

Final*DUE DATE: Dec 15, 2008***Instructions:**

- The exam **MUST** be submitted no later than 5pm Monday Dec 15, whether electronically or by hand.
- The exam is open-book: you may use any reference material from the textbook, or from the Cormen/Leiserson/Rivest/Stein book, or from any handout I have provided/linked to. You **MAY NOT** use answers found on the web at large. If I discover that you have done so, you will be severely penalized.
- If you need a desired result (running time of Kruskal's algorithm, etc) that is not directly related to the problem you're solving, you may cite it. For example, you can say, "We can compute an MST in $O(m \log n)$ time using Kruskal's algorithm (Kleinberg/Tardos, Chapter 4.5). Citations must come from one of the two textbooks indicated above.
- Unless clearly specified, all statements require proofs. If an algorithm is randomized, you must provide either an analysis of the expected running time, or a proof that the probability of failure is no greater than $\frac{1}{3}$.

1 Approximate Medians

The *median* of a set of n distinct numbers $x_1 \dots x_n$ is the number x_j such that the sets $L = \{x_i < x_j\}$ and $R = \{x_i > x_j\}$ each have cardinality $(n - 1)/2$ (we'll assume for convenience that n is odd).

Another way of defining the median is via its *rank*. Let the rank of a number x_i be given by

$$r(x_i) = |\{x \leq x_i\}|$$

In other words, the minimum element of the set has rank 1, the maximum has rank n , and the median has rank $(n + 1)/2$.

For many applications, we use the median as a way of creating a balanced split in the data for performing recursion (QUICKSORT is one such example). For such applications, it's not critical that we compute the exact median, since any reasonably balanced splitter will suffice as well. This could potentially benefit us, since computing an approximate median might be faster than computing the exact median.

Let an ϵ -approximate median be an element x such that

$$(1 - \epsilon)(n + 1)/2 \leq r(x) \leq (1 + \epsilon)(n + 1)/2$$

In other words, the rank of x *approximates* the rank of the median element.

Question. Show that given a set of n distinct numbers, we can compute an ϵ -approximate median in time *that depends only on* ϵ . In other words, your algorithm running time *will not depend on* n .

2 Sloppy Search

We know that binary search on a set of n numbers takes $O(\log n)$ time. Suppose we are now given a set S of n numbers in the range $[1..n^2]$. Further, we are willing to be sloppy: given a query number q , instead of determining whether q is exactly contained in the set, we are willing to tolerate an answer $\tilde{q} \in S$ such that

$$|q - \tilde{q}| \leq \epsilon q$$

For example, if we set $\epsilon = .1$, then for a query $q = 5$, we are allowed to return any number in S in the range $[4.5, 5.5]$.

Question. Show that you can process the input such that a query can be answered sloppily in time $O(g(\epsilon) \log \log n)$, where $g(\epsilon)$ is some function of ϵ .

HINT: Think about reducing the size of the input so that “regular” binary search can be applied.

3 To claim the Clay, or not to claim...

You've just been promoted to assistant CAO (Chief Algorithms Officer) at the offices of Church, Turing and Gödel, Esq. Things are going smoothly: you've debunked three fake $P = NP$ proofs, and spotted at least one attempted violation of the Halting Problem. But then, one day, your underlings Alice and Bob burst into your office, and the following conversation ensues:

ALICE I just discovered this amazing approximation algorithm !

BOB So did I !

ALICE In fact, it's a $(1 + \epsilon)$ -approximation.

BOB So's mine !

ALICE Mine runs in linear time

BOB And so does mine.

ALICE Aha, but my actual running time is $O(n/\epsilon^{10})$.

BOB Mine is even better: $O(n/\epsilon^2)$.

YOU (trying to cut through the clamor) But which problem did you solve ?

ALICE VERTEX COVER

BOB KNAPSACK

YOU Well, that's impressive. Bob, you can take your result down to the department of suddenly irrelevant algorithms. Alice, you'll need to book a plane ticket to Boston to collect your \$1,000,000 prize for proving $P = NP$.

Question. Explain your responses to Alice and Bob.

4 Greedy Colorings

As usual, a coloring of a graph is an assignment of colors $1, \dots, k$ to vertices so that no two vertices connected by an edge have the same color. Consider the following greedy strategy for coloring a graph:

1. Order the vertices in some manner.
2. Color the vertices in order, assigning the lowest-available color to a vertex (in other words, check its colored neighbours and pick the lowest available color).

For example, if we're given a path $v_1 - v_2 - \dots - v_n$, and we order the vertices in the path order, then we use color 1 for v_1 , color 2 for v_2 , color 1 for v_3 , and so on. More generally, we use as many colors as we need to color the graph in this manner.

Question. **a)** Show that for any graph, there exists a sequence of vertices such that the greedy coloring uses the optimal number of colors. Note that since graph coloring is NP-complete, you're not expected to find this sequence in polynomial time (but if you do, there's an instant A+ for you !)

b) On the other hand, the greedy coloring scheme can be quite bad. Show that there exists a 2-colorable graph and a vertex ordering for which the greedy coloring algorithm uses $\Theta(n)$ colors.

HINT: A 2-colorable graph is bipartite.