CS381 **Homework 1 Sketch of Solution** **Fall 2012**

Question 1. $(\log\log n)^3$, $(\log n)^{0.2}$, $\sqrt{n}$, $\{n\log n, \log(n!)\}$, $n^{1.3}$, n^4, 2^n, $n!$

Some had trouble with $\log(n!)$: In such cases it often helps to "sandwich" the troublesome function (in this case $n!$) between two other functions, like this:

$$(n/2)^{(n/2)} < n! < n^n$$

Taking logarithms does not change the inequalities (because log is an increasing function) and gives:

$$(n/2)\log(n/2) < \log(n!) < n\log n$$

which shows that $\log(n!)$ has same order of growth as $n\log n$. (Another way of reaching the same conclusion is by using the Stirling approximation for $n!$)

Question 2.

1. At level i of the recursion tree, the problem size associated with a node is $n/3^i$, therefore the deepest level (i.e., the height h) corresponds to $n/3^h = 1$, which gives $h = \log_3 n$.

2. If the local work at a node is c_2n^2 then for its 6 children it is

$$6c_2(n/3)^2 = 6c_2n^2/9 = (2/3)c_2n^2$$

which is 2/3 of the parent's work. So the work done for level i is $(2/3)^i c_2n^2$.

3. The total work done over all levels is no more than:

$$c_2n^2(1 + (2/3) + (2/3)^2 + (2/3)^3 + \cdots) = c_2n^2(1/(1-(2/3))) = 3c_2n^2$$

and therefore $T(n)$ is $O(n^2)$.

Question 3.

1. At level i of the recursion tree, the largest problem size associated with a node is $n/2^i$, therefore the deepest level (i.e., the height h) corresponds to $n/2^h = 1$, which gives $h = \log n$.

2. At level i of the recursion tree, the smallest problem size associated with a node is $n/4^i$, therefore the shallowest-leaf level ℓ corresponds to $n/4^\ell = 1$, which gives $\log n = \ell\log 4 = 2\ell$, hence $\ell = (\log n)/2 = h/2$.

3. If the local work at a node is c_2n^2 then for its 10 children it is

$$2c_2(n/2)^2 + 8c_2(n/4)^2 = c_2n^2$$

which is same as for the parent. So the work done for any level that is $\leq \ell$ is exactly c_2n^2. For a level greater than ℓ the work is $\leq c_2n^2$.

4. The work for the first ℓ levels is $\ell c_2 n^2 = (c_2 n^2 \log n)/2$, and for all of the h levels it is no more than $hc_2 n^2 = c_2 n^2 \log n$. Therefore $T(n)$ is $O(n^2 \log n)$.

Question 4. The first recursive call is now on a set $\hat{S}$ of size $n/11$. The second recursive call is on a set that does *not* contain half of $\hat{S}$, and each of the elements of $\hat{S}$ so excluded causes 5 other elements from its group of 11 to also be excluded, i.e., the number of elements excluded from the second recursive call is at least

$$(|\hat{S}|/2)(1+5) = 6n/22 = 3n/11$$

and therefore the number of elements included in the second recursive call is no greater than

$$n - (3n/11) = 8n/11$$

This means the recurrence is $T(n) = c_1$ for small n (say, for $n \leq 20$), otherwise

$$T(n) \leq T(n/11) + T(8n/11) + c_2 n$$

whose solution is $O(n)$ because (as explained in class) the work drops from one level of the recursion tree to the next level by a factor of

$$(1/11) + (8/11) = 9/11 < 1$$

Question 5. We describe a general recursive algorithm $Majority(B, \rho)$, where B is any multiset of size m and ρ is any integer, that runs in time $O(m \log(m/\rho))$ and returns either an element that occurs more than ρ times in B, or a "not found" message (if B contains no such element). Before giving the details of that algorithm, we note that such an algorithm can be used to solve our problem by calling $Majority(A, n/k)$, which would run in time $O(n \log(n/(n/k))) = O(n \log k)$, as desired. So we focus on describing and analyzing how $Majority(B, \rho)$ works. Its steps are given below (where m is the size of B).

1. If $\rho \geq m$ then return "not found" (i.e., the answer for this B is "no element occurs more than ρ times in B"). Otherwise proceed to the steps that follow.

2. Use the linear-time selection algorithm to find the median (call it x) of the elements in B. Count how many times x occurs in B (say it occurs σ times).

3. If $\sigma > \rho$ then return x, otherwise do the following. Compute a set $B_<$ of elements of B that are less than x, and a set $B_>$ of elements of B that are greater than x. Observe that each of $B_<$ and $B_>$ has no more than $m/2$ elements (because x is the median of B).

4. Recursively call $Majority(B_<, \rho)$, and if it returns element y then return y as the answer. If on the other hand the call $Majority(B_<, \rho)$ returns a "not found", then call $Majority(B_>, \rho)$ and return the exact same answer that $Majority(B_>, \rho)$ returns (whether it is a "not found" or an element z of B).

The recurrence for the time complexity is

$T(m) = c_1$ if $m \le \rho$, and

$T(m) \le 2T(m/2) + c_2 m$ if $m > \rho$

where c_1 and c_2 are constants. This corresponds to a recursion tree that is binary, in which the total work for level i is cm, and the problem size associated with a node at level i is $m/2^i$. The height h of the recursion tree is the smallest value of i for which $m/2^i$ becomes $\le \rho$, i.e., h satisfies $m/2^{h-1} > \rho$ and $m/2^h \le \rho$. This gives

$$\rho/2 < m/2^h \le \rho$$

which implies that h is $O(\log(m/\rho))$, as desired.