CS 161 W19: Recitation 1 Problems

January 2019

Exercise 0

For each of the following functions, prove whether $f = O(g)$, $f = \Omega(g)$, or $f = \Theta(g)$. For example, by specifying some explicit constants n_0 and $c > 0$ such that the definition of Big-Oh, Big-Omega, or Big-Theta is satisfied.

$$\begin{array}{lll} (a) & f(n) = n\log(n^3) & g(n) = n\log n \\ (b) & f(n) = 2^{2n} & g(n) = 3^n \\ (c) & f(n) = \sum_{i=1}^{n} \log i & g(n) = n\log n \end{array}$$

Exercise 1

Recall the Master theorem from lecture:
Theorem *Given a recurrence* $T(n) = aT(\frac{n}{b}) + O(n^d)$ *with* $a \geq 1$*, and* $b > 1$ *and* $T(1) = \Theta(1)$*, then*

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

Give a Big-Oh expression for each of the following recurrence relations:

1. $T(n) = 3T(\frac{n}{2}) + \Theta(n^2)$
2. $T(n) = 4T(\frac{n}{2}) + \Theta(n)$
3. $T(n) = 2T(\sqrt{n}) + O(\log n)$

Exercise 2

Given an array of integers $A[1 \dots n]$, compute a contiguous subarray $A[i \dots j]$ with the maximum possible sum. The entries of the array might be positive or negative.

1. What is the runtime of a brute force solution ?
2. The maximum sum subarray may lie entirely in the first half of the array or entirely in the second half. What is the third and only other possible case ?
3. Using the above observation, apply divide-and-conquer to arrive at a more efficient algorithm.
4. Give a rigorous proof by induction that your algorithm is correct. Make sure to explicitly state your inductive hypothesis, base case, inductive step, and conclusion.
5. What is the runtime of your solution ?
6. Advanced (Take Home) - Can you do even better using other non-recursive methods? ($O(n)$ is possible.)

Exercise 3

You have seen how integer multiplication can be improved upon with divide-and-conquer. Let us see a more generalized example of matrix multiplication. Assume that we have matrices A and B and we'd like to multiply them.

1. What is the naive solution and what is its runtime? Think about how you multiply matrices.
2. Now, let us divide up the problem into smaller chunks:

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

   We now have a divide-and-conquer strategy! Find a recurrence relation that captures the running time of this strategy, and then solve the recurrence relation to find the big-Oh runtime of the algorithm.

3. Can we do better? It turns out we can, by calculating only 7 of the sub-problems:

$$\begin{aligned} P_1 &= A(F - H) & P_5 &= (A + D)(E + H) \\ P_2 &= (A + B)H & P_6 &= (B - D)(G + H) \\ P_3 &= (C + D)E & P_7 &= (A - C)(E + F) \\ P_4 &= D(G - E) \end{aligned}$$

   Then we can solve XY as

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

   We now have a more efficient divide-and-conquer strategy! Find a recurrence relation that captures the running time of this strategy, and then solve the recurrence relation to find the big-Oh runtime of the algorithm.