MCS/CS 401, Spring 2020, Midterm 1, Solution outlines

March 17, 2020

1. Is $3^n = O(2^n)$?

   No. If $3^n = O(2^n)$ then for c and n_0 it holds that $3^n \leq c2^n$ for every $n \geq n_0$. Dividing both sides by 2^n we get

   $$\left(\frac{3}{2}\right)^n \leq c.$$

   But $3/2 > 1$, so $(3/2)^n$ gets arbitrarily large as $n \to \infty$. So for every c there is a value n_1 such that $(3/2)^n > c$ if $n \geq n_1$.

   Additional note: taking logarithms of both sides we get $n \log(3/2) \leq \log c$, showing that $\lceil \log c / \log(3/2) \rceil + 1$ is a possible value for n_1.

2. Give a Θ-bound for the solution of the following recurrence using the master theorem:

   $$T(n) = 7 \cdot T\left(\frac{n}{2}\right) + n^3.$$

   Here $a = 7$, $b = 2$ and $f(n) = n^3$. It holds that $\log_b a = \log_2 7 < \log_2 8 = 3$. Thus there is a sufficiently small ϵ such that $(\log_2 7) + \epsilon \leq 3$. (*Note*: in fact $\log_2 7 < 2.81$, so $\epsilon = 0.1$ works.) Then

   $$n^3 = \Omega\left(n^{(\log_b a)+\epsilon}\right),$$

   corresponding to Case 3 in the master theorem. The additional constraint that needs to be checked is

   $$7\left(\frac{n}{2}\right)^3 \leq cn^3$$

   for some $c < 1$. Here $c = 7/8$ works. So Case 3 applies and $f(n) = \Theta(n^3)$.

3. Consider the following algorithm to produce a permutation of an array A of size n:

for $i = 1$ **to** $n - 1$

swap $A[i]$ with $A[RANDOM(i + 1, n)]$

Prove or disprove: the algorithm only produces permutations of the original array which are different from the identity permutation, and each such non-identity permutation is produced with the same probability.

Note: the identity permutation is the permutation which leaves every element in its original place.

Hint: you can argue about the total number of permutations produced by all possible runs of the algorithm.

The first part is true, but the second part is false, so the statement is false.

The algorithm only produces permutations different from identity. In the first iteration $A[1]$ is replaced by one of $A[2], \ldots, A[n]$, so $A[1]$ will not stay in place.

However, the algorithm does not produce every non-identity permutation. It is sufficient to consider the case $n = 3$. Then starting from 123, for $i = 1$ we get either 213 or 321, depending on whether $A[2]$ or $A[3]$ is swapped. In the next iteration the last two elements are swapped. So we get 231 or 312. Thus the algorithm can produce only 2 non-identity permutations (each with probability 1/2). There are altogether $3! - 1 = 5$ non-identity permutations. The remaining 3 non-identity permutations are produced with probability 0.

Note: a similar argument gives that in the general case the algorithm can produce $(n - 1)!$ non-identity permutations, which is fewer than the total number $n! - 1$ of all non-identity permutations.

4. Show the operation of HEAPSORT (displaying the sequence of trees) on the array

$$A = \langle 6, 3, 4, 5, 7, 1, 9 \rangle,$$

starting with the procedure BUILD-MAX-HEAP and continuing with the first call of the procedure MAX-HEAPIFY after that.

Omitted. *Note*: the problem did not ask for the complete HEAPSORT algorithm.

5. Given a set of n elements, we would like to find the k largest elements in sorted order, using the following approach. Use an efficient selection algorithm to find the k'th largest element, partition with respect to that element, and then sort the k largest elements.

Determine the worst-case number of comparisons of the resulting algorithm in terms of n and k. For what values of k do we get a linear time algorithm?

We consider the following algorithm:

(a) find the k'th largest element x using SELECT learned in class: running time $O(n)$.

(b) compare every element with x (this identifies the k largest elements): running time $O(n)$.

(c) sort the k largest elements using MERGESORT: running time $O(k \log k)$.

Thus the total running time is $O(n + k \log k)$.

This running is linear if $k \log k = O(n)$. This holds, for example, when $k \leq \sqrt{n}$, as then we get $\sqrt{n} \log \sqrt{n} = O(n)$. It also holds when $k \leq n^c$ for any $c < 1$, and even when $k = n/\log n$.