Page 3: Divide and Conquer Algorithm Design

3. [15 points] **Event Planning.** You are organizing an event this weekend and have sent invitations to n guests. You maintain two lists. The first list, $invites$, contains the names of all invited guests, sorted alphabetically by their last name. The second list, $rsvps$, contains the names of guests who have responded (i.e., not sorted by last name). Today, you see that exactly $n-1$ guests have responded. Give a **divide and conquer** algorithm that identifies the remaining guest who has not yet responded in $O(n)$ string comparisons. You can assume that all guests have unique last names. *Hint: think about how to Partition.*

Take the median name x from `invites`. We partition `rsvps` using x as the pivot value. Namely, let `left` be the subset of names in `rsvps` less than x and `right` be the subset of names greater than x. If x is not found in `rsvps`, return x. Count the number of names in `left` and `right` and compare to the corresponding number in `invites`. Recurse on the corresponding subset of `invites` that is missing a guest (and the corresponding subset of `left`/`right`). Since we pivot around the median element, we rule out $n/2$ guests each iteration. The running time of this algorithm is $T(n) = T(n/2)+\Theta(n)$, which by Case (1) of Master's Theorem, satisfies $T(n) \in \Theta(n)$.

Page 4: Divide and Conquer, cont.

4. [5 points] Argue why the "Event Planning" problem from the previous page has a linear worst-case lower bound. Namely, show that no algorithm can find the missing guest in $o(n)$ time.

   If there is any guest in `rsvps` that the algorithm does not look at, then there are two possible guests from `invites` that are still unaccounted for, and the algorithm has no way to distinguish these two cases.

5. [6 points] For each of the properties below, give an example of a divide and conquer algorithm from lecture or the homework that fulfills it. [3 points each]

   a) A constant-time divide step and a $\Theta(n)$ combine step.

      Merge Sort, Closest Pair

   b) A $\Theta(n)$ divide step and constant-time combine step.

      Quicksort, QuickSelect, Conference Superstar

6. [4 points] The following two questions ask why **Karatsuba's** algorithm and **Strassen's** are more efficient than the naive divide-and-conquer solutions to the problems they solve. For each of the following questions, circle **Karatsuba's**, **Strassen's**, both or neither.

   Which is more efficient because it solves fewer subproblems? **Karatsuba's** **Strassen's**
   Both

   Which is more efficient because it solves smaller subproblems? **Karatsuba's** **Strassen's**
   Neither