Preliminary: due February 11, 2014 8:00 pm

This question is an opportunity for you to check your understanding of the topics and practice writing formal solutions. This is a valuable learning opportunity – if you see that you’re at a loss, get help quickly!

Your goal should not be to get the right answer, but to convince the marker that you know what you’re doing. This question is marked on the following 3-point scale:

| 3: You’ve mastered this topic | 1: You don’t really know what you’re doing |
| 2: You’re almost there, but missing something | 0: You didn’t submit/had absolutely no clue |

This question must be completed INDIVIDUALLY.

Consider the following three methods of solving a particular problem (input size $n$):

1. You divide the problem into three subproblems, each $\frac{1}{3}$ the size of the original problem, solve each recursively, then combine the results in time linear in the original problem size.

2. You divide the problem into 16 subproblems, each $\frac{1}{4}$ of size of the original problem, solve each recursively, then combine the results in time quadratic in the original problem size.

3. You reduce the problem size by 1, solve the smaller problem recursively, then perform an extra “computation step” that requires linear time.

**UPDATE:** Assume the base case has size 1 for all three methods. For each method, write a recurrence capturing its worst-case runtime. Which of the three methods yields the fastest asymptotic runtime?

In your solution, you should use the Master Theorem wherever possible. In the case where the Master Theorem doesn’t apply, clearly state why not based on your recurrence, and show your work solving the recurrence using another method (no proofs required).
Challenges: due February 15, 2014 noon

Answer each question completely, always justifying your claims and reasoning. Your solution will not just be graded on correctness, but on its clarity as well. Technically correct answers that are hard to understand will not receive full marks. Mark values for each question are contained in the [square brackets].

You may work in groups of up to THREE to complete these questions.

1. Recall the recurrence for the worstcase runtime of quicksort from lecture:

   \[ T(n) = \begin{cases} 
   c, & \text{if } n \leq 1 \\
   T(|L|) + T(|G|) + dn, & \text{if } n > 1 
   \end{cases} \]

   where \( L \) and \( G \) are the partitions of the list. Clearly, how the list is partitioned matters a great deal for the runtime of quicksort.

   (a) \([4]\) Suppose the lists are always evenly split; that is, \(|L| = |G| = \frac{n}{2}\) at each recursive call. (For simplicity, we’ll ignore the fact that each list really would have size \(\frac{n-1}{2}\).)

      Find a tight asymptotic bound on the runtime of quicksort using this assumption.

   (b) \([4]\) Now suppose that the lists are always very unevenly split: \(|L| = n - 2\) and \(|G| = 1\) at each recursive call. Find a tight asymptotic bound on the runtime of quicksort using this assumption. For simplicity, you may assume that \(n\) is even.

2. The CSC165 Course Notes presents two algorithms for computing the maximum subsequence sum in a list (see pages 61-63), which run in times \(\Theta(n^3)\) and \(\Theta(n^2)\). Devise a divide-and-conquer algorithm for this problem that runs in time \(O(n \log n)\) (or better). You must do the following things for this question:

   (a) \([7]\) Write clear pseudocode for your algorithm, including comments.

      Also give a few statements of informal justification about why your code is correct.

      Note: we won’t be “running” your pseudocode, but it is acceptable to actually implement your code in Python, and then copy it into your solution.

   (b) \([5]\) Fully analyse the runtime of your algorithm; you may use the Master Theorem.

      For full marks, your algorithm must use a divide-and-conquer approach; it is highly recommended you split up the list into two halves (as in mergesort).

**Programming Question(s): purely for your benefit**

Occasionally, we will suggest programming exercises for you to try to reinforce the concepts presented in lecture and on the problem set. These are not to be handed in, but we think they can be very beneficial to your own learning (not to mention valuable coding practice). Feel free to discuss these questions with your peers and the course staff throughout the term!

1. The obvious thing to do is to implement your algorithm for max subsequence sum. The next obvious thing to do is identify why it is still inefficient, performing repeated calculations. Then, try to do better! A recursive approach that recurses on \(A[0..n - 2]\) and combines the result with \(A[n - 1]\) can do this in linear time, as can the equivalent iterative approach.

2. Since the quicksort algorithm depends so much on the choice of pivot, you might want to experiment with different ways of better selecting the pivot.

3. Another optimization made to quicksort is to switch to insertion sort when the list size is below 10. Try to implement this, and see if it really improves the runtime.