CS:5350 Homework 3, Spring 2016

Due in class on Thu, Feb 11

Collaboration: You are welcome to form groups of size 2 and work on your homeworks in groups. Of course, you are not required to work in groups. Every group should make one submission and names of both group members should appear on the submission and both students in a group will receive the same score. Other than the TA and the professor, you can only discuss homework problems with your group partner. Collaboration can be positive because talking to someone else about these problems can help to clarify your ideas and you will also (hopefully) get to hear about different ways of thinking about the problem. On the other hand, collaboration can be negative if one member of the group rides on work being done by the other member – please avoid this situation. If your solutions are (even partly) based on material other than what has been posted on the course website, you should clearly acknowledge your outside sources.
Late submissions: No late submissions are permitted. You will receive no points for your submission if your submission is not turned in at the beginning of class on the due date.
Evaluation: Your submissions will be evaluated on correctness *and* clarity. Correctness is of course crucial, but how clearly you communicate your ideas is also quite important.

1. Given a sequence $A = (a_1, a_2, \ldots, a_n)$ and an element t, let us now consider the following procedure. Here t is either an element in A or is `null`.

```
pairupAndRemove(A, t)
    Pair up elements of A arbitrarily, to get ⌊n/2⌋ pairs
    Note: If n is odd, one element in A is not in any pair – call this t'
    Let A' be the empty sequence
    for each pair (x, y) do
        if x = y
            add x to A'
    if n is odd
        return (A', t')
    else
        return (A', t)
```

   (a) Just so that you understand what the above procedure does, suppose that $A = (1, 0, 1, 1, 0, 1, 0)$ and $t =$ `null`. Write down *all possible* outputs when the above procedure is called with arguments A and t. Note that the step in the algorithm that pairs up elements is under specified, i.e., you can pair up elements in different ways and these differing choices could lead to different outputs. For this problem you are being asked to consider all possible pairings that the procedure could perform and write down all possible outputs this could lead to.

   (b) A sequence $A = (a_1, a_2, \ldots, a_n)$ is said to have a *majority element* x if *more than half* the entries are identical and equal to x. We now extend this definition to a pair (A, t) as follows. Give each element in A one vote and give the element t one-tenth of a vote. Count the number of votes received by each element and if an element in A receives more than $n/2$ votes then it is the majority; otherwise there is no majority. For example, if $A = (5, 4, 5, 4)$ and $t = 5$ then the element 5 receives 2.1 votes and 4 receives 2 votes and 5 is the majority element of the pair (A, t). From this example, it should be clear that t serves a sort of a "tie breaker," but with less than one vote. Now consider the following claim:

> **Claim:** Suppose that $n > 0$ and the pair (A, t) has a majority element x then the pair returned by `pairupAndRemove`(A, t) to also has x as the majority element.

Prove this claim for odd n. (The claim is true for even n also, but you don't have to prove it for even n.) To get started let m be the number of instances of x in A and let u be the number of instances of all elements in A that are different from x. Let $e = m - u$ denote the "excess" number of instances of x in A. Think about possible values of e and what could happen to e as a result of the `pairupAndRemove` procedure.

(c) Now consider the following problem:

> MAJORITY
> INPUT: A sequence $A = (a_1, a_2, \ldots, a_n)$.
> OUTPUT: The majority element in A, if A has one; otherwise, a message indicating that A does not have a majority element.

Use the claim in part (b) to design an $O(n)$-time divide-and-conquer algorithm for the MAJORITY problem. Make sure you think about (i) how the algorithm is first called, (ii) the bases case(s) and (iii) about the situation in which there is no majority. Describe your algorithm clearly in pseudocode.
Note: This seems to be a popular interview question! As a result, there is a lot of discussion about this problem on the internet. Feel free to use internet resources, but make sure your answer is in your own language and that you correctly acknowledge any resources you use.

(d) Write down the recurrence relation for the running time of your algorithm. (You don't have to solve it.)

2. Study Section 1.8 from Jeff Erickson's notes on Karastuba's algorithm for fast multiplication. Then answer Problem 28(e) (on Page 26) in Chapter 1 from these notes.

3. On Slide 10 in the posted notes on the *Closest Point Pair* problem there is the following recurrence:

$$U(n, d) = 2U(n/2, d) + U(n, d-1) + O(n)$$

for the running time of the divide-and-conquer algorithm to solve the *All Close Point Pairs* (ACPP) problem. It is then claimed that this solves to $U(n, d) = O(n(\log n)^{d-1})$.

This claim is not quite accurate because for $d = 1$, the claim would translate to $U(n, 1) = O(n)$, but every known algorithm for the 1-dimensional ACPP problem requires $\Omega(n \log n)$. Furthermore, if we use $U(n, 1) = O(n \log n)$ in the recurrence for $d = 2$, we get that $U(n, 2) = O(n \log^2 n)$, again contradicting the claim in this slide.

To understand what is going on, let us focus on the 2-dimensional ACPP problem and **show** that it is possible to improve the running time from $O(n \log^2 n)$ to $O(n \log n)$ by pre-sorting the points (by x-coordinate and separately by y-coordinate) and then sending in the sorted lists into the recursive calls.

Your answer should consist of a plain English description of this modified algorithm (no pseudocode necessary) followed by the correct recurrence relation for $U(n, 2)$.
Note: Once we show that $U(n, 2) = O(n \log n)$, it is easy to use the recurrence in the posted notes to show that $U(n, d) = O(n(\log n)^{d-1})$.
