CS 173: Discrete Structures, Fall 2009
Homework 8 Solutions

This homework contains 3 problems worth a total of 30 regular points.

1. **More on recurrences [10 points]**

   In the discussion of Karatsuba's algorithm for multiplying integers (end of lecture 25), I left out some details of the analysis. Let's fill in some of them:

   (a) I claimed that if T has the following recurrence (where c and d are constants)

   $$T(1) = c$$
   $$T(n) = 4T(n/2) + dn$$

   then T is $O(n^2)$. Show that this is true by unrolling the recurrence, assuming that n is a power of two.

   (b) I claimed that $n(\frac{3}{2})^{log_2 n}$ is $O(n^{log_2 3})$. Show that this is correct. Hint: use algebra and standard properties of logs and exponentials.

   Solution:

   (a) Lets assume $n = 2^k$.

$$\begin{aligned}
T(n) = T(2^k) &= 4T(2^k/2) + dn \\
&= 4(4T(2^k/2^2) + d(2^k/2)) + dn \\
&= 4^2T(2^k/2^2) + 2d2^k + d2^k \\
&= 4^2(4T(2^k/2^3) + d2^k/2^2) + 2d2^k + d2^k \\
&= 4^3T(2^k/2^3) + 2^2d2^k + 2d2^k + d2^k \\
&= \dots \\
&= 4^kT(2^k/2^k) + 2^{k-1}d2^k + 2^{k-2}d2^k + \cdots + d2^k \\
&= 4^kT(1) + d2^k(2^{k-1} + 2^{k-2} + \cdots + 1) \\
&= 4^kc + d2^k(2^k - 1) \\
&= (2^k)^2c + d2^k(2^k - 1) \\
&= cn^2 + dn^2 - dn \\
&= O(n^2)
\end{aligned}$$

   (b)

$$n(\frac{3}{2})^{log_2 n} = n(\frac{3^{log_2 n}}{2^{log_2 n}}) = n(\frac{3^{log_3 n \cdot log_2 3}}{n}) = 3^{log_3(n^{log_2 3})} = n^{log_2 3}$$

   In the second and third equalities we are using the rules $log_a b = log_c b \cdot log_a c$ and $a \cdot log_b c = log_b(c^a)$ for logarithms, respectively.

2. **Algorithm analysis [10 points]**

Consider the following mystery function:

1. foo ($a_1, a_2, \ldots a_n$: real numbers)
    2. D = $|a_1 - a_2|$
    3. for $x = 1$ to n
        4. for $y = 1$ to n
            5. Q = $|a_x - a_y|$
            6. if ($x \neq y$ and $Q < D$) D = Q

    return D

(a) Give a brief English description of what the function foo computes.

(b) How many times are lines 5 and 6 executed, as a function of n?

(c) What is the big-O running time of this algorithm? Briefly justify your answer.

(d) This is a really bad algorithm for this task. Write pseudocode for a function with a better big-O running time. Hint: your new function can call any of the standard algorithms we've seen in lecture e.g. binary search.

(e) What is the big-O running time of your new function from part d? Hint: if your new function called a standard library function, you need to include its cost in your big-O analysis. For example, a call to binary search requires $O(\log n)$ time, where n is the size of the array you're feeding to binary search.

Solution:

(a) It outputs the difference of the two closest numbers in the list.

(b) Lines 5 and 6 are executed n^2 number of times, because they are inside two nested loops, each of which iterates n times.

(c) It is $O(n^2)$. The algorithm contains two nested loops each counts up to n. The required time for the operations inside the loops is $O(1)$. So the total running time is $O(n^2)$.

(d) 1. foo ($a_1, a_2, \ldots a_n$: real numbers)
    2. MergeSort ($a_1, a_2, \ldots a_n$)
    3. D = $a_2 - a_1$
    4. for $x = 2$ to $n - 1$
        5. Q = $a_{x+1} - a_x$
        6. if ($Q < D$) D = Q
    7. return D

(e) It is $O(n \log n)$. We need $O(n \log n)$ to run the MergSort algorithm plus linear time to find the minimum value.

3. **Son of algorithm analysis [10 points]**

A particularly irritating student in the beginning programming class submitted the following function. (The max function returns the largest of the numbers given to it.)

1. bar($a_1, a_2, \ldots a_n$: real numbers)
    2. if (n = 1) then return 0
    3. else
        4. L = bar(a_2,a_3,. . . ,a_n)
        5. R = bar(a_1,a_2,. . . ,a_{n-1})
        6. Q = $|a_1 - a_n|$
        7. return max(L,R,Q)

(a) Give a succinct English description of what bar computes.

(b) Suppose that $T(n)$ is the running time for bar on the input n. Write a recurrence (with initial condition!) for $T(n)$.

(c) Draw a recursion tree for $T(n)$.

(d) Use the tree to derive a big-O solution for $T(n)$.

(e) Briefly explain why this isn't a good design and how you would write a more efficient algorithm for this task.

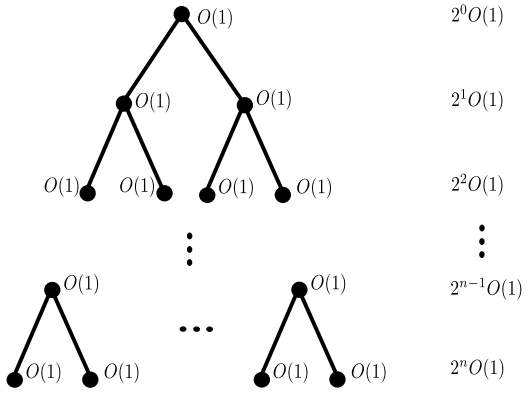

Figure 1: Problem 3(c)

Solution:

(a) It outputs the difference between the two numbers in the list that are furthest apart.

(b) $T(n) = 2T(n-1) + O(1)$ and $T(1) = O(1)$.

(c) See Figure 1.

(d) The total work at level i of the tree is $2^i O(1)$. The height of the tree is n. So the total running time is $O(1) \cdot \sum_{i=1}^{n} 2^i = O(2^n)$.

(e) There is a huge amount of redundant work that the algorithm is doing because of the overlapping subproblems. Observe that the algorithm computes the difference of any two numbers a_i and a_j $(2 \leq i, j \leq n-1)$ in both of the recursive calls.

One better algorithm would be to sort the list and then subtract the first value from the last value, similar to the solution for problem 2d. This would take $O(n \log n)$ time. This is a big improvement over the original algorithm (and a fairly reasonable answer for you to have given).

But we can do better. Here is a linear algorithm:

1. foo ($a_1, a_2, \ldots a_n$: real numbers)
    2. return (Maximum($a_1, a_2, \ldots a_n$) - Minimum($a_1, a_2, \ldots a_n$))

You can find the maximum of a list of numbers, or the minimum, with one linear-time scan through the list.