```
CARDS(A,p,r)
if p < r
    then q ← ⌊(p + r)/2⌋
        left = CARDS(A,p,q)
        right = CARDS(A,q+1,r)
        if CHECKER(left, right) #These 2 cards are of the same bank account
            then return left
        else
            result = TEST(A,p,r)
            return result
else
    return A[p]

TEST(A,p,r)
    q ← ⌈(p + r)/2⌉
    create array T[1…r] of zeros #There are at most r different accounts
    for i ← p to r
        do T[A[i]] += 1
    Max = 0
    Pos = 0
    tester = False
    for i in T
        do if T[i] > Max
            then Max = T[i]
                Pos = i
                tester = True
        elif T[i] == Max
            tester = False
    if tester == True and Max ≥ q
        then return Pos
    else
        return None
```

For this algorithm, it takes as input an Array A of cards, and for each recursive call, it divides the array by half. And this uses the divide and conquer strategy. It divides the array by half until there is only one element, and it would then return that one element. For the returned elements, it would check if the right and left are the same, if so, it would just return one of them as it's the most frequent element for this subarray. If not, then it would call TEST to traverse the array A from position p to r. And it would record the number of times each element happened to array T, and will use it to find the element that happened more than half of the times if it exists. Otherwise, the NONE would be returned as there's no element happened more than half times or 2 element that happened the most are of the same times, and the returned value could be used for the comparison later with another half of the subarray. And in the end, the algorithm would return NONE if not more than half the cards correspond to the same account, or the card otherwise.

Within the TEST function, it traverse the whole array in the worst case, and would take time Θ(n). For CARDS, it has time efficiency $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(\mathrm{n})$. And for convenience, we'll omit the floors, ceilings, and boundary conditions. Hence, $T(n) = 2T(n/2) + \Theta(\mathrm{n})$.

By using the master method, for this recurrence, we have a = 2, b = 2, f(n) = n, and $n^{\log_b a} = n^{\log_2 2} = n^1 = n$. Case 2 applies, since $f(n) = \Theta\left(n^{\log_b a}\right) = \Theta(\mathrm{n})$, and thus the solution to the recurrence is $T(n) = \Theta\left(n^{\log_b a}\, lg\, n\right) = \Theta(n\, lg\, n)$.