CSE101 Discussion Notes

Scott Yilek

September 27, 2006

1. Introductions
2. Recommendations
3. Order Notation
4. Simple algorithmic analysis
5. Questions

1 Introductions

• Scott Yilek, syilek@cs.ucsd.edu
  Brian McFee, bmcfee@cs.ucsd.edu

• Office Hours: Scott(B240A) T6-7pm TH11-12; Brian(B225) T10-12

2 Recommendations

• GO TO RUSSELL’S OFFICE HOURS EVERY WEEK

• Take the calibration homework seriously

• Come to discussion section and email TAs if you have any questions

• Start the homeworks early

3 Order Notation

NOTE: See section 2.3 of book (JS) and order notation cheat sheet on website

Why Do we need Order notation? Because in designing and comparing algorithms in this class, we are most concerned with how our running times grow as the input size increases and not necessarily the exact running times. There are 5 flavors of order notation, intuitively corresponding to $<,\leq,=,\geq$, and $>$. 
3.1 Useful Facts

\[ n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 \]
\[ 2^{\log_2(n)} = n \]
\[ a \log b = \log b^a \]
\[ 1 + 2 + \ldots + n = \frac{n(n+1)}{2} \]
\[ 1 + 2 + 4 + 8 + \ldots + 2^n = 2^{n+1} - 1 \]

3.2 Big-O

**Definition.** \( O(g(n)) = \{ f(n) : \text{there exist positive constants } C_1 \text{ and } N_1 \text{ such that } 0 \leq f(n) \leq C_1 \cdot g(n), \text{ for all } n \geq N_1 \} \).

- \( O(g(n)) \) is a SET.
- Intuitively, set of functions which are \( \leq \) to \( g(n) \)

**Example.** \( 10n^3 + 2n^2 \in O(n^3) \)

Need to find \( C_1 \) and \( N_1 \):

TRICK: Turn all terms that are not \( n^3 \) into \( n^3 \) terms:

\[ 10n^3 + 2n^2 \leq 10n^3 + 2n^3 \leq 12n^3 \]

So, we simply set \( C_1 = 12 \) and \( N_1 = 1 \).

**Example.** [JS 2.3.22] \( 2^n \in O(n!) \)

Again need to find \( C_1 \) and \( N_1 \):

\[ 2^n = \underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{n \text{ times}} \]
\[ n! = \underbrace{n \cdot (n-1) \cdot \ldots \cdot 3 \cdot 2 \cdot 1}_{n \text{ terms}} \]

Notice that for all \( n \geq 2 \), \( n! \) has \( n-1 \) terms that are greater than or equal to 2. But we would like \( n \) terms greater than or equal to 2. So set \( C_1 = 2 \) and \( N_1 = 2 \).

**USEFUL** \( f_1(n) + f_2(n) + \ldots + f_d(n) \in O(h(h)) \) where \( h(n) \) is the max among the \( f \)s. ONLY IF \( d \) is a constant!!!!!!!!!!!!

3.3 Little-O

**Definition.** \( o(g(n)) = \{ f(n) : \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \} \).

Essentially equivalent to

\[ o(g(n)) = \{ f(n) : \text{for every positive constant } c \text{ there exists a positive constant } n_0 \text{ such that for all } n \geq n_0, 0 \leq f(n) < cg(n) \} \]
• $o(g(n))$ is a SET.
• Intuitively, set of functions which are $<$ to $g(n)$

**Example.**  $n^2 \in o(n^3)$

$$\frac{n^2}{n^3} = \frac{1}{n} \rightarrow 0 \text{ as } n \text{ moves towards infinity}$$

**Example.**  Is $n - 25 \in o(n)$?

No,

$$\frac{n - 25}{n} \rightarrow 1$$

### 3.4 Big-Omega

**Definition.** $\Omega(g(n)) = \{f(n) : \text{there exist positive constants } C_2 \text{ and } N_2 \text{ such that } 0 \leq g(n) \leq C_2 \cdot f(n), \text{ for all } n \geq N_2\}$.

• $\Omega(g(n))$ is a SET.
• Intuitively, set of functions which are $\geq$ to $g(n)$

**Example.**  Is $10n^3 + 2n^2 \in \Omega(n^3)$?

Find $C_2$ and $N_2$. Let $C_2 = 10, N_2 = 1$.

### 3.5 Little-Omega

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

Essentially equivalent to

$$\omega(g(n)) = \{f(n) : \text{for every positive constant } c, \text{ there exists a positive constant } n_0 \text{ such that for all } n \geq n_0, 0 \leq cg(n) < f(n)\}$$

• $o(g(n))$ is a SET.
• Intuitively, set of functions which are $>$ to $g(n)$

**Example.**  $n \log(n) \in \omega(n)$

$$\frac{n \log(n)}{n} = \log(n) \rightarrow \infty$$
3.6 Theta

Definition. \( \Theta(g(n)) = \{ f(n) : f(n) \in O(g(n)) \land f(n) \in \Omega(g(n)) \} \)

- \( \Theta(g(n)) \) is a SET.
- Intuitively, set of functions which are = to \( g(n) \)

Example. Is \( 10n^3 + 2n^2 \in \Theta(n^3) \)?
Yes, because of previous examples.

Example. Is \( n \in \Theta(n^2) \)?
No, \( n \not\in \Omega(n^2) \). This is clear from inspection, but how do we prove this? The best way to show \( f(n) \not\in \Omega(g(n)) \) is to show that \( f(n) \in o(g(n)) \). If we recall that \( f(n) \in \Omega(g(n)) \) essentially means that \( f(n) \geq g(n) \) and \( f(n) \in o(g(n)) \) essentially means \( f(n) < g(n) \), then this makes sense: to show that \( f(n) \) is not greater than or equal to \( g(n) \), we would show that it’s greater than \( g(n) \).
So, \( n \not\in \Omega(n^2) \) since \( n \in o(n^2) \), which we can prove by showing that \( \lim_{n \to \infty} \frac{n}{n^2} = 0 \).

3.7 More Examples

1. Is \( n^n \in \omega(2^n) \)?
Yes,
\[
\frac{n^n}{2^n} = \frac{2^n \log n}{2^n} = (2^n \log n - 1) = (2^n)^{\log(n/2)} = \left(\frac{n}{2}\right)^n
\]
and
\[
\lim_{n \to \infty} \left(\frac{n}{2}\right)^n = \infty
\]

2. Is \( 2^{n+2} \in O(2^n) \)?
Yes, \( C_1 = 4, N_1 = 1 \). Then \( 2^{n+2} \leq 4 \cdot 2^n \).

3. Is \( n \log n \in O(\log^2 n) \)?
No, since \( n \log n \in \omega(\log^2 n) \):
\[
\frac{n \log n}{\log^2 n} = \frac{n \log n}{n \cdot \log n} = \frac{n \log n}{n} \log n
\]
and \( \lim_{n \to \infty} \frac{n}{\log n} \) is \( \infty \).
4. True or False: $n^5 + 50n^4 + n^3 + 10n^2 + n \in O(n^5)$

TRUE, let $c = 62, n_0 = 1.$

5. True or False: $\underbrace{n^5 + n^5 + \ldots + n^5}_{\log n \text{ times}} + n^4 + n^3 + n^2 + n \in O(n^5)$

FALSE! it equals $n^5 \log n + n^4 + n^3 + n^2 + n.$

6. True or False: $\log(n^5) \in \omega(\log n)$

FALSE, $\log(n^5) = 5 \log n$, meaning $\log(n^5) \in O(\log n)$. Functions that are in $O(\log n)$ cannot be in $\omega(\log n)$. (Intuitively, if $f(n) \leq g(n)$, $f(n)$ cannot be strictly greater than $g(n)$.

## 4 Algorithm Analysis

Assume proc1($n$) runs in $O(1)$ time and proc2($n$) runs in $O(n^2)$ time. Neither changes $n$.

**Algorithm1($n$):**

```plaintext
begin
    I ← 1;
    while $I \leq n$ do :
        begin
            proc1($I$);
            $I$ + +;
        end
end
```

Time analysis: proc($I$) is called $n$ times. So $\underbrace{O(1) + O(1) + \ldots + O(1)}_{n \text{ times}} = n \cdot O(1) \in O(n)$.

**Algorithm2($n$):**

```plaintext
begin
    I ← 1;
    while $I \leq n$ do :
        begin
            proc1($I$);
            $I$ + +;
        end
        proc2($I$);
end
```

Time analysis: proc1 is called $n$ times, so $O(n)$. Plus, proc2 is called once on input of size $n$, so $O(n^2)$. $O(n) + O(n^2) = O(n^2)$.
Algorithm3\( (n) \):
\[
\text{begin} \\
\quad I \leftarrow 1; \\
\quad \text{while } I \leq 800 \text{ do :} \\
\quad \quad \text{begin} \\
\quad \quad \quad \text{proc2}(n); \\
\quad \quad \quad I ++; \\
\quad \quad \text{end} \\
\quad \text{end}
\]

Time analysis: proc2 is called on input of size \( n \) a constant (800) amount of times. So we have \( 800 \cdot O(n^2) \) which means Algorithm3 is in \( O(n^2) \).