CS 161 Winter 2020 Section 1

January 9-10, 2020

Asymptotic Analysis

Asymptotic Analysis Definitions

Definition 1 (Big-Oh notation)**.** Let f, g be functions from the positive integers to the non-negative reals. Then we say that:

$f = O(g)$ if there exist constants $c > 0$ and n_0 such that for all $n > n_0$,

$$f(n) \leq c \cdot g(n).$$

$f = \Omega(g)$ if there exist constants $c > 0$ and n_0 such that for all $n > n_0$,

$$f(n) \geq c \cdot g(n).$$

$f = \Theta(g)$ if $f = O(g)$ and $f = \Omega(g)$.

Asymptotic Analysis Problems

1. 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.

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

2. Give an example of f, g such that f is not in $O(g)$ and g is not in $O(f)$.

Induction

Snowball Fight

On a flat ice sheet, an *odd* number of penguins are standing such that their pairwise distances to each other are all different. At the strike of dawn, each penguin throws a snowball at the penguin closest to them. Show that there is always some penguin that doesn't get hit by a snowball.

Divide and Conquer

Maximum Sum Subarray

Given an array of integers $A[1..n]$, find a contiguous subarray $A[i,..j]$ with the maximum possible sum. The entries of the array might be positive or negative, and assume there is at least one element in the array.

1. What is a brute force solution, and what would be its runtime?
2. The maximum sum subarray may lie entirely in the first half of the array, or it may lie 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) What is the algorithm?
   (b) Prove its correctness (Hint: use strong induction).
   (c) What is the runtime?
4. Advanced (Take Home) - Can you do even better using other non-recursive methods? ($O(n)$ is possible)

(Optional Bonus Problem) Toom-Cook Multiplication

In this problem we will explore ideas that help improve the $O(n^{1.58})$ Karatsuba multiplication algorithm.

Suppose we want to compute the product of two n-digit numbers, x and y. In Karatsuba multiplication, we split each number into two $n/2$-digit numbers, but this time we will split them into three $n/3$-digit numbers. In particular, suppose that we write

$$x = a_2 \cdot 10^{2n/3} + a_1 \cdot 10^{n/3} + a_0 \qquad \text{and} \qquad y = b_2 \cdot 10^{2n/3} + b_1 \cdot 10^{n/3} + b_0.$$

Then

$$\begin{aligned} xy &= (a_2 \cdot 10^{2n/3} + a_1 \cdot 10^{n/3} + a_0)(b_2 \cdot 10^{2n/3} + b_1 \cdot 10^{n/3} + b_0) \\ &= (a_2b_2) \cdot 10^{4n/3} + (a_2b_1 + a_1b_2)10^n + (a_2b_0 + a_1b_1 + a_0b_2)10^{2n/3} + (a_1b_0 + a_0b_1)10^{n/3} + (a_0b_0) \end{aligned}$$

Thus, we can create a recursive multiplication algorithm if we can compute the five coefficients above using some $n/3$-digit multiplications. Given this,

(a) Find a way to compute the coefficients using nine $n/3$-digit multiplications. What is the runtime of the corresponding algorithm?

(b) Find a way to compute the coefficients using seven $n/3$-digit multiplications. What is the runtime of the corresponding algorithm?

(c) How many $n/3$-digit multiplications can you afford if you want to beat Karatsuba?

(d) **(Out of scope of this class.)** Find a way to compute the coefficients using six $n/3$-digit multiplications. What is the runtime of the corresponding algorithm?

(e) **(Even more out of scope of this class.)** Find a way to compute the coefficients using five $n/3$-digit multiplications. What is the runtime of the corresponding algorithm?