Homework 8

Due Wednesday, November 4 at 11:59 PM

Homework Guidelines

Collaboration policy Collaboration on homework problems, with the exception of programming assignments and reading quizzes, is permitted, but not encouraged. If you choose to collaborate on some problems, you are allowed to discuss each problem with at most 5 other students currently enrolled in the class. Before working with others on a problem, you should think about it yourself for at least 45 minutes. Finding answers to problems on the Web or from other outside sources (these include anyone not enrolled in the class) is strictly forbidden.

You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. You must also identify your collaborators. If you did not work with anyone, you should write "Collaborators: none." It is a violation of this policy to submit a problem solution that you cannot orally explain to an instructor or TA.

Solution guidelines For problems that require you to provide an algorithm, you must give the following:

1. a precise description of the algorithm in English and, if helpful, pseudocode,
2. a proof of correctness,
3. an analysis of running time and space.

You may use algorithms from class as subroutines. You may also use any facts that we proved in class.

You should be as clear and concise as possible in your write-up of solutions.

A simple, direct analysis is worth more points than a convoluted one, both because it is simpler and less prone to error and because it is easier to read and understand. Points might be subtracted for illegible handwriting and for solutions that are too long. Incorrect solutions will get from 0 to 30% of the grade, depending on how far they are from a working solution. Correct solutions with possibly minor flaws will get 70 to 100%, depending on the flaws and clarity of the write up.

1. **(Recurrences)**

   Consider the following algorithm A:

```
def A(n):
    if n == 0:
        return 0
    if n is even:
        return A(n/2)
    else:
        return A(2(n-1)) + 1
```

   (a) Prove that the A terminates when its input n is a nonnegative integer.

   (b) Write a recurrence for the running time (in terms of n) of A.

   (c) Give a closed form for its asymptotic running time (using whichever method you like)

   (d) If we look at the tree of recursive calls to A made by this algorithm, what is its depth (asymptotically, in terms of n)? How many leaves does it have (asymptotically, in terms of n)?

   (e) What function of n does A compute? It's easy to express it in terms of the binary expansion of n (i.e. the bits you get when you write n in base 2).

   (a) To prove that A terminates when its input of n is a non-negative integer, we will examine the three cases individually. We hope to show that in each case the input n can only decrease in size and eventually will hit the base case.

   - Case 1: $n == 0$
     If $n == 0$, the algorithm simply returns zero and will terminate.
   - Case 2: The input n is even
     If the input is even, the algorithm A is called recursively on $\frac{n}{2}$ so the input is cut in half.
   - Case 3: The input n is odd
     If the input is odd we see that A returns $A(2(n-1)) + 1$. Here it is less clear that n should always decrease in a series of calls. However, we can continue from this case to see if we can discover anything about the subsequent calls.
     - First we recognize that for any odd number n, $n-1$ is even.
     - Since the next call will be on an even number, we know that case 2 will be true. We can then substitute the original $2(n-1)$ into case 2.

$$A(\frac{2(n-1)}{2}) = A(n-1).$$

     - From the call above, we see that the passed value resolves to $n-1$. Since we know $n-1$ is even, the next call must be:

$$A\left(\frac{n-1}{2}\right).$$

From these points we see that after 3 calls to A consisting of constant-time operations we arrive at a point where the input is once again halved. We finally must recognize that if any even number is halved and any odd number is decremented by one and then halved we must eventually arrive at $n = 1$. This input would call case 3: $A(2(1-1))+1$ in which the recursive call reaches the base case of $n = 0$.

If A is always taking constant time steps to half the input and it will eventually reach the base case, the algorithm must terminate.

(b) Since we know constant time is used to get to another halving of the input from the substitution performed in part a, the recurrence can be written as follows:

$$T(n) \leq \begin{cases} T\left(\frac{n}{2}\right) + O(1) & \text{if } n \text{ is odd} \\ T\left(\frac{n-1}{2}\right) + O(1) & \text{if } n \text{ is even} \end{cases}$$

To make the analysis easier, we can use just the even case since the even case will always leave a larger input than the odd case. The result of any bounding derived from the even case will therefore contain the odd case.

$$T(n) \leq T\left(\frac{n}{2}\right) + O(1).$$

(c) Using the master's method, we will give the asymptotic running time. $f(n) = O(1)$ so $c = 0 = \log_2 1$ giving case 2 as the correct master's formula. Simply, the running time is then $O(\log n)$.

(d) The depth of the tree made by this algorithm asymptotically is $\log n$. At each step, $O(1)$ calculations are performed to reduce the size of the input by a factor of 2. To reach the base-case of zero, the input must be reduced by the factor of two $\log n$ times. This algorithm only produces one leaf for each recursive call - we can see this because only one call to A exists in each conditional statement. Because of this characteristic, the number of leaves should be asymptotically equivalent to the depth of the tree, $\log n$.

(e) The algorithm returns the number of "on" bits that are needed to record n in memory.

2. **(More recurrences)**

Suppose you are choosing between the following three algorithms, all of which have O(1) base cases for size 1:

(a) Algorithm A solves problems of size n by dividing them into five subproblems of size $n/2$, recursively solving each subproblem, and then combining the solutions in linear time.

(b) Algorithm B solves problems of size n by recursively solving one subproblem of size $n/2$, one subproblem of size $2n/3$, and one subproblem of size $3n/4$ and then combining the solutions in linear time.

Hint: Approach this algorithm via the substitution method (pp. 211-217) to avoid tough summations. Also, solving that one as $O(n^d)$ for the smallest valid integer d is all we're looking for.

(c) Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in asymptotic notation) and which would you choose? You may use the Master Method.

(a) Algorithm A using the Master Method:
The recurrence for algorithm A is as follows:

$$T(n) \leq 5T\left(\frac{n}{2}\right) + cn.$$

This gives $f(n) = O(n) \Rightarrow c = 1$ which is less than $\log_2 5 = 2.323$. Using the corresponding formula we get $n^{\log_2 5}$ which can be expressed with big-O and big-omega notation as follows:

$$O\left(n^{2.323}\right) \text{ and } \Omega\left(n^{2.322}\right).$$

(b) Algorithm B using substitution:
First we write the recurrence:

$$T(n) \leq T\left(\frac{n}{2}\right) + T\left(\frac{2n}{3}\right) + T\left(\frac{3n}{4}\right) + cn.$$

We assume the solution takes the form an^d and plug this into the recurrence.

$$T(n) \leq a\left(\frac{1}{2}\right)^d n^d + a\left(\frac{2}{3}\right)^d n^d + a\left(\frac{3}{4}\right)^d n^d + cn.$$

$$T(n) \leq an^d\left(\left(\frac{1}{2}\right)^d + \left(\frac{2}{3}\right)^d + \left(\frac{3}{4}\right)^d\right) + cn.$$

To solve the recurrence, we must find where the term attached to an^d is less than 1. Using mathematical software, we receive the roots of that equation. We can then state the boundaries of the asymptotic running time - the n^d term will dominate.

$$O\left(n^{2.57}\right) \text{ and } \Omega\left(n^{2.56}\right).$$

(c) Algorithm C using the Master Method:
The recurrence for algorithm C is as follows:

$$T(n) \leq 9T\left(\frac{n}{3}\right) + cn.$$

This gives $f(n) = cn^2 \Rightarrow c = 2$ which is equal to $\log_3 9 = 2$. This leads us to use the second case of the Masters Method:

$$\Theta\left(n^{\log_3 9} \log n\right) = \Theta\left(n^2 \log n\right).$$

This tells us the proffered algorithm is algorithm C since:

$$\Theta_c\left(n^2 \log n\right) < O_b\left(2.57\right) < O_a\left(2.323\right).$$