CS 161 Fall 18 Section 1

September 2018

Asymptotic Analysis

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. *Bonus: prove little-Oh and little-Omega*

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

Probability

Let X be a random variable which is 1 with probability 1/100 and 0 with probability 99/100.

1. What is the expected value $\mathbb{E}[X]$?
2. Suppose you draw n independent random variables, $X_1, X_2, \ldots, X_n$, distributed like X. What is the expected value $\mathbb{E}[\sum_{i=1}^{n} X_i]$?
3. Suppose I draw independent random variables $X_1, X_2, \ldots$ and I stop when I see the first "1". For example, if I draw
$$X_1 = 0, X_2 = 0, X_3 = 0, X_4 = 1$$
then I would stop at X_4. Let N be the last index that we draw. (So in the previous example, $N = 4$). How big do you expect N to be?

Divide and Conquer

Warm up

You have now seen how digit 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 if we divide up the problem to smaller chunks like this:

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

We now have a divide and conquer strategy! Find the recurrence relation of this strategy and the runtime of this algorithm

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

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

And we can solve XY by

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

We now have a more efficient divide and conquer strategy! What is the recurrent relation of this strategy and what is the runtime of this algorithm

Maximum Sum Subarray

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

1. What is the complexity 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 apply divide and conquer to arrive at a more efficient algorithm
   (a) Prove that your algorithm works
   (b) What is the complexity of your solution ?
4. Advanced (Take Home) - Can you do even better using other non-recursive methods? ($O(n)$ is possible)

Recurrence Relations

Recall the Master theorem from lecture:

Theorem 0.1 *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*

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

What is the Big-Oh runtime for algorithms with the following recurrence relations?

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