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

$(\log \log n)^2$, $\log(n!)$, $\sqrt{n}$, $n!$, $n^{1.1}$, $n \log n$, 2^n, n^2, $(\log n)^{0.3}$

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 $T(n) = 3T(n/2) + c_2 n^2$ where c_1 and c_2 are constants. We assume that $n = 2^q$ 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 a 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 $c_2 n^2$).
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. (25 points) Design an $O(n \log n)$ time algorithm that, given an integer x and an array A of n distinct integers, determines whether x is the sum of two of the integers in A.

Question 4. (20 points) Suppose that, in the algorithm we explained in class for selecting the kth smallest element in a set of size n, we had partitioned the set S into $n/3$ chunks of size 3 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. (20 points) We covered in class an $O(n \log n)$ time divide-and-conquer algorithm for the maximum-subarray problem (also found in Section 4.1 of the textbook). The purpose of this question is to show that the divide-and-conquer would be faster if it solved a more general problem than merely returning the position and sum for the maximum subarray. Specifically, the call FIND-MAXIMUM-SUBARRAY($A, low, high$) is now supposed to return *all* of the following:

1. As before, the position and sum of the maximum subarray in the region of the array A between indices *low* and *high*: $M = \max_{low \leq i \leq j \leq high}(A[i] + \cdots + A[j])$, and the pair of indices i, j that achieve the max.
2. $L = \max_{low \leq l \leq high}(A[low] + \cdots + A[l])$ and the index l that achieves the max.
3. $R = \max_{low \leq r \leq high}(A[r] + \cdots + A[high])$ and the index r that achieves the max.

4. $W = A[low] + \cdots + A[high]$.

- (15 points) Show in detail how the answers from the left recursive call

  $$\text{FIND-MAXIMUM-SUBARRAY}(A, low, mid)$$

  and the answers from the right recursive call

  $$\text{FIND-MAXIMUM-SUBARRAY}(A, mid + 1, high)$$

  can be combined in constant time; here mid is the index of the middle, $mid = \lfloor (low + high)/2 \rfloor$. In other words, denoting the quantities returned by the left recursive call as $M', i', j', L', l', R', r', W'$, those returned by the right recursive call as $M'', i'', j'', L'', l'', R'', r'', W''$, you are supposed to explain how each of M, i, j, L, l, R, r, W is obtained in constant time from the values returned by the left and right recursive calls.

- (5 points) The above implies the following recurrence for the time: $T(1) = c_1$, and $T(n) = 2T(n/2) + c_2$ if $n > 1$, where c_1 and c_2 are constants. What is the order of the solution? Justify your answer (either using the recursion tree method, or algebraically).

Date due: Tuesday September 3, 2013