CS-UY 2413: Design & Analysis of Algorithms

Prof. Lisa Hellerstein

Fall 2024
New York University

Homework 2

Due 11:59pm Monday, Sep 30, New York time.

By handing in the homework you are agreeing to the Homework Rules; see EdStem.

Our Master Theorem: The version of the Master Theorem that we covered in class is on the last page of this homework. We won't be covering the version of the Master Theorem in the textbook and you're not responsible for knowing it. (But you may find it interesting!)

Reminder: For $r \neq 1$, $r^0 + r^1 + \cdots + r^k = \frac{r^{k+1}-1}{r-1}$.

1. For each example, indicate whether $f = o(g)$ (little-oh), $f = \omega(g)$ (little-omega), or $f = \Theta(g)$ (big-Theta). No justification is necessary.

   (a) $f(n) = 2n^2 + 5n$, $g(n) = n^2$

   (b) $f(n) = \log_2 n$, $g(n) = n^{0.1}$

   (c) $f(n) = n^3$, $g(n) = 2^n$

   (d) $f(n) = 7\sum_{i=0}^{n} 2^i$, $g(n) = 2^n$

   (e) $f(n) = \log_2 n^2$, $g(n) = \log_4 n$

2. Give a formal proof of the following statement: If $f(n) \geq 1$ for all $n \in N$, $g(n) \geq 1$ for all $n \in N$, $f(n) = O(g(n))$, and $g(n)$ is unbounded (meaning $\lim_{n\to\infty} g(n) = \infty$) then $\sqrt{f(n)} = O(\sqrt{g(n)})$. Use the formal definition of big-Oh in your answer. In your proof, you can use the fact that the value of $\sqrt{n}$ increases as n increases.

3. For each of the following recurrences, determine whether Our Master Theorem (on the last page of this HW) can be applied to the recurrence. If it can, use it to give the solution to the recurrence in Θ notation; no need to give any details. If not, write "Our Master Theorem does not apply."

   (a) $T(n) = 2T(n/2) + n \log n$

   (b) $T(n) = 9T(n/3) + n^2$

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

(d) $T(n) = 4T(n/2) + n^2$

4. Our Master Theorem can be applied to a recurrence of the form $T(n) = aT(n/b) + n^d$, where a, b, d are constants with $a > 0, b > 1, d > 0$. Consider instead a recurrence of the form $T_{new}(n) = aT_{new}(n/b) + n \log_2 n$ where $a > 0, b > 1$. For each of the following, state whether the given property of T_{new} is true. If so, explain why it is true. If not, explain why it is not true. (Even if you know the version of the Master Theorem in the textbook, don't use it in your explanation.)

   (a) $T_{new}(n) = O(n^2)$ if $\log_b a = 2$

   (b) $T_{new}(n) = \Omega(n \log_2 n)$

5. Consider the recurrence $T(n) = 2T(n/2) + n^2$ for $n > 1$, and $T(1) = 1$.

   (a) Compute the value of $T(4)$, using the recurrence. Show your work.

   (b) Use a recursion tree to solve the recurrence and get a closed-form expression for $T(n)$, when n is a power of 2. (Check that your expression is correct by plugging in $n = 4$ and comparing with your answer to (a).)

   (c) Suppose that the base case is $T(2) = 5$, instead of $T(1) = 1$. What is the solution to the recurrence in this case, for $n \geq 2$?

6. Consider a variation of mergesort that works as follows: If the array has size 1, return. Otherwise, divide the array into quarters, rather than in half. Recursively sort each quarter using this variation of mergesort. Then merge the first two quarters. Then merge the result with the last two quarters.

   (a) Write a recurrence for the running time of this variation of mergesort. It should be similar to the recurrence for ordinary mergesort. Assume n is a power of 4.

   (b) Apply Our Master Theorem to the recurrence to get the running time of the algorithm, in theta notation. Show your work.

7. Consider the following recursive algorithm. Assume n is a power of 2.

   - If the array has only one element, return.
   - Recursively process the first half of the array.
   - Recursively process the second half of the array.
   - Recursively process the first half of the array again.

   (a) Suppose the algorithm's task is to set every element in the array to 0. Is the algorithm correct? Explain.

   (b) Write a recurrence expressing the running time of the algorithm.

(c) Apply Our Master Theorem to your recurrence. What is the running time of the algorithm, in theta notation?

8. Consider a recursive algorithm that, given an array of size n (a power of 3), divides it into three subarrays of size $n/3$, recursively processes each subarray, and then performs a linear-time operation on the entire array. The base case is $T(1) = 1$. The recurrence is $T(n) = 3T(n/3) + n$.

   (a) Solve this recurrence using the Master Theorem.
   (b) What is the runtime in Big-O notation?

9. Design a recursive algorithm that finds the maximum element in an array of size n (assume n is a power of 2). The algorithm divides the array into two halves, recursively finds the maximum in each half, and then compares the two results to find the overall maximum. Write the recurrence relation for this algorithm's runtime and solve using the Master Theorem.

10. Suppose we have a recursive algorithm that solves a problem of size n by recursively solving two subproblems of size n/2, and then performing O(n log n) work to combine the solutions. Write down the recurrence relation for the running time T(n). Solve this recurrence using the Master Theorem.

Theorem 0.1 (Our Master Theorem)

Let a, b, d, n_0 be constants such that $a > 0$, $b > 1$, $d \geq 0$ and $n_0 > 0$. Let $T(n) = aT(n/b) + \Theta(n^d)$ for when $n \geq n_0$, and $T(n) = \Theta(1)$ when $0 \leq n < n_0$. Then,

$$T(n) = \begin{cases} \Theta(n^d \log n) & \text{if } d = \log_b a \\ \Theta(n^{\log_b a}) & \text{if } d < \log_b a \\ \Theta(n^d) & \text{if } d > \log_b a \end{cases}$$

We assume here that $T(n)$ is a function defined on the natural numbers. We use $aT(n/b)$ to mean $a'T(\lfloor n/b \rfloor) + a''T(\lceil n/b \rceil)$ where $a', a'' > 0$ such that $a' + a'' = a$.