CS-UY 2413: Design & Analysis of Algorithms
Homework 2

Prof. Lisa Hellerstein
New York University
Fall 2024

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 $r \neq 1$, $\sum_{i=0}^{k} r^i = \frac{r^{k+1}-1}{r-1}$.

Problems

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) $f(n) = n \log n$, $g(n) = n^2$
   (c) $f(n) = 7n^3$, $g(n) = n^3 + 100n$
   (d) $f(n) = \sum_{i=0}^{n} 2^i$, $g(n) = 2^n$
   (e) $f(n) = \log_2 n$, $g(n) = \log_{10} n$

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

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 $T_{new}(n) = aT_{new}(n/b) + n \log n$ 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) $T_{new}(n) = \Omega(n \log^2 n)$ if $\log_b a = 1$

5. Consider the recurrence $T(n) = 2T(n/2) + n$ 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 $T(2) = 3$, 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. Finally, 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 sorting algorithm. Assume n is a power of 2.

   - 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.
   - Reverse the entire array.

(a) Explain why this algorithm is incorrect. Provide a counterexample.

(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. Design a recursive algorithm that takes as input a sorted array of integers and returns the number of unique integers in the array. Assume that the array has at least one element.

   (a) Write the pseudocode for your algorithm.

   (b) Write a recurrence relation for the runtime of your algorithm.

   (c) Solve the recurrence relation using the Master Theorem or another appropriate method. State the runtime complexity in Big-O notation.

9. Consider a recursive algorithm that finds the maximum element in an unsorted array. The algorithm works as follows: If the array has one element, that element is the maximum. Otherwise, it recursively finds the maximum of the first half of the array, and the maximum of the second half, and then compares these two values to find the overall maximum. Assume the array size n is a power of 2.

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

   (b) Solve the recurrence relation using the Master Theorem or another appropriate method. What is the time complexity in Big-O notation?

   (c) Is this algorithm more efficient than the iterative algorithm for finding the maximum element in an unsorted array? Justify your answer.

10. Suppose you have a divide-and-conquer algorithm that solves a problem of size n by recursively solving two subproblems of size n/2 and then combining the solutions in O(n log n) time.

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

    (b) Solve the recurrence relation using the Master Theorem or another appropriate method. What is the time complexity in Big-O notation?

11. Describe a divide and conquer algorithm to compute the sum of the elements of an array.

    (a) Write the pseudocode for your algorithm.

    (b) Give a recurrence relation for the running time of your algorithm.

    (c) Solve the recurrence relation using the Master Theorem or another appropriate method. What is the time complexity in Big-O notation? Compare to the iterative approach.

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 $a' + a'' = a$.