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) $f(n) = n^2 \log n$, $g(n) = n^3$
   (b) $f(n) = n^2$, $g(n) = n \log n$
   (c) $f(n) = 2^n$, $g(n) = n^2$
   (d) $f(n) = \sum_{i=1}^{n} i^2$, $g(n) = n^3$
   (e) $f(n) = \log_2 n$, $g(n) = \log_{16} n$

2. Give a formal proof of the following statement: If $f(n) \geq 1$ for all $n \in \mathbb{N}$, $g(n) \geq 1$ for all $n \in \mathbb{N}$, $f(n) = O(g(n))$, 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) $T(n) = T(n/2) + 1$

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 $T_{new}(n) = aT_{new}(n/b) + n \log_d n$ where $a > 0$, $b > 1$, $d > 1$ (and $T(1) = 1$). 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) $T_{new}(n) = O(n \log^2 n)$ if $\log_b a = 1$

   (b) $T_{new}(n) = \Omega(n^{\log_b a} \log n)$

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 fourths, rather than in half. Recursively sort each fourth using this variation of mergesort. Then merge the first (leftmost) fourth with the second fourth. Then merge the result with the third fourth, and finally merge with the last fourth.

   (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 sorting algorithm. Assume n is a power of 2. (Note: This is not a version of mergesort. No merges are performed.)

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

   (a) Prove that the algorithm is correct by showing that the array will be sorted after the three recursive calls are performed, assuming the three recursive calls correctly sort their (sub)arrays. Note that the middle element of the array is included in each of the 3 recursive calls.

(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. Let's analyze a different divide-and-conquer algorithm. The algorithm takes an input array of size n (a power of 3). It divides the array into three equal-sized subarrays. It recursively sorts the first two subarrays. Then it recursively sorts the last two subarrays. Finally, it recursively sorts the first two subarrays again. Assume that the base case is when the array size is 1.

   (a) Write a recurrence relation for the running time $T(n)$ of this algorithm.

   (b) Solve the recurrence using our Master Theorem. What is the asymptotic running time of the algorithm?

9. Consider the recurrence $T(n) = 3T(n/2) + n \log n$. Can Our Master Theorem be applied to this recurrence? If so, apply it and state the solution. If not, explain why not.

10. Consider a recursive algorithm that, given an array of size n (assume n is a power of 2), divides the array into two halves. It then recursively sorts the first half, then recursively sorts the second half, and finally it compares each element in the first half to each element in the second half (in total, $n^2/4$ comparisons). Write a recurrence for the running time of the algorithm. Solve this recurrence using our Master Theorem if possible. If not, explain why not.

11. 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 $n\sqrt{n}$ additional work. Write a recurrence relation for the running time and solve using the Master Theorem, or explain why it cannot be applied.

[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 $a' + a'' = a$.