CS 577: HW 1

Daniel Ko

Summer 2020

§1 Among the collection of n cards, is there a set of more than n/2 of them that are all equivalent to one another?

§1.1 Present your algorithm and explain the notations as necessary.

```
Data: cards, a collection of n cards; originalCards, the original/unmodified collection of n
      cards
Result: A card such that more than n/2 are equivalent if it exists, else null.
def fraudMajority(cards):
    if len(cards) ≤ 1:
        return cards[0];

    left = fraudMajority(cards[0:floor(len(cards)/2)])
    right = fraudMajority(cards[floor(len(cards)/2):len(cards)])
    numOfLeft = originalCards.count(left)
    numOfRight = originalCards.count(right)
    if numOfLeft > len(originalCards)/2:
        return left
    elif numOfRight > len(originalCards)/2:
        return right
    else:
        return null
```

§1.2 Prove the correctness of your algorithm in making the final judgement regarding whether there exists a set of more than n/2 of cards that are all equivalent to one another.

Proof. We show by strong induction that our algorithm is correct. Let $P(n)$ be the predicate, "fraudMajority halts and returns a card in a collection of n cards such that more than $n/2$ are equivalent if it exists". We define $n \in \mathbb{N}$.

Base Case: When $n = 0$, fraudMajority halts and returns null which is the correct output because the collection is empty. Hence, $P(0)$ holds.

Inductive step: Suppose $P(n)$ holds for all $n \leq k \in \mathbb{N}$. We show that $P(k+1)$ holds.

We observe that fraudMajority recursively calls collections of sizes $\lfloor (k+1)/2 \rfloor$ and $(k+1) - \lfloor (k+1)/2 \rfloor$. Since $\lfloor (k+1)/2 \rfloor \leq k$ and $(k+1) - \lfloor (k+1)/2 \rfloor \leq k$, we know by our strong inductive hypothesis that these recursive calls must halt. Hence, fraudMajority halts with an input of $k+1$ cards.

fraudMajority finds the majority card of its left and right side, of size $\lfloor (k+1)/2 \rfloor$ and $(k+1) - \lfloor (k+1)/2 \rfloor$ respectively. Since both of these collections are less than k, by our strong inductive hypothesis, these recursive calls must return the correct output, the majority card of that partition if it exists. Then our algorithm calculates the total number of the majority card on the right and left on our $k+1$ collection and returns the majority. If no majority exists, it returns null. Thus, fraudMajority returns the correct output for a collection of $k+1$ cards.

By strong induction, we have proven that $P(n)$ is true for all $n \in \mathbb{N}$. Therefore, fraudMajority is correct. □

§1.3 Show the recurrence of the computing complexity and explain how did you obtain it. Note that you do not have to solve the recurrence since we can derive the solution using the Master's theorem.

We claim that $T(n) = 2T(n/2) + 2n$ which is $\Theta(n \log n)$.

Proof. Each call to our algorithm will result in two recursive calls with its input sized being halfed, which accounts for $2T(n/2)$. Additionally, calculating the number of originalCards will require n time and we do this computation for both the left and right card, hence requiring an overall $2n$ time. The rest of the computations take constant time. Using the master theorem, it is clear that we get values of $a = 2, b = 2, d = 1$ for this particular recurrence which results in a complexity of $\Theta(n \log n)$. □

§2 Please describe an algorithm that finds a number that is bigger than any of its neighbors.

For brevity, we define a peak to be a number that is bigger than any of its neighbors.

§2.1 Present your algorithm and explain the notations as necessary.

```
Data: arr, the nxn matrix representing the grid of boxes.
Result: A peak.
def findLarge(arr):
    max ← max element of boundary elements, middle row, and middle column
    # Comparison is vacuously true for the neighbors that do not exist
    if max > max's neighbors:
        return max
    # New submatrix does not contain the rows/columns used to find max
    recursive_arr ← the submatrix of the quadrant containing the largest neighbor
    return findLarge(recursive_arr)
```

§2.2 Prove the correctness of your algorithm in finding the right box that has a number that is larger than all its neighbors.

Proof. We show by strong induction that our algorithm is correct. Let $P(n)$ be the predicate, "findLarge halts and returns a peak in a nxn matrix". We define $n \in \mathbb{N}^+$.

Base Case: When $n = 1$, findLarge halts the only element in the matrix, which by definition is larger than its neighbors, a peak. Hence, $P(1)$ holds.

Theorem

A peak exists in any given matrix where its elements are unique integers.

Proof. Let the set containing all the unique integers in the matrix be $S = \{a_1, \cdots, a_\alpha\}$. Take set $S' = \{b_1, b_2, \cdots\}$ to be the upper bound of S, where the upper bound is defined as $\forall a_i \in S \;\; \forall b_j \in S', a_i \leq b_j$. Using the well ordering principle, there must exist as smallest element $b \in S'$. By definition, this must be the greatest element $a \in S$. Hence, there exist a largest integer in a finite set of unique integers. By definition, this largest integer is a peak. □

Corollary

There exists a peak in the submatrix of the quadrant containing the largest neighbor.

Proof. By previous theorem, there must exist a peak in the submatrix of the quadrant containing the largest neighbor because any subset of the original matrix containing unique elements, will have unique elements. Consider the special case where the peak is on the edge of the new submatrix. Let α be the largest neighbor in the original matrix and γ be the max and peak on the edge. We know that $\gamma > \alpha$ because when we fix a max, by definition it is the greatest integer in the boundary elements, middle row, and middle column, which includes α. This directly implies that γ is greater than all its neighbors within the original matrix because of the fact that α was the greatest integer of the values that directly border the submatrix. □

Inductive step: Suppose $P(n)$ holds for all $n \leq k \in \mathbb{N}^+$. We show that $P(k+1)$ holds.

We observe that findLarge recursively calls with an matrix of size $\lfloor (k-1)/2 \rfloor \, x \, \lfloor (k-1)/2 \rfloor$. Since $\lfloor (k-1)/2 \rfloor \leq k$, we know by our strong inductive hypothesis that this recursive call must halt. Hence, findLarge halts with a matrix of size $(k+1)x(k+1)$.

If a peak does not exist in the boundary elements, middle row, and middle column, findLarge recursively calls the submatrix with the biggest neighbor. By the corollary above, there exist a peak in this submatrix. Since the size of the submatrix is $\lfloor (k-1)/2 \rfloor \, x \, \lfloor (k-1)/2 \rfloor$, and $\lfloor (k-1)/2 \rfloor < k$, by our strong inductive hypothesis this recursive call returns a valid peak.

By strong induction, we have proven that $P(n)$ is true for all $n \in \mathbb{N}^+$. Therefore, findLarge is correct. □

§2.3 Analyze the number of boxes that one need to open using your algorithm.

We claim that $T(n^2) = T((n/2)^2) + \Theta(6(n/2))$ which is $\Theta(n)$.

Proof. Suppose we have a size nxn matrix. Each recursive call will have a input size being one quarter of original matrix, which accounts for $T((n/2)^2)$. Additionally, when searching for the max element, we must search the boundary elements, middle row, and middle column, which would require, 6 times half the new input size i.e. $\Theta(6(n/2))$. The rest of the computations take constant time. Let $m = n^2$

$$T(n^2) = T((n/2)^2) + \Theta(6(n/2))$$
$$T(m) = T(m/4) + \Theta(3\sqrt{m})$$

Using the master theorem, it is clear that we get values of $a = 1, b = 4, d = \frac{1}{2}$ for this particular recurrence which results in a complexity of $\Theta(\sqrt{m}) = \Theta(n)$. □