CS-170 Solution to Exam 1 February 22, 1995

David Wolfe

Problems have been rephrased a little bit for clarity.

There are two requirements for passing the course: (1) You must submit the questionaire from the first day of classes and (2) If I did not take your picture, you must submit one to me. Please complete both of these promptly. And if you have not given me \$2 for copy charges, please do so.

1. (20 points) Ms. Jones has an algorithm which she proved (correctly) runs in time $O(2^n)$. She coded the algorithm correctly in C, yet she was surprised when it ran quickly on inputs of size up to a million. What are at least **two** possible explanations of this behavior? (Two rather different plausible explanations will receive full credit. If you give additional explanations, you may lose points for unplausible ones. Be brief in your explanations.)

I can think of four explanations, the first of which I think are most plausible:

- Perhaps the algorithm runs in linear time. $O(2^n)$ means *at most* time a constant times 2^n. Even a linear time algorithm is $O(2^n)$. $\Theta(2^n)$ would mean the algorithm takes time *at least* a constant times 2^n.
- Perhaps the input to her program is not the worst case input; she is entered the *easiest* inputs deal with.
- The exponential behavior doesn't kick in until n is huge.
- The following two explanations are implausible since $2^{1,000,000}$ is so ridiculously huge. After all, the number of particles in the visible universe is less than 10^{90}, or 2^{200}. These implausible explanations are (a) The constant in front of 2^n is extremely small, and (b) The computer is fast.

You earned 12 points for each clear good explanation and 4 points for an unplausible one to maximum of 20. If you gave any unplausible explanation, you received a maximum of 18 points.

2. (20 points) Consider the following recurrence, where $a + b < 1$ and $a, b \geq 0$:

$$\begin{aligned} T(n) &= 0 &&, \text{if } n \leq 1 \\ T(n) &= T(an) + T(bn) + n, &&\text{if } n > 1 \end{aligned}$$

Prove $T(n) = O(n)$.

We want to show $T(n) \leq cn$ for some value of c. Then,

$$\begin{aligned} T(n) &= T(an) + T(bn) + n \\ &\leq can + cbn + n \qquad \text{(by induction)} \\ &= (c(a+b)+1)n \\ &\leq cn \qquad \text{(for } c \text{ sufficiently large but fixed as below.)} \end{aligned}$$

Now, $c(a+b)+1 \leq c$ if $c \geq \frac{1}{1-a-b}$ (which is greater than zero since $a+b<1$), so we'll choose the fixed constant to be $c = \frac{1}{1-a-b}$. For the base case of $n=0$, $T(n) = 0 \leq c \cdot n$ for $n \leq 1$. *The most common mistake was the following fallacious proof (or an elaborately disguised variant):*

$$\begin{aligned} T(n) &= T(an) + T(bn) + n \\ &\leq O(n) + O(n) + n \quad \textit{(by induction)} \\ &= O(n) \end{aligned}$$

The reason this is fallacious is because it is critical that the same constant c be used for all n (once n exceeds some small n_0, if you wish). Any proof which failed to take advantage of the fact that $a+b<1$ fell into this trap. In fact, when $a+b=1$, the growth should be $\Theta(n \log n)$, and when $a+b>1$, the growth should be exponential in n.

3. (10 points) For which type of input data is Huffman coding more likely to achieve better compression: random characters or English text. Why? (A one sentence explanation is sufficient.)

English text. Huffman coding compresses better the more repetition is present. Short codewords will be used to express common letters or words.

4 & 5. **The following text refers to the next two problems:**

You are going on a long trip. You start along the road at mile post 0. Along the road that you will travel there are n hotels at mile posts $a_1 < a_2 < \cdots < a_n$ (a_i is measured from the start of the trip). When you choose to stop, you must stop at one of these hotels (but you can choose which hotels you want to stop at). You must stop at the last hotel, which is your destination. You decide that the ideal distance to travel per day is 300 miles (plus or minus a few is ok); so if x is the number of miles traveled in one day, you assign a cost function of $(300-x)^2$ that you want to minimize.

4. (30 points) Design a dynamic programming algorithm to determine your total cost when you choose to stop at those hotels which minimize the total cost function. (Hint: Let C_i be the minimum cost if you were to start at mile 0 and complete your trip at hotel i.)

   (a) (15 points) Give a recursive rule for computing C_i.

   (b) (10 points) Explain how you can use dynamic programming to compute C_i in polynomial time.

   (c) (5 points) Analyze the running time of your dynamic programming algorithm.

Let C_i be the minimum cost in which to break your travel up, assuming that your last stop is hotel i. Let's let hotel 0 denote our starting point. As a basis, let $C_0 = 0$. In general, to compute C_i, we consider all possible places k, $0 \leq k < i$, that we might have stopped the night before. The cost of having k as the previous stop is the minimum cost of getting to hotel k, followed by the cost of traveling in one day from k to i, a distance of $(a_i - a_k)$. Minimizing over all k gives the following rule:

$$C_i = \min 0 \leq k < i \left(C_k + (300 - (a_i - a_k))^2\right)$$

A dynamic programming problem simply loops through all i (from 1 to n), building a linear array for the C_i values using the recurrence to calculate C_i from $C_0, \ldots, C_{i-1}$. C_n is the minimum cost we're looking for. If, in the table, for each values of i we also fill in the value of k which minimizes C_i, we can recreate the actual hotels we should stop at.

It takes linear time to calculate each value of C_i, for a total time of $O(n^2)$.

For part a), you got at least 10 points if you could write out the recurrence relation, plus or minus minor errors (wrong base case, incorrect minimizing bounds, informal but clear description). Some credit was given for an incorrect formalization, as long as it was a recurrence relations.

Common mistakes included: not minimizing over a varying number of possibilities, using C_{i-1} in the place of C_k, and assuming you already knew the list of optimal stops to compute C_i. There was also some confusion as to where the recurrence should begin, at $i = n$ or $i = 1$. The problem is symmetric. Therefore if you defined C_i with your base case at C_n, you were not penalized but your definition of C_i had to be consistent (i.e. you could not use C_{i-1} to define C_i, since the recursion would never stop).

If you decided to use a different critter with more than one index and defined it correctly, you were not penalized for that. However, your answer to part b) had to be consistent in any case with your answer to part a). That is if you were using a one-dimensional critter in a), you were expected to describe a one-dimensional array in b).

For part b), if you mentioned that you needed a one-dimensional array you were given 3 points. Saying that the C_i's had to be computed in increasing order received another 3 points. If you got to that point then the remaining 4 points were for mentioning that dynamic programming allowed you to work through the subproblems only once and reuse your work.

A very common mistake here was mentioning the need for a two-dimensional table (costs from any hotel to any hotel) after defining a critter with only one index in part a). This received no credit since the table described was irrelevant to the problem. Another common error was to build a table from any hotel to any hotel for the cost function, which can always be calculated as the need arises. Again, since the table was irrelevant, no credit was awarded.

For part c), you had to answer consistently with your previous answers. If you described an $O(n^3)$ algorithm in a) and b), and then answered $O(n^2)$, you were awarded no credit. If you just wrote down the answer with no explanation, or if you gave a correct explanation but a wrong answer, you were given 2 points. If you could write down the order of entries in the array, or the order of each computation in the array, that was also worth 2 points.

5. (30 points) Propose and discuss a greedy heuristic for finding which hotels to stop at. (Your heuristic need not actually minimize the total cost function.)

   (a) (10 points) Propose a reasonable linear-time greedy heuristic for the problem.

(b) (15 points) Either prove your greedy heuristic minimizes the total cost function, or give a counterexample demonstrating how your heuristic may fail to give the optimum set of hotels to stop at.

(c) (5 points) Analyze the running time of your heuristic.

One greedy heuristic is each day stop at the hotel up ahead nearest to 300 miles from where you started travel that day. Alas, this doesn't give the best schedule for hotels at mile posts 290, 300, 580, since it's better to stop at 290 the first night (for a total cost of $10^2 + 10^2 = 200$) rather than 300 the first night (for a total cost of $0^2 + 20^2 = 400$).

This can be made to run in linear time, since each day we need only consider hotels up to the next one $>$ 300 miles ahead; and even that hotel will only be considered twice (today and tomorrow).

For the greedy heuristic, I took off 5 points if your solution was not locally optimal. Common examples of non-locally optimal heuristics took the last hotel within 300 miles or the first hotel at least 300 miles from the last stop. Notice, some of these heuristics are not guaranteed to always find a solution. Also, some solutions said stop at a hotel that's within a "few" miles of 300 miles from the last stop. This makes no sense what a "few" means. These also lost 5 points.

For part (b), a proof that your greedy heuristic works lost 15 points.

When analyzing your running time, you had to explain that each hotel was considered at most twice. Simply saying each hotel was considered at least once shows it's running time is $\Omega(n)$. Such an argument or a similar mistake in your analysis lost 3 points. No points were given if no explanation was given of why the running time is $O(n)$ or it was only shown to be something non-linear, such as $O(n^2)$. Notice you can not find those hotels closest to some mile mark in constant time. This would actually take $O(logn)$ time since it would require a binary search. It is also incorrect to analyze your running time based on the number of miles driven. It should be dependent on, n, the number of hotels.

6. (20 points) Consider the problem of finding the largest, second largest and third largest from a collection of 8 elements using comparisons. (The 3 largest elements should be reported in decreasing order.) You may assume the elements are distinct.

   Let u be an upper bound on the number of comparisons required to solve this problem.. (The value of u would be a number, like "17", since there is no parameter such as n in the problem.) Let l be a lower bound on the number of comparisons required.

   (a) (2 points) Is it possible that $u < l$? What would you conclude?

   (b) (2 points) Is it possible that $u > l$? What would you conclude?

   (c) (4 points) Is it possible that $u = l$? What would you conclude?

   (d) (6 points) Find an upper bound, u, on the number of comparisons required.

   (e) (6 points) Find a lower bound, l, on the number of comparisons required. (I recommend an information theoretic bound, since it's simplest.)

   (You'll certainly receive full credit for parts (d) and (e) if $|u - l| \leq 3$. A little worse should be ok, too.)

(a) No. Since l lower bounds the running time of *any* algorithm, a mistake must have been made.

(b) Yes. Either a better lower bound or a better algorithm exists. The best algorithm takes between l and u comparisons.

(c) Yes. Algorithm A is an optimal algorithm.

(d) I'll propose an algorithm, A, which takes 12 comparisons. Find the maximum using a tournament (7 comparisons). Then find the largest among those three who lost to the winner of the tournament (2 comparisons). Then find the largest among those (at most four) remaining who lost to either the largest or second largest (3 comparisons). This yields $u = 7 + 2 + 3 = 12$ comparisons.

A better algorithm selects the elements to compare in the second round in the right order to guarantee only three candidates for third largest. This yields 11 comparisons, but requires justification.

(e) There are $8 \cdot 7 \cdot 6 = 336$ possible outputs. An information theoretic lower bound is therefore $\log(8 \cdot 7 \cdot 6) = 9$ comparisons.

Another possible lower bound (not as good) is 7 to find the maximum alone. More sophisticated lower bounds are possible.

You earned 3 points for the right answers (yes or no) to parts a-c. You earned full credit if you demonstrated an understanding for what a lower bound is in your reasons for a-c. By far, the most common mistake was to think that a lower bound is the best case running time for an algorithm. This is false. The performance of an algorithm is measured by the worst case input, whether we're talking about a lower bound or an upper bound. A lower bound lower bounds the performance of all possible *algorithms which solve a problem.*

Unfortunately, since the original phrasing of the question referenced a specific algorithm, A, many students thought the lower bound referred to the performance of that algorithm. If you made this interpretation, the lower bound should have been on the worst case performance of the algorithm. On occasion, this would be of interest when a specific algorithm's performance is hard to analyze completely. Because of the confusion, I gave a full 8 points. I even gave full credit if you thought the lower bound referred to the performance of an algorithm on best input, but you should be aware that this is rarely of interest. **You should never again use best case input as a measure of a problem's performance in this course***, even though you got credit this time.*

For the upper bound, if you sorted (taking $n \log n = 24$) comparisons, you received 3 points. If you said sorting takes $O(n \log n)$, that means that it takes a constant times $n \log n$ comparisons, so this does not necessarily yield $n \cdot \log n$. If you found the maximum 3 times in $7+6+5$ comparisons, resp.) you earned 4 points.

For the lower bound, if you stated clearly that it takes at least as long as finding the maximum of 3 elements earned 4 points. A common mistake here was to give a better algorithm than you gave in part (d). (Perhaps you thought it would be the best possible, but you didn't prove it.) This just provides a better upper bound for the problem.