CS361 Final Answers

Cristopher Moore, Fall 2006

1. (40 points total) Consider the following pseudocode:

```
stuff(list L of length n) {
  if n==0 return;
  for (i=n down to 1) print L[i];
  list L1 = L[1...n/2]; // first half
  list L2 = L[n/2+1...n]; // second half
  list L3 = L[n/4+1...3n/4]; // middle two fourths
  stuff(L1);
  stuff(L2);
  stuff(L3);
}
```

   (a) Let $f(n)$ be the running time of `stuff` on a list of size n; in particular, let $f(n)$ be the total number of elements `stuff` prints out. Write a recurrence relation for $f(n)$.

   Answer. The function prints out n numbers and calls itself three times on lists of size $n/2$, so

$$f(n) = 3f(n/2) + n \ .$$

   (b) Solve this recurrence relation within Θ.

   Answer. Without the driving term, we have $f(n) = 3f(n/2)$. Trying a solution of the form $f(n) = n^a$ gives $a = \log_2 3$. This is larger than 1, so the driving term is insignificant, and we get

$$f(n) = \Theta(n^{\log_2 3}) \ .$$

2. (20 points) Name a function $f(n)$ which is bigger than any polynomial, but less than any exponential. In other words, $f(n) = \omega(n^k)$ for any constant k, but $f(n) = o(2^{cn})$ for any constant c.

   Answer. Lots of things work: a few examples are $f(n) = n^{\log n}$, or $2^{(\log n)^a}$ for $a > 1$, or $2^{\sqrt{n}}$.

3. (20 points) I want to sort a list of 5 elements with an algorithm that uses comparisons to decide how to rearrange the elements. What is the smallest number of comparisons I need to do this, and why?

   Answer. There are $5! = 120$ possible orders the elements can be in. By the "twenty questions" argument, we need to do at least $\log_2 120$ comparisons to distinguish them, and rounding up gives 7.

4. (40 points) When we remove the root element from a min-heap (for instance, because it represents the highest-priority task in a priority queue), we move the rightmost element from the bottom row up to the root, and then `downSift` that element, swapping it with the smaller of its two daughters until it finds a place where it is smaller than both its daughters. Prove that when we are done with this process, we again have a legal min-heap, in which every element is less than both its daughters. Start by describing the overall structure of your proof: if it is an inductive proof, state clearly the inductive hypothesis that you will try to prove is true at each stage of the process.

   Answer. Call the item we have moved up to replace the root x. First, note that since we assumed we had a min-heap to start with, the only places where the min-heap property might be violated are between x and its daughters. We will show that this inductively true. Suppose x's daughters are y and z, and let's say that y is the smaller one and that $x > y$. Then swapping x and y puts y on top; but since $y < z$ and $y < x$, y is now less than both its daughters, and it is again true that the only places which might violate the min-heap property are between x and its (new) daughters. This continues until x is less than both its daughters, but now the heap property holds everywhere.

5. (30 points) Given a binary tree, define the following quantity:

$$S = \sum_{\text{leaves}} 2^{-\text{depth}}$$

   Prove inductively that $S \leq 1$ for any binary tree. For which binary trees is $S = 1$?

   Answer. There are several proofs, but the simplest is to assume that this is true for both subtrees: call their values S_1 and S_2, and assume that $S_1, S_2 \leq 1$. Making these subtrees' roots the children of the root increases the depths of their leaves by 1, and this decreases their contribution by a factor of 1/2, so

$$S = \frac{S_1 + S_2}{2} \leq 1 \ .$$

   The base case is given by a tree consisting just of the root node, for which $S = 2^0 = 1$.

   Another nice proof involves growing the tree by adding new leaves: if we add two daughters to a leaf then S stays the same, and if we add only one it decreases.

   The trees for which $S = 1$ are those where every internal node has two daughters. This includes completely balanced trees, but lots of others as well.

6. (30 points total) Suppose I have a dynamic array structure called `growingArray`. At each point in time, it consists of an array `int array[maxSize]` and a number `size`. Each time I insert a new number x, I set `array[size]` $= x$ and increment `size`, unless `size` $=$ `maxSize` which means the array is full. In this case, I double `maxSize`, allocate a new array of that size, copy the contents of the old array into the new array, and then continue as before.

   (a) What is the total running time of a series of n inserts, within Θ, and what is the average running time? Justify your answer.

   Answer. The total running time is 1 per insert, for a total of n, plus the time it takes to copy the old array into new ones. This only happens when the size is a power of 2, so this is

   $$1 + 2 + 4 + 8 + \cdots + 2^k$$

   where 2^k is the largest power of 2 less than or equal to n. For simplicity say $2^k = n$; then this sum is

   $$1 + 2 + 4 + 8 + \cdots + n$$

   or, written backwards,

   $$n + \frac{n}{2} + \frac{n}{4} + \cdots = n\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right)$$

   This geometric sum converges to a constant (in fact, to 2), so we get a total of $\Theta(n)$. Dividing by the number of inserts, the average running time of a single insert operation is $\Theta(1)$.

   (b) What happens to the average running time if instead of doubling `maxSize` each time the array is full, I increase `maxSize` by 1000?

   Answer. Now we copy the array every thousand steps, which takes

   $$1000 + 2000 + 3000 + \cdots + n$$

   time. This is $\Theta(n^2)$, so the average time per insert is $\Theta(n)$.

7. (20 points) Suppose x is a number mod n. Roughly how many multiplications does it take to calculate

   $$x^{1{,}000{,}000} \bmod n\,?$$

   Your answer does not have to be exact, but it should be correct to within a factor of 2. Explain your answer.

   Answer. Since $1,000,000$ is about 2^{20}, it takes about 20 repeated squarings to get

   $$x, x^2, x^4, x^8, \cdots$$

   up to $x^{2^{20}}$, and then at most 20 multiplications to combine the powers of 2 that appear in 1000000's binary digit sequence. The total is at most 40.

   To be more precise we only need 19 squarings to get $x, x^2, x^4, \ldots, x^{2^{19}}$, and then just 6 multiplications to combine the powers $2^{19} + 2^{18} + 2^{17} + 2^{16} + 2^{14} + 2^9 + 2^6 = 1000000$. But we don't need to be this precise!