CIS 121—Data Structures and Algorithms with Java—Spring 2018

Big-Oh Notation—Monday, January 22/Tuesday, January 23

Learning Goals

- Review Big-Oh and learn big/small omega/theta notations
- Discuss running time analysis of algorithms

What are Algorithms?

What is an algorithm? An **algorithm** is any well-defined computational procedure that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**. Therefore, an algorithm is a sequence of computational steps that transform the input into output. We can also view an algorithm as a tool for solving a well-specified computational problem. The statement of the problem specifies the desired input/output relationship, and the algorithm describes a specific computational procedure for achieving that relationship.

Best, Average, Worst Case Run Time Analysis

When analyzing algorithms, we are often interested in analyzing the best, average, and worst cases of running time.

Typically, best case performance is not really of concern due to triviality. Algorithms may be modified to make best case performance trivial for small input by hardcoding. In these cases, the best case performance is effectively meaningless!

Often, it is of more concern to perform **worst case analysis**, i.e. identifying the inputs that cause the algorithm to have the longest running time (or use the most space) and identifying the running time and usage bounds. Why is this useful?

- The worst case running time of an algorithm gives an upper bound on the running time for any input. Knowing this provides a guarantee that the algorithm never takes any longer.
- For some algorithms, the worst case may occur fairly often.
- Often, the "average case" is roughly as bad as the worst case.

Finally, in some cases we may be interested in the **average case** running time of an algorithm, where we would use **probabilistic analysis** to examine particular algorithms. This doesn't surface too much, as what constitutes an "average input" is often not apparent. Often, we would then assume that all inputs of a particular size are equally likely (uniform distribution). This assumption is often violated in practice, so we might modify an algorithm to be **randomized**, to enable a probabilistic analysis and *expected* running time.

Big-Oh Definitions

Presented below is a brief overview of asymptotic notation that will be fundamental in this course.

Big-Oh Notation

Definition (1)**.** $f(n) \in O(g(n))$ if there exist positive constants n_0 and c such that $f(i) \leq cg(i)$ for all $i \geq n_0$.

Definition (2)**.** $f(n) \in O(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ or a constant.

Simplified: If $f(n)$ is in $O(g(n))$, $g(n)$ is an asymptotic upper bound for $f(n)$.

Little-o Notation

Definition (1)**.** $f(n) \in o(g(n))$ if for any positive constant c, there exists a positive constant n_0 such that $f(i) < cg(i)$ for all $i \geq n_0$.

Definition (2)**.** $f(n) \in o(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

Note: $f(n) \in o(g(n)) \Rightarrow f(n) \in O(g(n))$

Big-Omega Notation

Definition (1)**.** $f(n) \in \Omega(g(n))$ if there exist positive constants n_0 and c such that $f(i) \geq cg(i)$ for all $i \geq n_0$.

Definition (2)**.** $f(n) \in O(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ or a constant.

Simplified: If $f(n)$ is $\Omega(g(n))$, $g(n)$ is an asymptotic lower bound for $f(n)$.

Little-ω Notation

Definition (1)**.** $f(n) \in \omega(g(n))$ if for any positive constant c, there exists a positive constant n_0 such that $f(i) > cg(i)$ for all $i \geq n_0$.

Definition (2)**.** $f(n) \in \omega(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

Note: $f(n) \in \omega(g(n)) \Rightarrow f(n) \in \Omega(g(n))$

Big-Theta Notation

Definition (1)**.** $f(n) \in \Theta(g(n))$ if and only if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.

Definition (2)**.** $f(n) \in \Theta(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} =$ a nonzero constant.

Simplified: If $f(n)$ is $\Theta(g(n))$, $g(n)$ is an **asymptotically tight** bound for $f(n)$.

The notations refer to *classes of functions*. When you read $f(n) = O(g(n))$, this is equivalent to the statement: $f(n) \in O(g(n))$. Specifically, $f(n)$ is in the class of functions which are asymptotically bounded above by $g(n)$. Likewise, Big-Ω, Big-Θ, etc., all reflect classes of functions.

Problems (Big-Oh)

Problem 1a

Prove that $3n^2 + 100n = \Theta(5n^2)$

Solution.

$$\begin{aligned}
\lim_{n\to\infty} \frac{3n^2 + 100n}{5n^2} &= \lim_{n\to\infty} \frac{3n + 100}{5n} \\
&= \lim_{n\to\infty} \frac{3n}{5n} \\
&= 3/5 \\
3n^2 + 100n &= \Theta(5n^2)
\end{aligned}$$

Problem 1b

Prove that $n \log n = \Omega(n)$

Solution.

We will prove this by induction.

Base Case: $n = 4$. $4 \log 4 = 8 > 4$, so this holds.

Induction Hypothesis: Assume that $k \log k \geq k$ for some $k \geq 4$.

Induction Step: We need to show that $(k+1)\log(k+1) \geq k+1$

Log is monotonically increasing so $\log(k+1) > logk$

$$\begin{aligned}
(k+1)\log(k+1) &> (k+1)\log k \\
&= k \log k + \log k \\
&> k \log k + 1 \\
&> k + 1
\end{aligned}$$

Problem 2

You are given the following algorithm for `Bubble-Sort`:

Algorithm 1 Bubble Sort

```
function Bubble-Sort(A, n)
    for i ← 0 to n − 2 do
        for j ← 0 to n − i − 2 do
            if A[j] > A[j + 1] then
                swap(A[j], A[j + 1])
            end if
        end for
    end for
end function
```

Given some sequence $\langle a_1, a_2, \ldots, a_n \rangle$ in A, we say an inversion has occurred if $a_j < a_i$ for some $i < j$. At each iteration, `Bubble-Sort` checks the array A for an inversion and performs a swap if it finds one. How many swaps does `Bubble-Sort` perform in the *worst-case* and in the *average-case*?

Solution.

It should be fairly obvious that the worst-case scenario for bubble-sort occurs when A contains elements in reverse-sorted order. Let I denote the number of inversions. In this situation, the total number of inversions is the exact number of possible pairs of elements, as each swap removes exactly one inversion:

$$I_{worst} = \binom{n}{2}$$

In the average-case scenario, we determine the expected number of inversions in a random array of n elements. Specifically, we consider a random permutation of n distinct elements, $\langle a_1, a_2, \ldots, a_n \rangle$.

Let X_{ij} denote an indicator R.V. such that,

$$X_{ij} = \begin{cases} 1 & \text{if } a_i, a_j \text{ inverted} \\ 0 & \text{otherwise} \end{cases}$$

Thus, we have

$$\begin{aligned} I_{average} &= \sum_{i,j\,:\,i<j} X_{ij} \\ \mathrm{E}[I] &= \sum_{i,j\,:\,i<j} \mathrm{E}[X_{ij}] \\ &= \sum_{i,j\,:\,i<j} \Pr[X_{ij} = 1] \\ &= \frac{\binom{n}{2}}{2} \\ &= \frac{n(n-1)}{4} \end{aligned}$$

In the above, we let $\Pr[X_{ij} = 1] = \frac{1}{2}$ because in a random permutation (uniform probability), the probability of $a_i < a_j$ is the same as $a_i > a_j$. $\square$

Problem 3

Prove or disprove the following statement:

$$\lg(n!) \text{ is } \Theta(n \lg n).$$

Solution.

We first show $\lg(n!)$ is $O(n \lg n)$. Picking $c = 1$ and $n_0 = 1$, we have

$$\lg(n!) = \sum_{i=1}^{n} \lg i \leq n \lg n$$

This is clearly true for all $n > n_0$. Therefore, we are done.

We then show that $\lg(n!)$ is $\Omega(n \lg n)$. Our strategy is to find an easier to work with lower-bound for $\lg n!$ that is larger than some $cn \lg n$.

$$\begin{aligned} \lg n! &= \lg 1 + \lg 2 + \cdots + \lg n \\ &\geq \lg \frac{n}{2} + \lg(\frac{n}{2} + 1) + \cdots + \lg n \quad \text{delete the first half of the terms} \\ &\geq \frac{n}{2} \cdot \lg \frac{n}{2} \quad \text{replace remaining terms by smallest one} \end{aligned}$$

Choosing $c = \frac{1}{4}$ and $N = 4$, it is clear that $\frac{n}{2} \lg \frac{n}{2} \geq \frac{n}{4} \lg n$ with some algebraic manipulation:

$$\begin{aligned} \frac{n}{2} \lg \frac{n}{2} &\geq \frac{n}{4} \lg n \\ \frac{n}{2} \lg n - \frac{n}{2} &\geq \frac{n}{4} \lg n \\ n \lg n &\geq 2n \\ \lg n &\geq 2 \end{aligned}$$

Therefore, $\lg(n!)$ is $\Omega(n \lg n)$. $\square$

Problem 4

Prove that a function $f(n)$ (which is eventually positive for all $n \geq n_p$) is polynomially bounded iff $\log(f(n)) \in O(\log n)$. In other words:

$$f(n) \in O(n^k) \Leftrightarrow \log(f(n)) \in O(\log n)$$

where k is some constant.

Solution.

($\Rightarrow$) Suppose $f(n) \in O(n^k)$ for some k.

Then $\exists c, n_0$ such that $f(n) \leq c \cdot n^k$ for all $n \geq n_0$. Since $f(n)$ is eventually positive for all $n \geq n_p$, consider all $n \geq \max(n_0, n_p)$. Take the log (a monotonically increasing function for positive values) of both sides, and we get that:

$$\begin{aligned} \log(f(n)) &\leq \log(c \cdot n^k) \\ &\leq \log(c) + \log(n^k) \\ &\leq \log(c) + k \log(n) \\ &\in O(\log n) \end{aligned}$$

($\Leftarrow$) Suppose $\log(f(n)) \in O(\log n)$. Note here that logarithm, no matter which base, is all in $O(\log n)$. WLOG, we can assume here that the base is e, and these are natural logs.

Then $\exists c, n_0$ such that $\ln(f(n)) \leq c \ln n$ for all $n \geq n_0$. Exponentiating both sides (a monotonically increasing function), we get that:

$$\begin{aligned} e^{\ln(f(n))} &\leq e^{c \ln n} \\ f(n) &\leq e^{\ln(n^c)} \\ &\leq n^c \end{aligned}$$

which is in $O(n^k)$ for $k \geq c$. $\square$

Problem 5

Prove or disprove the following:

1. $\lceil \lg n \rceil!$ is polynomially bounded.

2. $\lceil \lg \lg n \rceil!$ is polynomially bounded.

Solution.

You can use the results from problem 3 and 4 to solve this question.

1. Let $g(n) = \lceil \lg n \rceil!$, and let $y = \lceil \lg n \rceil$. Then

$$\begin{aligned} \lg g(n) &= \lg y! \\ &\in \Theta(y \lg y) \end{aligned}$$

by the result from problem 3. Substituting y back in,

$$\begin{aligned} (y \lg y) &= (\lceil \lg n \rceil)(\lg \lceil \lg n \rceil) \\ &\notin O(\lg n), \end{aligned}$$

since $\lg \lceil \lg n \rceil$ will eventually exceed any constant c. (So $\nexists c \ni (\lceil \lg n \rceil)(\lg \lceil \lg n \rceil) \le c \cdot \lg n$.) Also if you calculate the limit

$$\lim_{n \to \infty} \frac{(\lceil \lg n \rceil)(\lg \lceil \lg n \rceil)}{\lg n} = \lim_{n \to \infty} \lg \lceil \lg n \rceil = \infty,$$

it shows that $(\lceil \lg n \rceil)(\lg \lceil \lg n \rceil) = \omega(\lg n)$. So clearly $(\lceil \lg n \rceil)(\lg \lceil \lg n \rceil) \notin O(\lg n)$. By the if and only if statement from problem 4, since $\lg g(n) \notin O(\lg n)$, we can also say that $g(n)$ is not polynomially bounded.

2. Let $h(n) = \lceil \lg \lg n \rceil!$, and let $y = \lceil \lg \lg n \rceil$. Similar to part 1,

$$\begin{aligned} \lg h(n) &= \lg y! \\ &\in \Theta(y \lg y), \end{aligned}$$

Substituting y back in,

$$\begin{aligned} (y \lg y) &= (\lceil \lg \lg n \rceil)(\lg \lceil \lg \lg n \rceil) \\ &= O(\lceil \lg \lg n \rceil \lceil \lg \lg n \rceil) \\ &= O((\lg \lg n)^2) \\ &= O(2^{\lg \lg n}) \\ &= O(\lg n) \end{aligned}$$

Or if you calculate the limit

$$\lim_{n \to \infty} \frac{(\lceil \lg \lg n \rceil)(\lg \lceil \lg \lg n \rceil)}{\lg n} = 0.$$

This shows that $(\lceil \lg \lg n \rceil)(\log \lceil \lg \lg n \rceil) = o(\lg n) \Rightarrow (\lceil \lg \lg n \rceil)(\lg \lceil \lg \lg n \rceil) \in O(\lg n)$. By the if and only if statement from problem 4, since $\lg h(n) \in O(\lg n)$, we can also say that $h(n)$ is polynomially bounded.

$\square$