

**Question 1. (20 points)** Rank the following functions by increasing order of growth (i.e., the slowest-growing first, the fastest-growing last):

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

where all the logarithms are to the base 2. If two functions have equal orders of growth then list them grouped together, e.g., between brackets {like this}.

**Question 2. (15 points)** The purpose of this question is to analyze, using the recursion tree method, an algorithm whose time complexity  $T(n)$  satisfies the following recurrence:

$T(1) = c_1$ , and if  $n > 1$  then  $T(n) = 6T(n/3) + c_2n^2$  where  $c_1$  and  $c_2$  are constants. We assume that  $n = 3^q$  for some integer  $q$ .

1. Derive an expression, as a function of  $n$ , for the height of the recursion tree (recall that the height of a tree is the largest number of parent-to-child links one goes through from the root to deepest leaf).
2. Write down an expression for the work associated with level  $i$  of the recursion tree (e.g., for level 0, which is the root, it is  $c_2n^2$ ).
3. Derive the “asymptotic order” of the solution for  $T(n)$  (i.e., its rate of growth as a function of  $n$ , not its exact value).

**Question 3. (20 points)** The purpose of this question is to analyze, using the recursion tree method, an algorithm whose time complexity  $T(n)$  satisfies the following recurrence:

$T(1) = c_1$ , and if  $n > 1$  then  $T(n) = 2T(n/2) + 8T(n/4) + c_2n^2$  where  $c_1$  and  $c_2$  are constants. We assume that  $n = 4^q$  for some integer  $q$ .

1. Derive an expression, as a function of  $n$ , for the height  $h$  of the recursion tree.
2. Derive an expression, as a function of  $n$ , for the depth  $\ell$  of the least-depth leaf in the recursion tree (i.e., leaf that is closest to the root). Also write down  $\ell$  as a function of  $h$ .
3. Write down an expression for the work associated with level  $i$  of the recursion tree for  $i \leq \ell$ .
4. Derive the “asymptotic order” of the solution for  $T(n)$  (i.e., its rate of growth as a function of  $n$ , not its exact value).

**Question 4. (20 points)** Suppose that, in the algorithm we explained in class for selecting the  $k$ th smallest element in a set of size  $n$ , we had partitioned the set  $S$  into  $n/11$  chunks of size 11 each (instead of  $n/5$  chunks of size 5 each). Analyze the modified algorithm, and give the recurrence relation governing its running time. What is the order of complexity of the solution to the recurrence? Briefly justify your answer.

**Question 5. (25 points)** Suppose that, given an  $n$ -element multiset  $A$  (not sorted), we want an  $O(n)$  time algorithm for determining whether  $A$  contains a *majority* element, i.e., an element that occurs more than  $n/2$  times in  $A$ . It is easy to solve this in  $O(n)$  time by using the linear-time selection algorithm by finding the median (call it  $x$ ), then counting how many times  $x$  occurs in  $A$  and returning it as the majority if the count exceeds  $n/2$  (otherwise the answer is “there is no majority”). Now consider the following generalization of the problem: Given  $A$  and an integer  $k < n$ , we want an algorithm that determines whether  $A$  contains a value that occurs more than  $n/k$  times in it (if many such values exist, then it is enough to find one of them). Design an algorithm for doing this, and analyze its complexity as a function of  $n$  and  $k$ . Your grade on this question will depend on how fast your algorithm is (of course it also has to be correct). Partial credit of 10 points is given for an  $O(kn)$  time algorithm, full credit is for an  $O(n \log k)$  time algorithm.

**Date due: Tuesday September 4, 2012**

**Assignment Project Exam Help**

**<https://tutorcs.com>**

**WeChat: cstutorcs**