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.

Collaborators: None

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).

ANSWER

a) Regarding the input, there can be three different cases, when $n < 0$, when $n > 0$, and finally $n = 0$. For this final case, the algorithm will return 0, which is a non-negative integer, for the other two we need to go a bit deeper.

Both $n > 0$ and $n < 0$ will fall in one of two cases, either the number is even, and if so the algorithm will recursively call itself on the half of n, which means that, regardless of it being positive or negative, it'll be closer to 0. Or the number is odd, in which case the algorithm will call itself recursively on the double of $n - 1$, and return that plus one.

This ever growing trend towards 0, mean that the algorithm will always end up hitting it's base case, and return 0 plus whatever else is up the tree. What is up the tree is ALWAYS positive, since it'll either be what was returned before, or that plus one, meaning it never goes below the base case of 0.

b) $T(n) = \begin{cases} 2, & \text{if } n = 1 \\ (n\%2) \times T(2(n-1)) + (|n\%2 - 1|) \times T(n/2) + 1, & \text{for all other cases} \end{cases}$

c) The closed form asymptotic running time is $T(n) \leq O(\log_2 n \times 2 + 5)$.

This is true for our base case: $T(1) = 2$, and $\log_2 1 \times 2 + 5 = 5$, so $T(1) \leq O(\log_2 n \times 2 + 5)$.

If we assume that for all $1 \leq j \leq n-1$ we have $T(j) \leq O(\log_2 j \times 2 + 2)$, then for n we have either:

$T(n) = 0 \times T(2(n-1)) + 1 \times T(n/2) + 1 = T(n/2) + 1 = \log_2 n/2 \times 2 + 3 = \log_2 n \times -1 \times 2 + 3 = \log_2 n \times -2 + 3 \leq O(\log_2 j \times 2 + 5)$

OR

$T(n) = 1 \times T(2(n-1)) + 0 \times T(n/2) + 1 = T(2(n-1)) + 1 = \log_2 2(n-1) \times 2 + 3 = \log_2 2n - 2 \times 2 + 3 = \log_2 n - 1 \times 2 + 5 = \log_2 j \times 2 + 5 \leq O(\log_2 n \times 2 + 5)$

d) The Depth of the tree would be the $log_2 n$. As such, the number of leaf nodes is $2^{log_2 n}$ or rather n. Should be noted here that every branch will continue branching out until it reaches the base case, so every branch will end with a leaf in level $log_2 n$.

e) Function A is computing the number of bits equal to "1" in the binary representation of n.

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.

ANSWER

a) Problem (a), as per the master theorem, we have a recurrence runtime of $T(n) = 5 \times T(n/2) + n$ with a base case $T(1) = 1$, which means it has an asymptotic runtime of $O(n^{log_2 5})$

b) For problem (b), we have a recurrence runtime of $T(n) = T(n/2) + T(\frac{2n}{3}) + T(\frac{3n}{4}) + n$. This results in an asymptotic runtime of $O(n^3)$.

Base case $T(1) = 1 \leq O(1)$
If we now assume that that $T(j) \leq O(j^3)$ for any j, $1 \leq j \leq n-1$, then for n we have that:

$T(n) = T(n/2) + T(\frac{2n}{3}) + T(\frac{3n}{4}) + n = (\frac{n}{2})^3 + (\frac{2n}{3})^3 + (\frac{3n}{4})^3 + n = \frac{n^2}{8} + \frac{8n^2}{27} + \frac{27n^2}{64} + n = n^3(\frac{216}{1728} + \frac{512}{1728} + \frac{729}{1728}) + n = \frac{1457}{1728} \times n^3 + n = 0.84 \times n^3 + n \leq O(n^3)$

c) Problem (c), as per the master theorem, we have a recurrence runtime of $T(n) = 9 \times T(n/3) + n^2$ with a base case $T(1) = 1$, which means it has an asymptotic runtime of $O(n^2)$

With that in mind, and choosing purely on the basis of runtime, I'd pick algorithm c.