Algorithms — CS 102 — Winter 2015
Instructor: Achlioptas
Homework 1

1. (10 pts) Suppose you are sent a sorted list of the distinct integer ID numbers of the students in the class. The IDs arrive in increasing order and your router dutifully puts them into a circular buffer, but then bursts into flames because you left it under a pile of dirty clothing on a hot day. You are thus left with an array containing an increasing sequence of integers, but one that starts somewhere in the middle of the array and wraps around. For example, $[68, 73, 75, 122, 16, 19, 33, 36, 38, 55, 59]$. Design an algorithm to find the last ID (the largest integer) in the list in $O(\log n)$ time.

2. (20 pts) You are tallying votes from an election in which n people voted. If any one candidate gets more than half (at least $\lfloor n/2 \rfloor + 1$ votes), they win. Otherwise a runoff is needed. For privacy reasons you are not allowed to look at any one ballot, but you have a machine that can take any two ballots and answer the question: "are these two ballots for the same candidate, yes or no?"

   (a) Design a divide and conquer algorithm that decides whether a runoff is needed which uses $O(n \log n)$ ballot equality tests, assuming that $n = 2^k$ for some integer k.

   (b) Explain how to supplement your algorithm to deal with n that is not a power of 2.

3. (10 pts) Consider an n-node complete binary tree T, where $n = 2^d - 1$ for some d. Each node v of T is labeled with a distinct real number x_v. A node v of T is a *local minimum* if the label x_v is less than the label x_w for all nodes w that are joined to v by an edge (parent and children). You are given such a complete binary tree T, but the labeling is only specified in the following *implicit* way: for each node v, you can determine the value x_v by probing the node v. Show how to find a local minimum of T using only $O(\log n)$ probes to the nodes of T.

4. (10 pts) In this problem we consider a *strictly decreasing* function $f : \mathbb{N} \to \mathbb{Z}$. That is, f takes as input any natural number ($i \in \mathbb{N}$) and returns an integer such that for any i, $f(i) > f(i+1)$. Additionally, $f(0) > 0$. We want to find

$$n = \min\{i \in \mathbb{N} : f(i) \leq 0\} \ .$$

   That is, we want to find the "'first place where f is at or below the horizontal axis."

   Assume we can compute $f(i)$ for any input i in constant time. Clearly, we can solve the problem in $O(n)$ time by evaluating $f(1), f(2), f(3), \ldots$ until we see a non-positive number. Give an $O(\log n)$ algorithm.

5. (20 pts) You are a manager at the local solar farm, *Fossil Fools*. You have a weather forecast for the next n days, and want to know what the most profitable single stretch to run your panels for would be. On some days, the running cost is more than the gain, but it may be worth leaving them on to get the days on either side (your solar panels will only turn on for one single stretch of days). More formally, if you are given an array $A[1...n]$ of integers, where $A[k]$ represents the amount of money you would make running your panels on day k, find values i and j with $1 \leq i \leq j \leq n$ maximizing

$$\sum_{k=i}^{j} A[k] \ . \tag{1}$$

   Example: If $A = [4, -5, 22, -7, 8, -10, 5]$, the solution is $i = 3$ and $j = 5$ (as $22 - 7 + 8 = 23$).

To design an efficient algorithm for this problem, we consider the s *fixed-start* problem s-FS. This problem consists of finding a value j with $s \leq j \leq n$ maximizing

$$\sum_{k=s}^{j} A[k] \ . \tag{2}$$

In English, it represents the simplification of the problem where I tell you that your segment absolutely must start on day s, and then ask what the optimal segment is. Similarly, the e *fixed-end* problem e-FE, consists of finding a value e with $1 \leq i \leq e$ maximizing

$$\sum_{k=i}^{e} A[k] \ . \tag{3}$$

Example: If $A = [4, -5, 6, 7, 8, -10, 5]$, the solution 4-FS is $j = 5$ (as 7+8=15) and the solution to 7-FE is $i = 3$ (as $6 + 7 + 8 - 10 + 5 = 16$).

(a) Describe $O(n)$ time algorithms for solving s-FS and e-FE (for given s and e).

(b) Given a $O(n)$ time algorithm for s-FS describe a simple $O(n^2)$ algorithm for the original problem.

(c) Given $O(n)$ time algorithms for s-FS and e-FE describe an $O(n \log n)$ divide-and-conquer algorithm for the original problem.

Disclaimer

You have to justify everything. This means that you have to explain (prove) both why your algorithm is correct, i.e., why it produces the correct answer for every input, and why it runs in the time that you claim. If you leave a problem completely blank, you get 25%, but if you submit an answer, it is graded from 0-100%. (Note, if you're leaving a problem blank, please make a note that you are doing it intentionally).

Think of the person reading what you write as your boss who would like to earn praise for the group by showcasing your solution at the group's presentation to "the higher ups". This means that your boss *wants* your solution to be correct, but she is not willing to risk public humiliation in case you "missed" something. So, she will be reading what you write very carefully, but with good intent[1]. Make her life easy by explaining things in a complete, lucid way that makes it clear that you've considered all the details. Explicitly stating all the details, describing data structures, and (God forbid!) presenting code-style-pseudocode is **not** the way to do this. Formulating principles, invariants and properties that are maintained by your algorithm **is** the way to do it[2]. And much harder.

[1] If you think that this is a negative way of putting things, enforcing cultural and class stereotypes, I strongly suggest you stay in college for as long as you can afford...

[2] The textbook itself, in my opinion, does a pretty good job at this. Don't be afraid to use words, but do so in complete, well-thought sentences. Take time. Don't "text" your answers.