Homework #3
Introduction to Algorithms/Algorithms 1
600.363/463
Spring 2013

Solutions

February 8, 2014

1 Problem 1 (20 points)

If a list of n distinct numbers is randomly permuted into the array $A[1..n]$, show that the expected number of indices i for which $A[i] > A[j]$ for all $j < i$ grows as $O(\log n)$. Hint #1: note that any permutation has at least one such index, since the first element of the list trivially satisfies $A[1] > A[j]$ for all $j < 1$, because no such index j exists. Hint #2: a review of CLRS appendix C and the related material on the harmonic numbers will be helpful.

Answer: let us call index i a *record* if $A[i] > A[j]$ for all $j < i$, and let X_i be a ranom variable defined as

$$X_i = \mathbb{I}\{\forall j < i : A[i] > A[j]\}.$$

We are interested, then, in the quantity

$$\mathbb{E}(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} \mathbb{E}X_i, \tag{1}$$

where the equality follows by linearity of expectation. For $i = 1$, we have $\mathbb{E}X_1 = 1$, since the first element always satisfies the $A[i] > A[j]$ condition trivially. For $i > 1$, the probability that $X_i = 1$ as the probability that a random permutation of i distinct numbers maps the largest of those i numbers to position i. This probability is simply $\frac{1}{i}$. Thus,

$$\mathbb{E}X_i = \Pr[X_i = 1] = \frac{1}{i}.$$

Returning to (1) and plugging in this value for $\mathbb{E}X_i$, we have

$$\mathbb{E}(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} \frac{1}{i} = H_n.$$

We know from CLRS (and from lecture) that $H_n = O(\log n)$, which completes the proof.

2 Problem 2 (20 points)

2.1 (10 points)

Resolve the following recurrences. Use the Master theorem, if applicable. In all examples assume that $T(1) = 1$. All square roots are positive unless noted otherwise and all logarithms are base 2 unless noted otherwise. To simplify your analysis, you may assume that $n = a^k$ for some a, k.

1. $T(n) = 3T(n/2) + 1$

   Applying the master theorem, we have $a = 3$, $b = 2$ and $f(n) = 1$. $f(n)$ thus satisfies $f(n) = O(n^{\log_b a - epsilon})$ for $\epsilon < \log_b a$, since constant functions are $O(n^c)$ for any $c > 0$. Thus, the master theorem states that

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

2. $T(n) = T(n/2) + 2\sqrt{n}$

   Applying the master theorem, we have $a = 1$, $b = 2$ and $f(n) = 2\sqrt{n}$. $f(n) = 2n^{\frac{1}{2}}$ is $\Omega(n^{\log_2 1 + \epsilon})$ for any $0 < \epsilon < \frac{1}{2}$. We must also check that there exists a $c < 1$ such that for sufficiently large n, $af(n/b) \leq cf(n)$. That is, we must show that there exists a $c < 1$ such that for suitably large n we have

   $$\sqrt{n/2} \leq c\sqrt{n}.$$

   $c = \frac{1}{\sqrt{2}} + \delta$ for some small $\delta > 0$ will suffice. Thus the master theorem states that we must have

   $$T(n) = \Theta(f(n)) = \Theta(\sqrt{n}).$$

3. $T(n) = 16T(n/16) + n^{\frac{3}{2}}$

   Applying the master theorem, we have $a = b = 16$ and $f(n) = n^{\frac{3}{2}}$. $f(n) = n^{\frac{3}{2}} = \Omega(n^{1+\epsilon})$ for $0 < \epsilon < \frac{1}{2}$. We must also check that there exists a $c < 1$ such that for sufficiently large n, $af(n/b) \leq cf(n)$. That is, we must show that there exists a $c < 1$ such that for suitablt large n we have

   $$16(\frac{n}{16})^{\frac{3}{2}} \leq cn^{\frac{3}{2}}.$$

   $\frac{1}{4} < c < 1$ will suffice for this purpose. Thus the master theorem states that we must have

   $$T(n) = \Theta(f(n)) = \Theta(n^{\frac{3}{2}}).$$

4. $T(n) = 28T(n/3) + n^3$

   Applying the master theorem, we have $a = 28, b = 3$ and $f(n) = n^3$. Thus we have $f(n) = n^3$, which gives $f(n) = O(n^{\log_b a - \epsilon})$ for suitably small $\epsilon > 0$, since $\log_3 28 > 3$. Thus the master theorem states that we must have

   $$T(n) = \Theta(n^{\log_3 28}).$$

5. $T(n) = nT(n/2)$

   The master theorem does not apply here. Expanding the recurrence, we have

   $$\begin{aligned} T(n) &= nT(n/2) = n(\frac{n}{2}T(n/4)) \\ &= \frac{n^2}{2}T(n/4) = \frac{n^2}{2}\frac{n}{4}T(n/8) \\ &= \frac{n^3}{8}T(n/8) = \frac{n^3}{8}\frac{n}{8}T(n/16) \\ &= \frac{n^4}{64}T(n/16) = \frac{n^4}{64}\frac{n}{16}T(n/32) \\ &= \cdots = \frac{n^{k+1}}{2^{S_k}}T(\frac{n}{2^k}) \end{aligned}$$

   where

   $$S_k = \sum_{i=1}^{k} i = \frac{k(k+1)}{2}.$$

The recurrence ends when $k = \log n$. Plugging this in, we have

$$T(n) = \frac{n^{\log n+1}}{2^{S_{\log n}}} T(1).$$

We can rewrite the denominator of this expression as

$$\begin{aligned} 2^{S_{\log n}} &= 2^{\frac{1}{2}(\log n+1)\log n} \\ &= \left(2^{\log n}\right)^{\frac{1}{2}(\log n+1)} \\ &= n^{\frac{1}{2}(\log n+1)}. \end{aligned}$$

Plugging this back in for 2^{S_k}, we have

$$T(n) = \frac{n^{\log n+1}}{n^{\frac{1}{2}(\log n+1)}} = n^{\frac{1}{2}(\log n+1)}.$$

6. $T(n) = 2T(n-1) + 1$
Again, the master theorem does not apply. Expanding the recurrence,

$$\begin{aligned} T(n) &= 2T(n-1) + 1 = 2(2T(n-2) + 1) + 1 \\ &= 4T(n-2) + 2 + 1 \\ &= 4(2T(n-3) + 1) + 2 + 1 \\ &= 8T(n-3) + 4 + 2 + 1 \\ &= \dots \\ &= 2^k T(n-k) + \sum_{i=0}^{k-1} 2^i. \end{aligned}$$

The recurrence bottoms out when $k = n - 1$. Plugging this in, we have

$$T(n) = 2^{n-1}T(1) + \sum_{i=0}^{n-2} 2^i.$$

Applying basic summation identites and using the fact that $T(1) = 1$, we have

$$T(n) = 2^{n-1} + 2^{n-1} - 1 = 2^n - 1.$$

7. $T(n) = 8T(n/2) + n^3$
Applying the master theorem, we have $a = 8, b = 2$ and $f(n) = n^3$. In this case, we have $f(n) = \Theta(n^{\log_b a})$, since $\log_2 8 = 3$. Thus the master theorem states that we must have

$$T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n^3 \log n).$$

8. $T(n) = T(n/2) + n \log n$
   Applying the master theorem, we have $a = 1, b = 2$ and $f(n) = n \log n$. In this case, we have $f(n) = \Omega(n^{\log_b a + \epsilon})$ for any $0 < \epsilon < 1$. We must also check that there exists a $c < 1$ such that for sufficiently large n, $af(n/b) \leq cf(n)$. That is, we must show that there exists a $c < 1$ such that for suitablt large n we have
   $$\frac{n}{2} \log \frac{n}{2} \leq cn \log n.$$
   Doing a bit of algebra, we can see that choosing $\frac{1}{2} < c < 1$ will suffice. Thus the master theorem states hat we must have
   $$T(n) = \Theta(f(n)) = \Theta(n \log n).$$

2.2 (10 points)

A sequence $a_1, a_2, \ldots, a_n$ has a *dominant element* if more than half of the elements in the sequence are the same. Give a divide and conquer algorithm that finds and returns a dominant element in a sequence of n numbers or return None if no such element exists and runs in $O(n \log n)$ time. Prove the correctness of your algorithm and analyze its runtime. (Note: there exists an $O(n)$ algorithm to solve this problem that doesn't make use of divide and conquer– if you're clever enough to figure it out, you're welcome to prove its correctness and runtime instead.)

Answer: The intuitive solution is to apply divide and conquer as we have seen before– split the list (call it A) into two halves, say L and R. If the list had a dominant element, then it must also be the dominant element in at least one of these two lists. Assuming we can check each list for a dominant element, we can perform the "merge" step of the divide and conquer algorithm by checking if the dominant element of L (assuming it exists) appears enough times in R to be a dominant element of A and checking if the dominant element of R appears enough times in L to be a dominant element of A (note that the pigeonhole principle implies that at most one of these two conditions can hold true). This checking operation requires $O(n)$ time. A divide and conquer approach would then have a recurrence pattern essentially the same as mergesort, and would this require time $O(n \log n)$.

We can do better using Moore's voting algorithm, in which we make a single linear pass through the list counting how many times two consecutive elements have the same value to find a candidate element, then makes a second pass to verify whether or not the candidate element is actually a dominant element.