CSE 332: Data Abstractions

Section 2: Asymptotics & Recurrences Solutions

0. Big-Oh Proofs

For each of the following, prove that $f \in \mathcal{O}(g)$.

(a) $f(n) = 7n$ $\quad$ $g(n) = \dfrac{n}{10}$

Solution: Choose $c = 70$, $n_0 = 1$. Then, note that $7n = \frac{70n}{10} \leq 70\left(\frac{n}{10}\right)$ for all $n \geq 1$. So, $f(n) \in \mathcal{O}(g(n))$.

(b) $f(n) = 1000$ $\quad$ $g(n) = 3n^3$

Solution: Choose $c = 3$, $n_0 = 1000$. Then, note that $1000 \leq n \leq n^3 \leq 3n^3$ for all $n \geq 1000$. So, $f(n) \in \mathcal{O}(g(n))$.

(c) $f(n) = 7n^2 + 3n$ $\quad$ $g(n) = n^4$

Solution: Choose $c = 14$, $n_0 = 1$. Then, note that $7n^2 + 3n \leq 7(n^4 + n^4) \leq 14n^4$ for all $n \geq 1$. So, $f(n) \in \mathcal{O}(g(n))$.

(d) $f(n) = n + 2n \lg n$ $\quad$ $g(n) = n \lg n$

Solution: Choose $c = 3$, $n_0 = 1$. Then, note that $n + 2n \lg n \leq n \lg n + 2n \lg n = 3n \lg n$ for all $n \geq 1$. So, $f(n) \in \mathcal{O}(g(n))$.

1. Asymptotics Disproof

Prove that $n^2 \notin \mathcal{O}(n)$.

Solution:

Assume for the sake of contradiction that $n^2 \in \mathcal{O}(n)$. Then, there exist $c, n_0 > 0$ such that $n^2 \leq cn$ for all $n \geq n_0$. If $n^2 \leq cn$, then $n \leq c$. Consider $n_1 = \max(n_0, c+1)$. Since $n_1 \geq n_0$, we know $n_1 \leq c$, but $c + 1 \not\leq c$ for any c. This is a contradiction! So, $n^2 \notin \mathcal{O}(n)$.

2. Is Your Program Running? Better Catch It!

For each of the following, determine the asymptotic worst-case runtime in terms of n.

(a)

```
int x = 0;
for (int i = n; i >= 0; i--) {
   if ((i % 3) == 0) {
      break;
   }
   else {
      x += n;
   }
}
```

Solution: This is $\Theta(1)$, because n, $n-1$, or $n-2$ will be divisible by three. So, the loop runs at most 3 times.

(b)

```
int x = 0;
for (int i = 0; i < n; i++) {
   for (int j = 0; j < (n * n / 3); j++) {
      x += j;
   }
}
```

Solution:

$$\sum_{i=0}^{n-1} \sum_{j=0}^{n^2/3-1} 1 = \sum_{i=0}^{n} \frac{n^2}{3} = n\left(\frac{n^2}{3}\right) = \Theta(n^3)$$

(c)

```
int x = 0;
for (int i = 0; i <= n; i++) {
   for (int j = 0; j < (i * i); j++) {
      x += j;
   }
}
```

Solution:

$$\sum_{i=0}^{n} \sum_{j=0}^{i^2-1} 1 = \sum_{i=0}^{n} i^2 = \left(\frac{n(n+1)(2n+1)}{6}\right) = \Theta(n^3)$$

3. Induction Shminduction

Prove $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$ by induction on n.

Solution:

Let $P(n)$ be the statement "$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$" for all $n \in \mathbb{N}$. We prove $P(n)$ by induction on n.

Base Case. Note that $\sum_{i=0}^{0} 2^i = 0 = 2^0 - 1$. So, $P(0)$ is true.

Induction Hypothesis. Suppose $P(k)$ is true for some $k \in \mathbb{N}$.

Induction Step. Note that

$$\begin{aligned}
\sum_{i=0}^{k+1} 2^i &= \sum_{i=0}^{k} 2^i + 2^{k+1} \\
&= 2^{k+1} - 1 + 2^{k+1} && \text{[By IH]} \\
&= 2^{k+2} - 1
\end{aligned}$$

Note that this is exactly $P(k+1)$.

So, the claim is true by induction on n.

4. The Implications of Asymptotics

For each of the following, determine if the statement is true or false.

(a) $f(n) \in \Theta((g(n)) \to f(n) \in \mathcal{O}(g(n))$

Solution:

This is true. By definition of $f(n) \in \Theta((g(n))$, we have $f(n) \in \mathcal{O}(g(n))$.

(b) $f(n) \in \Theta(g(n)) \to g(n) \in \Theta(f(n))$

Solution:

This is true. By definition of $f(n) \in \Theta(g(n))$, we have $f(n) \in \mathcal{O}(g(n))$ and $f(n) \in \Omega(g(n))$. So, there exist $n_0, n_1, c_0, c_1 > 0$ such that $f(n) \leq c_0 g(n)$ for all $n \geq n_0$ and $f(n) \geq c_1 g(n)$ for all $n \geq n_1$. Define $n_2 = \max(n_0, n_1$ and note that both inequalities hold for all $n \geq n_2$. Then, dividing both sides by their constants, we have:

$$g(n) \geq \frac{f(n)}{c_0}$$
$$g(n) \leq \frac{f(n)}{c_1}$$

So, we've found constants $\left(\frac{1}{c_0}, \frac{1}{c_1}\right)$ and a minimum n (n_2) that satisfy the definitions of Omega and Oh. It follows that $g(n) is \Theta(f(n))$.

(c) $f(n) \in \Omega((g(n) \to g(n) \in \mathcal{O}(f(n))$

Solution:

This is true. This is basically identical to the previous part (except we only have to do half the work).

5. Asymptotic Analysis

For each of the following, determine if $f \in \mathcal{O}(g)$, $f \in \Omega(g)$, $f \in \Theta(g)$, several of these, or none of these.

(a) $f(n) = \log n$ $g(n) = \log \log n$

Solution: $f(n) \in \Omega(g(n))$

(b) $f(n) = 2^n$ $g(n) = 3^n$

Solution: $f(n) \in \mathcal{O}(g(n))$

(c) $f(n) = 2^{2n}$ $g(n) = 2^n$

Solution: $f(n) \in \Omega(g(n))$

6. Summations

For each of the following, find a closed form.

(a) $\sum_{i=0}^{n} i^2$

Solution:

Since we're summing up squares, let's guess that it's $\mathcal{O}(n^3)$. If it is, then we know it's of the form:

$$an^3 + bn^2 + cn + d$$

Let's look at small examples:

- $n = 0 \to 0$
- $n = 1 \to 1$
- $n = 2 \to 5$
- $n = 3 \to 14$
- $n = 4 \to 30$

Plugging these answers in, we get the following equations:

- $d = 0$
- $a + b + c = 1$
- $8a + 4b + 2c = 5$
- $27a + 9b + 4c = 14$

Solving these equations gives us: $d = 0, c = \frac{1}{6}, b = \frac{1}{2}, a = \frac{1}{3}$

So, the summation is $\frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{3}$.

(b) $\sum_{i=0}^{\infty} x^i$

Solution:

Define $S = \sum_{i=0}^{\infty} x^i$ and consider

$$xS = x\sum_{i=0}^{\infty} x^i = \sum_{i=0}^{\infty} x^{i+1} = \sum_{i=1}^{\infty} x^i = S - 1$$

So, since $xS = S - 1$; solving for S gives us $S = \frac{1}{1-x}$.

7. Recurrences and Closed Forms

For each of the following code snippets, find a recurrence for the worst case runtime of the function, and then find a closed form for the recurrence.

(a) Consider the function f:

```
f(n) {
   if (n == 0) {
      return 1;
   }
   return 2 * f(n - 1) + 1;
}
```

- Find a recurrence for $f(n)$.

Solution:

$$T(n) = \begin{cases} c_0 & \texttt{if } n = 0 \\ T(n-1) + c_1 & \texttt{otherwise} \end{cases}$$

- Find a closed form for $f(n)$.

Solution:

Unrolling the recurrence, we get $T(n) = \underbrace{c_1 + c_1 + \cdots + c_1}_{n \text{ times}} + c_0 = c_1 n + c_0$.

(b) Consider the function g:

```
g(n) {
   if (n == 1) {
      return 1000;
   }
   if (g(n/3) > 5) {
      return 5 * g(n/3);
   }
   else {
      return 4 * g(n/3);
   }
}
```

- Find a recurrence for $g(n)$.

Solution:

$$T(n) = \begin{cases} c_0 & \texttt{if } n = 1 \\ 2T(n/3) + c_1 & \texttt{otherwise} \end{cases}$$

- Find a closed form for $g(n)$.

Solution:

The recursion tree has height $\log_3(n)$. Level i has work $\left(\frac{c_1 2^i}{3^i}\right)$. So, putting it together, we have:

$$\sum_{i=0}^{\log_3(n)-1}\left(\frac{c_1 2^i}{3^i}\right) + 2^{\log_3(n)}c_0 = c_1\sum_{i=0}^{\log_3(n)-1}\left(\frac{2}{3}\right)^i + n^{\log_3(2)}c_0 \quad = \frac{1-\left(\frac{2}{3}\right)^{\log_3(n)}}{1-\frac{2}{3}} + n^{\log_3(2)}c_0$$
$$= 3 - \left(\frac{2}{3}\right)^{\log_3(n)} + n^{\log_3(2)}c_0$$

8. Big-Oh Bounds for Recurrences

For each of the following, find a Big-Oh bound for the provided recurrence.

(a) $T(n) = \begin{cases} 1 & \texttt{if } n = 1 \\ 8T(n/2) + 4n^2 & \texttt{otherwise} \end{cases}$

Solution:

Note that $a = 8$, $b = 2$, and $c = 2$. Since $\log_2(8) = 3 > 2$, we have $T(n) \in \Theta(n^{\log_2(8)}) = \Theta(n^3)$ by Master Theorem.

(b) $T(n) = \begin{cases} 1 & \texttt{if } n = 1 \\ 7T(n/2) + 18n^2 & \texttt{otherwise} \end{cases}$

Solution:

Note that $a = 7$, $b = 2$, and $c = 2$. Since $\log_2(7) = 3 > 2$, we have $T(n) \in \Theta(n^{\log_2(7)})$ by Master Theorem.

(c) $T(n) = \begin{cases} 1 & \texttt{if } n = 0 \\ T(n-1) + 3 & \texttt{otherwise} \end{cases}$

Solution:

There are n terms to unroll and each one is constant. This is $\Theta(n)$.

(d) $T(n) = \begin{cases} 1 & \texttt{if } n = 1 \\ T(n/2) + 3 & \texttt{otherwise} \end{cases}$

Solution:

Note that $a = 1$, $b = 2$, and $c = 0$. Since $\log_2(1) = 0 = 2$, we have $T(n) \in \Theta(\lg(n))$ by Master Theorem.

(e) $T(n) = \begin{cases} 1 & \texttt{if } n = 0 \\ T(n-1) + T(n-2) + 3 & \texttt{otherwise} \end{cases}$

Solution:

Note that this recurrence is bounded above by $T(n) = 2T(n-1) + 3$. If we unroll that recurrence, we get $3 + 2(3 + 2(3 + \cdots + 2(1)))$. This is approximately $\sum_{i=0}^{n} 3 \times 2^i = 3(2^{n+1} - 1) = \mathcal{O}(2^n)$. We can actually find a better bound (e.g., it's not the case that $T(n) \in \Omega(2^n)$.

9. Hello, elloH, lleoH, etc.

Consider the following code:

```
p(L) {
   if (L == null) {
      return [[]];
   }
   List ret = [];

   int first = L.data;
   Node rest = L.next;

   for (List part : p(rest)) {
      for (int i = 0; i <= part.size()) {
         part = copy(part);
         part.add(i, first);
         ret.add(part);
      }
   }
   return ret;
}
```

(a) Find a recurrence *for the output complexity* of $p(L)$. That is, if $|L| = n$, what is the size of the output list, in terms of n? Then, find a Big-Oh bound for your recurrence.

Solution:

The base case returns a list of length one. The recursive case adds one list in each iteration of the for loop for each list returned. So, the recurrence is $\text{Out}(n) = \begin{cases} 1 & \texttt{if } n = 0 \\ n\text{Out}(n-1) & \texttt{otherwise} \end{cases}$

So, $\text{Out}(n) \in \mathcal{O}(n!)$

(b) Now, find a recurrence *for the time complexity* of $p(L)$, and a Big-Oh bound for this recurrence as well.

Solution:

$$T(n) = \begin{cases} 1 & \texttt{if } n = 0 \\ T(n-1) + \text{Out}(n-1)n & \texttt{otherwise} \end{cases}$$

Unrolling, we get $T(n) = n! + (n-1)! + (n-2)! + \cdots + 0! + 1 \leq n(n!) \leq (n+1)! \in \mathcal{O}((n+1)!)$

10. MULTI-pop

Consider augmenting a standard `Stack` with an extra operation:

`multipop(k)`: Pops up to k elements from the `Stack` and returns the number of elements it popped

What is the amortized cost of a series of `multipop`'s on a `Stack` assuming `push` and `pop` are both $\mathcal{O}(1)$?

Solution:

Consider an *empty* `Stack`. If we run various operations (`multipop`, `pop`, and `push`) on the `Stack` until it is once again empty, we see the following:

- In general, `multipop(k)` takes time proportional to k.
- If over the course of running the operations, we `push` n items, then each item is associated with *at most* one `multipop` or `pop`.
- It follows that the largest number of time the `multipops` can take in aggregate is n.
- Note that the *smallest possible number of operations*is $n + 1$ (n `pushes` and 1 `multipop`).

So, the amortized analysis for this series of operations is at most $\dfrac{2n}{n+1} = \mathcal{O}(1)$.