**CS 577: Introduction to Algorithms** 9/28/2015

# Homework 3 Solutions (Review Problems)

Instructor: Dieter van Melkebeek TA: Kevin Kowalski

## Problem 1

The recursion tree has depth $d \doteq \log_3(n)$ and has $4^i$ nodes of size $n/3^i$ at level $i$ for $0 \leq i \leq d$. The bottom level corresponds to instances of size 1 or 2, which each take at most $e$ steps for some positive constant $e$. Thus, the total amount of work spent on the bottom level is at most

$$e \cdot 4^d = e \cdot 4^{\log_3 n} = e \cdot (3^{\log_3 4})^{\log_3 n} = e \cdot (3^{\log_3 n})^{\log_3 4} = e \cdot n^{\log_3 4}.$$

If the local work is at most $c \cdot n^2$ for some positive constant $c$, then aggregating the work over all levels $i < d$ yields

$$\sum_{i=0}^{d-1} 4^i \cdot c \cdot \left(\frac{n}{3^i}\right)^2 = c \cdot n^2 \cdot \sum_{i=0}^{d-1} \left(\frac{4}{9}\right)^i \leq c \cdot n^2 \cdot \sum_{i=0}^{\infty} \left(\frac{4}{9}\right)^i = \frac{9c}{5} \cdot n^2.$$

Thus, the overall running time is at most $\frac{9c}{5} \cdot n^2 + e \cdot n^{\log_3 4} = \Theta(n^2)$. The recursion-tree is top-heavy.

If the local work is at most $c \cdot n$ for some positive constant $c$, then aggregating the work over all levels $i < d$ yields

$$\sum_{i=0}^{d-1} 4^i \cdot c \cdot \frac{n}{3^i} = c \cdot n \cdot \sum_{i=0}^{d-1} \left(\frac{4}{3}\right)^i = c \cdot n \cdot 3 \cdot \left(\left(\frac{4}{3}\right)^d - 1\right) \leq 3c \cdot n \cdot \frac{4^{\log_3 n}}{3^{\log_3 n}} = 3c \cdot n \cdot \frac{n^{\log_3 4}}{n} = 3c \cdot n^{\log_3 4}.$$

Thus, the overall running time is at most $3c \cdot n^{\log_3 4} + e \cdot n^{\log_3 4} = \Theta(n^{\log_3 4})$. The recursion tree is bottom-heavy.

## Problem 2

Note that since $A[1..n]$ is a subarray of itself, $A$ must contain some entry with a completely unique color. Suppose that $A[\ell]$ is this entry. Then, any subarray that contains $A[\ell]$ automatically has a uniquely colored entry, so we only have to worry about subarrays contained entirely in $L \doteq A[1..\ell-1]$ and those contained entirely in $R \doteq A[\ell+1..n]$. Since each subarray can be colored independently of the other, this suggests a divide-and-conquer approach to the problem. First, we will demonstrate a specific coloring that achieves the minimum number of colors, and then we will argue that this number of colors is in fact the minimum.

For our specific coloring, we will let $\ell$ be the middle index, so that both $L$ and $R$ have size exactly $(n-1)/2 = 2^{m-1} - 1$. Note that this means that both $L$ and $R$ can be colored in the exact same way, so our strategy is to first color $A[\ell]$ with a unique color before recursively applying our coloring scheme to $L$ and copying the resulting coloring of $L$ to $R$. The base case of this scheme occurs when $n = 0$, in which case we do nothing.

If we let $k^*(n)$ denote the number of colors this scheme requires for an array of size $n = 2^m - 1$, then the recurrence this gives is

$$k^*(2^m - 1) = 1 + k^*(2^{m-1} - 1),$$
$$k^*(0) = 0.$$

One can show by induction on $m$ that the solution to this recurrence is $k^*(n) = m = \log_2(n + 1)$.

Now, we will show that $\log_2(n + 1)$ colors is in fact the minimum number of colors that is achievable. From our initial discussion, we know that some entry $A[\ell]$ must be colored with a color that is unique to $A[1..n]$. At least one of the two arrays $L = A[1..\ell - 1]$ and $R = A[\ell + 1..n]$ must then be of size at least $(n-1)/2 = 2^{m-1} - 1$, and this array must be colored with some set of colors that does not include the one we colored $A[\ell]$ with. Since we need 0 colors to color an array of size 0, this gives us the recurrence

$$k(2^m - 1) \geq 1 + k(2^{m-1} - 1),$$
$$k(0) = 0.$$

Applying the same induction argument as in the $k^*$ solution then gives us that $k(n) \geq m = \log_2(n + 1)$.

Combining this with the observation that there is a coloring that uses exactly $m$ colors gives us that $k(n) = m = \log_2(n + 1)$, as desired.