Solution 5: Divide & Conquer

Problem 1. The recurrence $T(n) = 7T(n/2) + n^2$ describes the running time of an algorithm A. A competing algorithm A' has a running time of $T'(n) = aT'(n/4) + n^2$. What is the largest integer value for a such that A' is asymptotically faster than A?

Solution. The answer is $a = 48$. By Master Theorem, we know that we land in Case 1 for the first recurrence, so $T(n) = \Theta(n^{\log_2 7})$. For the second recurrence, we consider each of the 3 cases:

1. We land in case 1 if $2 < \log_4 a \iff a > 16$. Then, $T'(n) = \Theta(n^{\log_4 a})$. Thus, to be faster, we require

   $$\log_4 a < \log_2 7 \iff a < 49$$

   so the largest integer satisfying this case is $a = 48$ (which is consistent with the constraint $a > 16$.

2. To land in case 2, we require $a = 16$. Since this is less than the solution we found in case 1, this case cannot give us our answer.

3. Again, to land in case 3, we need $a < 16$ but this is less than the solution found in case 1.

Having exhausted all the cases, we are done. □

Problem 2. The Euclidean algorithm is a method for computing the greatest common divisor (gcd) of two numbers, taking advantage of the fact

$$\gcd(a,b) = \gcd(b, a-b)$$

for positive integers a, b satisfying $a \geq b$. Consider the following variation:

$$\gcd(a,b) = \begin{cases} 2\gcd(a/2, b/2) & \text{if } a, b \text{ even} \\ \gcd(a, b/2) & \text{if } a \text{ odd } b \text{ even} \\ \gcd((a-b)/2, b) & \text{if } a, b \text{ odd} \end{cases}$$

a. Prove the variation is correct.

b. Provide an algorithm that uses the variation to compute the greatest common divisor of two numbers a, b in $O(\log(ab))$

Solution. (a). We interpret $\gcd(a,b) = d$ as the largest integer so that if $a = dx$ and $b = dy$, then, $\gcd(x,y) = 1$. Handle each of the three cases separately:

1. If d is not even, then x and y must be even, contradicting $\gcd(x,y) = 1$. Therefore, we can write $a/2 = (d/2)x$ and $b/2 = (d/2)y$. This means that $\gcd(a/2, b/2) = d/2 \implies \gcd(a,b) = 2\gcd(a/2, b/2)$.
2. If d is even, then this means $d \mid a \implies 2 \mid a$, contradiction. Thus, d is odd and we can write $b/2 = d(y/2)$. It remains to verify that $\gcd(x, y/2) = 1$ given that $\gcd(x,y) = 1$, but this is true because x, y share no common factors, so removing a factor of 2 from y won't change that fact. Thus, $\gcd(a, b/2) = d = \gcd(a,b)$.
3. We use the standard Euclidean algorithm to get $\gcd(a,b) = \gcd(a-b, b)$. Note that we now land in case 2 since $a-b$ even and b is odd, which immediately gives $\gcd(a-b, b) = \gcd((a-b)/2, b) \implies \gcd(a,b) = \gcd((a-b)/2, b)$.

(b). The algorithm is to run the variant Euclidean algorithm until one of a, b are 0 or 1: in the former case, return the nonzero value. Otherwise, return 1. A bound on the runtime is to note that one of a, b are halved at each step. Since we run the algorithm until at least one of the values is ≤ 1, it will take in the worst case $\log_2 a + \log_2 b = \log(ab)$ steps. This shows the runtime is $O(\log(ab))$.

□

Problem 3. Given an n-bit binary integer, design a divide-and-conquer algorithm to convert it into its decimal representation. For simplicity, you may assume that n is a power of 2.

1. Provide a succinct (but clear) description of your algorithm, including pseudocode.
2. Prove the correctness of your algorithm.
3. Analyze the running time of your algorithm. Assume that it is possible to multiply two decimal integers numbers with at most m digits in $O(m^{\log_2 3})$ time.

Hint: An n-bit binary integer x can be expressed as $x = (x_{n-1}, x_{n-2}, \cdots, x_1, x_0)_2$ where $x_i \in \{0, 1\}$. Let $x_\ell = (x_{n/2-1}, x_{n/2-2}, \cdots, x_1, x_0)_2$ be the $(n/2)$-bit binary integer corresponding to the $(n/2)$ least significant digits of x. Let $x_m = (x_{n-1}, x_{n-2}, \cdots, x_{n/2+1}, x_{n/2})_2$ be the $(n/2)$-bit binary integer representing the $(n/2)$ most significant digits of x. Then, $x = x_\ell + 2^{n/2} \cdot x_m$. This should suggest us a way to set up a divide and conquer strategy... :) Careful about the number of subproblems!

Solution.

1. Using the hint, the idea is to split the n-bit integer into the first half and second half: call these $n/2$-bit halves x and y, respectively. Then, we want to compute $2^{n/2}x + y$. Continue calling the algorithm on x, y until they are of 1-bit each, at which point we return the value itself.

2. Clearly the base cases of length 1 work, since $0_2 = 0$ and $1_2 = 1$. It suffices to show that $x = 2^{n/2}x_l + x_r$ is correct, where x_l, x_r are as defined in the hint. Indeed, notice that

$$\begin{aligned} x &= (x_n, x_{n-1}, \cdots, x_1)_2 \\ &= (x_n, \cdots, x_{n/2+1}, 0, \cdots, 0)_2 + (x_{n/2}, \cdots, x_1)_2 \\ &= (x_n, \cdots, x_{n/2} + 1, 0, \cdots, 0)_2 + x_r \\ &= 2^{n/2}(x_n, \cdots, x_{n/2+1})_2 + x_r \\ &= 2^{n/2}x_l + x_r \end{aligned}$$

   since appending a zero to the end of a binary integer is equivalent to multiplying by 2 in decimal and there are $n/2$ zeros. Thus, the algorithm properly handles base cases and correctly combines the results from splitting.

3. Let $T(n)$ denote the number of operations needed for an $n-$bit binary integer. I claim that

$$T(n) = 2T(n/2) + O(n^{\log_2 3})$$

   After splitting, the conversion of x_l and x_l into decimal clearly take $T(n/2)$ each, yielding the $2T(n/2)$ term. As for the combine step, it suffices to determine the runtime of multiplying $2^{n/2}$ by x_l, since addition is done in linear time, $O(n)$.

   To compute $2^{n/2}$, we can, say, repeatedly square starting at 2. This requires squaring $\log_2(n/2) = O(\log n)$ times. Squaring is at worst multiplying two $n/4$-bit integers (in decimal). In decimal, we have $\log_{10}(2^{n/4}) = n/4 \cdot \log_{10}(2) = O(n)$ digits, so multiplication takes

$O(n^{\log_2 3})$ time. We do this for $n/8$, $n/16$, etc, so the runtime is

$$\begin{aligned} O(n^{\log_2 3} + (n/2)^{\log_2 3} + \cdots + 1) &= O(n^{\log_2 3} + \frac{1}{3}n^{\log_2 3} + \frac{1}{9}n^{\log_2 3} + \cdots) \\ &= O\left(\frac{1}{1 - 1/3}n^{\log_2 3}\right) \\ &= O(n^{\log_2 3}) \end{aligned}$$

It remains to multiply $2^{n/2}$ and x_l. However, both are $n/2$-bit integers, meaning the number of digits in the decimal representation of x_l and $2^{n/2}$ is

$$O(\log_{10}(2^{n/2})) = O(n)$$

Multiplying two decimal integers with at most m digits takes $O(m^{\log_2 3})$ time, and since each of x_l and $2^{n/2}$ have at most $O(n)$ digits, the multiplication takes $O(n^{\log_2 3})$ time. The recurrence relation becomes

$$T(n) = 2T(n/2) + O(n^{\log_2 3}) + O(n^{\log_2 3}) = 2T(n/2) + O(n^{\log_2 3})$$

We apply the Master theorem. Since $\log_2(3) > \log_2(2) = 1$, we have an instance of Case 3. Indeed, setting $\delta = 2/3$,

$$\delta n^{\log_2 3} = \frac{2}{3}n^{\log_2 3} = 2\frac{n^{\log_2 3}}{2^{\log_2 3}} = 2(n/2)^{\log_2 3} = 2f(n/2)$$

as desired. By Master theorem, then, $T(n) = \Theta(n^{\log_2 3})$, which is our overall runtime.

□