Prof. J. Spencer
Oct 9, 2007

BASIC ALGORITHMS MIDTERM

Solutions and Commentary

No books or notes. Calculators OK.
Total points: 115. You *must* leave out 15 (or more) problem parts and specify these parts.

> I was thrown out of college for cheating on the metaphysics exam.
> I looked into the soul of the boy next to me.
> – Woody Allen

1. (30) Consider the Gale-Shapley algorithm with n men and n women.

   (a) (15) Describe the algorithm (you do *not* need to show correctness.) Be sure to include in your description a description of what data structures you used.
   Note: This is pretty well covered in the text and on the website. In the algorithm it is important that the data structures be updated. So when woman w jilts man m' for man m, m' is added to the freeman queue.

   (b) (15) Give a *detailed analysis* for how much time the algorithm takes. Consider only upper bounds and write your answer as $O(f(n))$ for some standard function $f(n)$. (Very little credit for only the answer, the analysis is key here.)
   Note: Most students had that the WHILE loop would occur at most n^2 times since each man was going down the list of the n women. But one also needs to argue that the time in the WHILE loop is constant. This is where the data structure really comes in. You need a queue of FREEMEN to quickly find a freeman, you need the current mate of each woman w (including NIL for none) to decide if w is single, and then for each man m you need w's rank of m (NOT on a linked list, that would be too slow to search) so that w can quickly decide who she prefers.

2. (25) A computer does a billion operations a second. An algorithm `GOODSORT` sorts an array of size n with $n \log_n$ operations. An algorithm `BADSORT` sorts an array of size n with n^2 operations. An algorithm `AWFULSORT` sorts an array of size n with 2^n operations. (Remark:

Some students used log to the base 10. I didn't take off any points but in CS base 2 is the default value!)

(a) (5) What is, roughly, the largest n for which GOODSORT will sort an array of size n in a day.
Note: Most all students had that there are 86400 seconds in a day and so roughly $8 \cdot 10^{13}$ operations per day. Now you want a rough solution for $n \lg n = 8 \cdot 10^{13}$. In that area, $\lg n$ is roughly 40 so dividing by 40 gives n around $2 \cdot 10^{12}$. (Observe that n is now a smaller so $\lg n$ is smaller, but $\lg n$ is still pretty close to 40 so this is a pretty good estimate.)

(b) (5) What is, roughly, the largest n for which BADSORT will sort an array of size n in a day.
Solution: Taking the square root gives roughly nine million

(c) (5) What is, roughly, the largest n for which AWFULSORT will sort an array of size n in a day.
Solution: Taking the lg gives around 46. This is a good example of why exponential time algorithms should be avoided!

(d) (5) How long will it take GOODSORT to sort a billion items.
Solution. The number of operations is a billion times lg of a billion which is around 30 billion (recall lg(1000) is essentially 10 as 2^{10} is 1024 which is very close to 1000) so dividing by a billion gives around 30 seconds.

(e) (5) How long will it take BADSORT to sort a billion items.
Solution: A billion squared divided by a billion gives a billion seconds. But how long is that? A little calculation gives around 31 years. The gigasecondaversary is a time to celebrate!

3. (15) Let G be the complete graph with vertex set $\{1, \ldots, n\}$. This means that every pair of vertices are adjacent, so that $\texttt{ADJ}[i]$ consists of all vertices $v \neq i$. Assume, for convenience, that the adjacency lists are in increasing order.

Consider the breadth first search algorithm $\texttt{BFS}[G, 1]$. Describe how the algorithm will work. What will be the Breadth First Tree T at the conclusion of the algorithm. A nice picture would help!
Solution: This was pretty good. The key is that 1 finds $2, 3, \ldots, n$. After the other vertices look through their own adjacency lists but don't find anything new. So the tree is a star with 1 adjacent to $2, 3, \ldots, n$. (Note the DFS is very different!)

4. (15) No proofs needed below
Note: Of course, there are many correct answers.

(a) (5) Describe a natural problem (best would be something described in class) that takes time $\Theta(n^2)$.
A Solution: Multiplying two n digit numbers (or multiplying two polynomials of degree n with small coefficients) with the method described in class.

(b) (5) Describe a natural problem (best would be something described in class) that takes time $\Theta(n \lg n)$.
A Solution: Sorting n objects. Similarly, given a heap on n objects applying EXTRACTMIN n times until getting a sorted list.

(c) (5) Describe a natural problem (best would be something described in class) for which there is no algorithm (currently!) known taking polynomial time
A Solution: Finding an independent set of maximal size in a graph.

5. (30) Consider the Heap H with `length(H)`=10 given by (Mea culpa: This is not a heap as discussed during the class.)

i	1	2	3	4	5	6	7	8	9	10
$H[i]$	5	2	7	9	1	3	8	0	4	6

(a) (5) Draw a nice picture of the heap as a binary tree.
Everyone got this one.

(b) (15) Illustrate the steps of `EXTRACTMIN[H]`, showing (as nice pictures) the intermediate steps and the final state of H and with *brief, but cogent* explanations of each step.
Note: A number of students made a subtle but important and instructive error. They remove 5 from the top of the heap and then filled the top with the smaller of the children and then iterated down the tree. The problem with this is that it changes the shape of the tree. Since heaps are a dynamic data structure this is no good. Instead, you fill the top of the heap with the record in the last position, here 6. Then you `HEAPIFY-DOWN[H,1]`.

(c) (5) When a heap has length n how long does `HEAPIFY-UP[H,1]` take? (Warning, trick question!)
Solution: Constant time as you can't go up from 1. (We prefer constant time to "no time" as it takes time to realize that you should do nothing!)

(d) (5) When a heap has length n how long does `HEAPIFY-DOWN[H,1]` take? (Brief reason, please.)
Solution: $O(\lg n)$ as you might go all the way down the tree which has depth $\lg n$.

'A knot!', said Alice, already to make herself useful, and looking anxiously about her. 'Oh, do let me help to undo it!"
– from Alice in Wonderland, by Lewis Caroll