We give two solutions for this problem. The first solution is a divide and conquer algorithm, which is easier to think of. The second solution is a clever linear time algorithm.

Via divide and conquer: Let $e_1, \ldots, e_n$ denote the equivalence classes of the cards: cards i and j are equivalent if $e_i = e_j$. What we are looking for is a value x so that more than $n/2$ of the indices have $e_i = x$.

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

Note that if there are more than $n/2$ cards that are equivalent in the whole set, say have equivalence class x, than at least one of the two sides will have more than half the cards also equivalent to x. So at least one of the two recursive calls will return a card that has equivalence class x.

The reverse of this statement is not true: there can be a majority of equivalent cards in one side, without that equivalence class having more than $n/2$ cards overall (as it was only a majority on one side). So if a majority card is returned on either side we must test this card against all other cards.

```
If |S| = 1 return the one card
If |S| = 2
    test if the two cards are equivalent
    return either card if they are equivalent
Let S_1 be the set of the first ⌊n/2⌋ cards
Let S_2 be the set of the remaining cards
Call the algorithm recursively for S_1.
If a card is returned
    then test this against all other cards
If no card with majority equivalence has yet been found
    then call the algorithm recursively for S_2.
    If a card is returned
        then test this against all other cards
Return a card from the majority equivalence class if one is found
```

The correctness of the algorithm follows from the observation above: that if there is a majority equivalence class, than this must be a majority equivalence class for at least one of the two sides.

To analyze the running time, let $T(n)$ denote the maximum number of tests the algorithm does for any set of n cards. The algorithm has two recursive calls, and does at most $2n$ tests outside of the recursive calls. So we get the following recurrence (assuming n is divisible by 2):

$$T(n) \leq 2T(n/2) + 2n.$$

[1] ex628.974.324

As we have seen in the chapter, this recurrence implies that $T(n) = O(n \log n)$.

In linear time: Pair up all cards, and test all pairs for equivalence. If n was odd, one card is unmatched. For each pair that is not equivalent, discard both cards. For pairs that are equivalent, keep one of the two. Keep also the unmatched card, if n is odd. We can call this subroutine ELIMINATE.

The observation that leads to the linear time algorithm is as follows. If there is an equivalence class with more then $n/2$ cards, than the same equivalence class must also have more than half of the cards after calling ELIMINATE. This is true, as when we discard both cards in a pair, then at most one of them can be from the majority equivalence class. One call to ELIMINATE on a set of n cards takes $n/2$ tests, and as a result, we have only $\leq \lceil n/2 \rceil$ cards left. When we are down to a single card, then its equivalence is the only candidate for having a majority. We test this card against all others to check if its equivalence class has more than $n/2$ elements.

This method takes $n/2 + n/4 + \ldots$ tests for all the eliminates, plus $n - 1$ tests for the final counting, for a total of less than $2n$ tests.