}(k, a, b + \left\lceil \frac{m}{2} \right\rceil) 
\]
To find the median in the whole set we evaluate \( \text{median}(n, 0, 0) \). If we define \( Q(n) \) to be the number of queries asked by our algorithm on 2 databases of size \( n \) we have that \( Q(n) = Q(\lceil \frac{n}{2} \rceil) + 2 \). Therefore \( Q(n) \) is \( O(\log n) \).

Question 2

Let \( e_1, \ldots, e_n \) denote the equivalence classes of the cards: cards \( i \) and \( j \) are equivalent if \( e_i = e_j \). We are looking for a value \( x \) so that more \( n/2 \) of the indices have \( e_i = x \).

Divide the set of cards into two roughly equal piles: a set of \( \lceil n/2 \rceil \) cards and a second set of the remaining \( \lfloor n/2 \rfloor \) cards. We will recursively run the algorithm on the two sides, and we will assume that if the algorithm finds and equivalence class containing more than half of the cards, then it returns a sample card in that equivalence class.

Suppose that more than half of the cards are equivalent, for example more than half are equivalent to \( x \). Then at least one of the two halves must also have at least half of its elements being equivalent to \( x \) (and possibly both halves). So at least one of our two recursive calls will return a card equivalent to \( x \). Of course, the reverse is not true: just because more than half the elements on one side are equivalent to \( x \) does not mean that more than half of the total elements among both sides are equivalent to \( x \). Therefore if we get a majority card returned for either side, we must check it (or them, as we may get two different cards). The following algorithm is the result.

**Majority(\( S \))**

- if \( |S| = 1 \) return the card in \( S \)
- let \( S_1 \) be the first \( \lfloor n/2 \rfloor \) cards
- let \( S_2 \) be the remaining \( \lceil n/2 \rfloor \) cards
- call Majority(\( S_1 \))
- if this returns a card \( a \) test this against all other cards
- if \( a \) is a strict majority, return \( a \)
- call Majority(\( S_2 \))
- if this returns a card \( b \) test this against all other cards
- if \( b \) is a strict majority, return \( b \)
- return no strict majority found

The correctness of this algorithm follows from the observation that if there is a majority equivalence class, then this must be a majority equivalence class for at least one of the two sides. The running time analysis is similar to Mergesort and our Nearest Pair of Points algorithms: If we let \( T(n) \) be our running time with \( n \) cards, then \( T(n) = 2T(n/2) + 2n \). As we have seen, this implies that \( T(n) \) is \( O(n \log n) \).