Theory Problems

Help

The following problems are for those of you looking to challenge yourself beyond the required problem sets and programming questions. Most of these have been given in Stanford's CS161 course, Design and Analysis of Algorithms, at some point. They are completely optional and will not be graded. While they vary in level, many are pretty challenging, and we strongly encourage you to discuss ideas and approaches with your fellow students on the "Theory Problems" discussion form.

1. [Posted April 29th] You are given as input an unsorted array of n distinct numbers, where n is a power of 2. Give an algorithm that identifies the second-largest number in the array, and that uses at most $n + \log_2 n - 2$ comparisons.
2. [Posted April 29th] You are a given a *unimodal* array of n distinct elements, meaning that its entries are in increasing order up until its maximum element, after which its elements are in decreasing order. Give an algorithm to compute the maximum element that runs in O(log n) time.
3. [Posted April 29th] You are given a sorted (from smallest to largest) array A of n distinct integers which can be positive, negative, or zero. You want to decide whether or not there is an index i such that A[i] = i. Design the fastest algorithm that you can for solving this problem.
4. [Posted April 29th] Give the best upper bound that you can on the solution to the following recurrence: $T(1) = 1$ and $T(n) \leq T([\sqrt{n}]) + 1$ for $n > 1$. (Here [x] denotes the "floor" function, which rounds down to the nearest integer.)
5. [Posted April 29th] You are given an n by n grid of distinct numbers. A number is a *local minimum* if it is smaller than all of its neighbors. (A neighbor of a number is one immediately above, below, to the left, or the right. Most numbers have four neighbors; numbers on the side have three; the four corners have two.) Use the divide-and-conquer algorithm design paradigm to compute a local minimum with only O(n) comparisons between pairs of numbers. (**Note:** since there are n^2 numbers in the input, you cannot afford to look at all of them. **Hint:** Think about what types of recurrences would give you the desired upper bound.)
6. [Posted May 19th] Prove that the worst-case expected running time of every randomized comparison-based sorting algorithm is $\Omega(n \log n)$. (Here the worst-case is over inputs, and the expectation is over the random coin flips made by the algorithm.)
7. [Posted May 19th] Suppose we modify the deterministic linear-time selection algorithm by grouping the elements into groups of 7, rather than groups of 5. (Use the "median-of-medians" as the pivot, as before.) Does the algorithm still run in $O(n)$ time? What if we use groups of 3?
8. [Posted May 19th] Given an array of n distinct (but unsorted) elements $x_1, x_2, \ldots, x_n$ with positive weights $w_1, w_2, \ldots, w_n$ such that $\sum_{i=1}^{n} w_i = W$, a *weighted median* is an element x_k for which the total weight of all elements

with value less than x_k (i.e., $\sum_{x_i < x_k} w_i$) is at most $W/2$, and also the total weight of elements with value larger than x_k (i.e., $\sum_{x_i > x_k} w_i$) is at most $W/2$. Observe that there are at most two weighted medians. Show how to compute all weighted medians in $O(n)$ worst-case time.

9. [Posted May 19th] We showed in an optional video lecture that every undirected graph has only polynomially (in the number n of vertices) different minimum cuts. Is this also true for directed graphs? Prove it or give a counterexample.
10. [Posted May 19th] For a parameter $\alpha \geq 1$, an α*-minimum cut* is one for which the number of crossing edges is at most α times that of a minimum cut. How many α-minimum cuts can an undirected graph have, as a function of α and the number n of vertices? Prove the best upper bound that you can.
11. [Posted May 26th] In the 2SAT problem, you are given a set of clauses, where each clause is the disjunction of two literals (a literal is a Boolean variable or the negation of a Boolean variable). You are looking for a way to assign a value "true" or "false" to each of the variables so that all clauses are satisfied --- that is, there is at least one true literal in each clause. For this problem, design an algorithm that determines whether or not a given 2SAT instance has a satisfying assignment. (Your algorithm does not need to exhibit a satisfying assignment, just decide whether or not one exists.) Your algorithm should run in $O(m+n)$ time, where m and n are the number of clauses and variables, respectively. [Hint: strongly connected components.]
12. [Posted May 26th] In lecture we define the length of a path to be the sum of the lengths of its edges. Define the *bottleneck* of a path to be the maximum length of one of its edges. A *mininum-bottleneck path* between two vertices s and t is a path with bottleneck no larger than that of any other s-t path. Show how to modify Dijkstra's algorithm to compute a minimum-bottleneck path between two given vertices. The running time should be $O(m \log n)$, as in lecture.
13. [Posted May 26th] We can do better. Suppose now that the graph is undirected. Give a linear-time ($O(m)$) algorithm to compute a minimum-bottleneck path between two given vertices.
14. [Posted May 26th] What if the graph is directed? Can you compute a minimum-bottleneck path between two given vertices faster than $O(m \log n)$?

Created Thu 2 Feb 2012 11:18 PM PST

Last Modified Mon 26 May 2014 9:53 AM PDT