Recitation 1

1 Topics

- Asymptotic Complexity
- Naive Exponentiation Algorithm: Runtime Analysis
- Stock Buying on the Same Day

2 Asymptotic Complexity

2.1 Θ, O, and Ω - CLRS Chapter 3

$f(n) = \Theta(g(n))$

- Intuition: $f \approx g$.
- Words: After a certain point, we can put a lower and upper bound on f using constant multiples of g.
- Math: First, understand $\Theta(g(n))$ is actually a set of functions, $a(n)$, such that

$$\exists c_1, c_2, n_0 > 0 : \ c_1 g(n) < a(n) < c_2 g(n) \ \forall n \geq n_0.$$

So, $f(n) = \Theta(g(n))$ implies f is in this set, i.e.

$$\exists c_1, c_2, n_0 > 0 : \ c_1 g(n) < f(n) < c_2 g(n) \ \forall n \geq n_0.$$

$f(n) = O(g(n))$ and $F(n) = \Omega(g(n))$ are defined similarly to $f(n) = \Theta(n)$, but $f(n) = O(g(n))$ places only an upper bound on $f(n)$ and $f(n) = \Omega(g(n))$ places only a lower bound on $f(n)$. Note that $f(n) = \Theta(n)$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

2.2 Practice

Compute simple tight bounds for $f(n)$ where $f(n)$ is the following:

- 10^{80} (The number of atoms in the universe)
- $(20n)^7$
- $\log_5(n)$
- $\log(n^{100})$

- $5^{\log(3)}n^3 + 1080n^2 + \log(3)n^{3.1} + 6006$
- $\log(n!)$ Hint: Use Stirling's approximation $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$

2.3 Solutions

- 10^{80} is $\Theta(1)$ as there are no variables in this function. Therefore it is a constant, albeit a large one. Note that asymptotic complexity captures how the value of a function scales with the size of inputs, not the magnitude at any one point.
- $(20n)^7$ is $\Theta(n^7)$. Note that constant factors are ignored.
- $\log_5(n)$ is $\Theta(\log(n))$ as $\log_5(n) = \frac{\log(n)}{\log(5)}$. The base for logarithms should only be included if its value depends on a variable.
- $\log(n^{100})$ is $\Theta(\log(n))$! This is because $\log(n^{100}) = 100\log(n)$. Logs are powerful!
- $5^{\log(3)}n^3 + 1080n^2 + \log(3)n^{3.1} + 6006$ is $\Theta(n^{3.1})$. In addition to ignoring constant factors, ignore all but the most significant term. As n grows larger, the $\log(3)n^{3.1}$ term will dominate the sum.
- $\log(n!)$ is $\Theta(n\log(n))$. Use Sterling's Approximation:

  $$\log(n!) \approx \log(\sqrt{2\pi n}(\frac{n}{e})^n) = \log(n^n) + \log(\sqrt{2\pi n}) - \log(e^n) = n\log(n) + .5\log(2\pi n) - n\log(e)$$

  Therefore, $\log(n!) = \Theta(n\log(n))$.

3 Exponentiation: Review from Lecture

Problem: Given positive integers, a, b, c, compute $a^b \mod c$. Assume that a multiplication $\mod c$ (i.e. $a \cdot a \mod c$) takes runtime $O(\log^2 c)$ where $\log c$ is the number of bits in the input, c.

3.1 Naive Algorithm

The naive algorithm is simply to do the multiplication one at a time. You can think of this as running a recursive algorithm. Suppose that Exp is your recursive function, then:

$$Exp(a,b,c) = \begin{cases} a \cdot Exp(a,b-1,c) \mod c & \text{if } b > 1 \\ a \mod c & \text{otherwise} \end{cases}$$

We can write a recursive equation to represent the runtime of this naive algorithm:

$$T(a,b,c) = \begin{cases} T(a,b-1,c) + \Theta(\log^2 c) & \text{if } b > 1 \\ \Theta(1) & \text{otherwise} \end{cases}$$

Solving this recurrence gives $T(a,b,c) = \Theta(b\log^2 c)$. This is actually *not* an efficient algorithm as we define it in theoretical computer science. In fact, it is exponential with regard to the number

of bits in the input! In computer science, we generally represent the input in bits. We define an efficient algorithm to be one that is polynomial with regard to how long it takes to read the input. For example, a value a takes $\Theta(\log_2 a)$ time to read since a is represented in binary.

This means that for the Exponentiation problem, the number of bits we receive as input is $\log_2 a + \log_2 b + \log_2 c$ and this is also how long it takes to read the input. Suppose that $b \geq a \geq c$, then let $n = \log_2 b$. The runtime of the naive exponentiation algorithm is $\Theta(2^{\log_2 b} \log^2 c) = \Theta(2^n n^2)$ which is exponential in terms of the number of steps it takes to read the input!

3.1.1 Repeated Squaring*

You've probably seen this method elsewhere. As the name states, you repeatedly square the result until you obtain the target product. For example, using the repeated squaring method, we can calculate $256 = ((2^2)^2)^2$.

In general, for the Exponentiation problem, we can use the following recursive procedure:

$$Exp2(a, b, c) = \begin{cases} 1 & \text{if } b = 0 \\ (Exp2(a, b/2, c))^2 \mod c & \text{if } b \text{ is even} \\ a \cdot Exp(a, b-1, c) \mod c & \text{otherwise} \end{cases}$$

The runtime of the improved recursive procedure can be represented by the following recursive function:

$$T(a, b, c) = \begin{cases} T(a, b/2, c) + \Theta(\log^2 c) & \text{if } b > 1 \\ \Theta(1) & \text{otherwise} \end{cases}$$

This gives a runtime of $T(a, b, c) = \Theta(\log b \log^2 c)$, which is a lot better than the naive algorithm!

4 Stock Buying Problem

Exercise (don't peek below!): What differs when you are allowed to buy *and* sell on the same day? In lecture, we stated that we want to find the maximum profit when $i < j$ where i is the date you bought the stock and j is the date you sold the stock.

Answer: Not much! You just have to consider 0 (i.e. your maximum value can be 0 now) because buying and selling on the same day cannot result in a value greater than 0.