CS 4820, Spring 2018 **Homework 5, Problem 1**

Name: Rongguang Wang
NetID: rw564
Collaborators: Siyuao Liu (sl2928); Yihao Chen (yc2288)

(1) *(Note: This problem is Exercise 5.3 in Kleinberg & Tardos, with an extra "part b" appended.)* Suppose you're consulting for a bank that's concerned about fraud detection, and they come to you with the following problem. They have a collection of n bank cards that they've confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic stripe with some encrypted data, and it corresponds to a unique account in the bank. Each account can have many bank cards corresponding to it, and we'll say that two bank cards are *equivalent* if they correspond to the same account.

It's very difficult to read the account number off a bank card directly, but the bank has a high-tech "equivalence tester" that takes two bank cards and, after performing some computations, determines whether they are equivalent.

Their question is the following: among the collection of n cards, is there a set of more than $n/2$ of them that are all equivalent to one another? Assume that the only feasible operations you can do with the cards are to pick two of them and plug them into the equivalence tester.

(a) *(5 points)* Show how to decide the answer to their question with only $O(n \log n)$ invocations of the equivalence tester.

Solution:

Assume that there are n cards $e_1, \ldots, e_n$ and two cards i and j are treated as equivalent if $e_i = e_j$. The question is looking for a value x so that more than $n/2$ of the cards have $e_i = x$ and the time bound for finding the cards should be $O(n \log n)$. A subroutine was provided to test whether two cards are equivalent.

According to Divide and Conquer Algorithm, the sets of cards S should be divided into two nearly equal subsets: the first half of $n/2$ cards S_1 and the second half of $n/2$ cards S_2. The algorithm should run recursively on the two subsets. If a card is find such that more than half of the cards are equivalent to it, the algorithm will return the card. According to the observation, if there are more than half cards are equivalent in the card set S, one of the two subsets (S_1 and S_2) should have more than half cards that have the same value. Therefore, at least one subsets (S_1 or S_2) will return a card which has more than $n/2$ equivalent cards in the subset. Additionally, there may be a majority of equivalent cards in one subsets. So, the card should be tested against all other cards in the set S if a majority card is returned.

By dividing the set, the card, to be returned, which have more than $n/2$ (half) equivalent cards must have more than $m/2$ (half) equivalent cards in any one of the divided subsets of size m. If a card was returned, all cards in the set S should be traversed to find whether there are more than half cards that are equivalent to this card. If so, the card was found. Otherwise, return null. The base case are sets that have one card and two cards. Since one card exceed the half of the subset number, it will be returned. This also implies that the number of cards in the set S is odd. For the two cards case, if the two cards are equivalent, either one of the card should be returned. Otherwise, return null.

Algorithm:

```
given cards set S,

function CardsTest (S):
if size(S)=1, return the card
if size(S)=2,
    if card 1 = card 2,
      return either card
    else
      return null

assign the first half of the cards to S1
assign the second half of the cards to S2

call function CardsTest (S) with input S1 (first half of set S)

if a card is returned, test the card against all other cards in set S
    if the card equivalent to more than half of the cards in set S,
        return the card

else
    call function CardsTest (S) with input S2 (second half of set S)
      if a card is returned, test the card against all other cards in set S
          if the card equivalent to more than half of the cards in set S,
                return the card

return null
```

Algorithm's Correctness:

Lemma 1. *If there is a majority equivalent card, this card must be a majority equivalent card for at least one of the two subsets.*

Proof. Assume that there is a card C that has more than $n/2$ equivalent cards in the set S which contains all cards provided that dose not returned in any base case sets. There are m sets and the average size of the set is 2, the number of the equivalent cards of card C is less than m. Since $m \times 2 = n$, which means the number is less than $n/2$ and thats contradict to the assumption. Therefore, the card C must at least have more than half equivalent cards in any subsets. □

Running Time Analysis:

Let $T(n)$ be the maximum number of tests performed for any set of the n cards. The algorithm has two recursive calls and each has size $n/2$. At most $2n$ tests may performed outside of the recursive call. Therefore, the recurrence is as below,

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

which is $T(n) = O(n \log n)$.

(b) *(5 points)* Now modify the question so that you must decide whether there is a set of more than $n/4$ of the cards that are all equivalent to one another. Show how to decide the answer to this question with only $O(n \log n)$ invocations of the equivalence tester.

If you are confident that you have a correct solution to part (b) of this problem, you can skip part (a) and your score on the problem will be computed by doubling your score for part (b).

Solution:

Assume that there are n cards $e_1, \ldots, e_n$ and two cards i and j are treated as equivalent if $e_i = e_j$. The question is looking for a value x so that more than $n/4$ of the cards have $e_i = x$ and the time bound for finding the cards should be $O(n \log n)$. A subroutine was provided to test whether two cards are equivalent. The main difference between this question and $1(a)$ is that there may be three cards that have equivalent value to more than $n/4$ cards, whereas there could be only one card for $1(a)$.

The algorithm can be developed based on the solution of $1(a)$. The sets of cards S should also be divided into two nearly equal subsets: the first half of $n/2$ cards S_1 and the second half of $n/2$ cards S_2. The algorithm should run recursively on the two subsets. If a card is find such that more than a quarter of the cards are equivalent to it, the algorithm will return the card. The optimal minimum subset becomes to four instead of two now, since the potential of over $n/4$ never fails in the subsets that contains one, two and three cards. According to the observation, if there are more than a quarter cards are equivalent in the card set S, one of the two subsets (S_1 and S_2) should have more than a quarter cards that have the same value. Therefore, at least one subsets (S_1 or S_2) will return a card which has more than $n/4$ equivalent cards in the subset. Additionally, there may be a majority of equivalent cards in one subsets. So, the card should be tested against all other cards in the set S if a majority card is returned.

By dividing the set, the cards, to be returned, which have more than $n/4$ (a quarter) equivalent cards must have more than $m/4$ (a quarter) equivalent cards in any one of the divided subsets of size m. If cards were returned, all cards in the set S should be traversed to find whether there are more than a quarter cards that are equivalent to this card. If so, the card was found. Otherwise, return null. The base case are sets that have one card, two cards, three cards and four cards. Since one card, two cards and three cards all exceed a quarter of the subset number, the cards will all be returned. For the four cards case, if there are two cards or more are equivalent, the card should be returned. Otherwise, return null.

Algorithm:

given cards set S,

```
function CardsTest (S):
if size(S)= 1, 2, or 3, return the card
if size(S)= 4,
    if 2 cards or more are equivalent,
        return two cards
    else
        return null

assign the first half of the cards to S1
```

```
assign the second half of the cards to S2

call function CardsTest (S) with input S1 (first half of set S)

if a card is returned, test the card against all other cards in set S
      if the card equivalent to more than a quarter of the cards in set S,
            return the card

else
    call function CardsTest (S) with input S2 (second half of set S)
      if a card is returned, test the card against all other cards in set S
          if the card equivalent to more than a quarter of the cards in set S,
                return the card

return null
```

Algorithm's Correctness:

Lemma 2. *If there is a majority equivalent card, this card must be a majority equivalent card for at least one of the two subsets.*

Proof. Assume that there is a card C that has more than $n/4$ equivalent cards in the set S which contains all cards provided that dose not returned in any base case sets. There are m sets and the average size of the set is 4, the number of the equivalent cards of card C is less than m. Since $m \times 4 = n$, which means the number is less than $n/4$ and thats contradict to the assumption. Therefore, the card C must at least have more than a quarter equivalent cards in any subsets. □

Running Time Analysis:

Let $T(n)$ be the maximum number of tests performed for any set of the n cards. The algorithm has two recursive calls and each has size $n/2$. At most $6n$ tests may performed outside of the recursive call. Therefore, the recurrence is as below,

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

which is $T(n) = O(n \log n)$.