CS 180 – Lecture 7

Note Title 9/9/2009

Announcements

- HW due tomorrow
- Lab 2 tomorrow
- Programming project #1 – out tonight
    - probably due in $1\frac{1}{2}$ – 2 weeks
    - pair project

Ch 3 - How to analyze running time?

So how?

time it

- disadvantage : computer matters
  compiler
  OS
  language
  input set matters

Counting primitive operations

Identify high-level primitive operations independent of language, compiler, OS, or computer

<u>Ex</u>: variable assignment/update
comparisons
branching
add
subtract
multiplication
⋮

Ex: (pseudocode to find max in an array)

→ write easily readable, language independant code

```
Algorithm arrayMax(A, n):
    Input: An array A of n ≥ 1 numbers
    Output: The maximum element of A
                      assignment operator
currentMax ← A[0]
for i ← 1 to n-1
    if currentMax < A[i] then
        currentMax ← A[i]
return currentMax
```

Advantage of pseudocode:

- independent of language
- easy to read & translate to <u>any</u> language

<u>Ex:</u> (in C++)

```
int arrayMax(int A[], int n) {
    int currentMax = A[0];
    for (int i = 1; i < n; i++)
        if (currentMax < A[i])
            currentMax = A[i];
    return currentMax;
```

Counting operations:

Algorithm arrayMax(A, n):
  Input: An array A of $n \geq 1$ numbers
  Output: The maximum element of A

```
currentMax ← A[0]                       ← 3
for i ← 1 to n-1                        2 + n-1 comparisons
    if currentMax < A[i] then           n-1 comparisons
        currentMax ← A[i]               ← between 0 & 2(n-1) operations
return currentMax                       1
```

$$\text{min: } 3+2+n-1+n-1+0+1 = 2n+4$$

$$\text{max: } 3+2+n-1+n-1+2(n-1)+1 = 4n+2$$

So how many operations in best (or worst) case?

$2n+4$ to $4n+2$

(see previous)

Average case versus worst case

→ Look at all possible inputs + average

→ analyze time for worst possible input

aveage : $3n+3$

worst case : $4n+2$

Asymptotic Notation

How important is <u>exact</u> number of computations?

In general, any primitive statement depends on a small number of low-level operations, independent of language or computer.

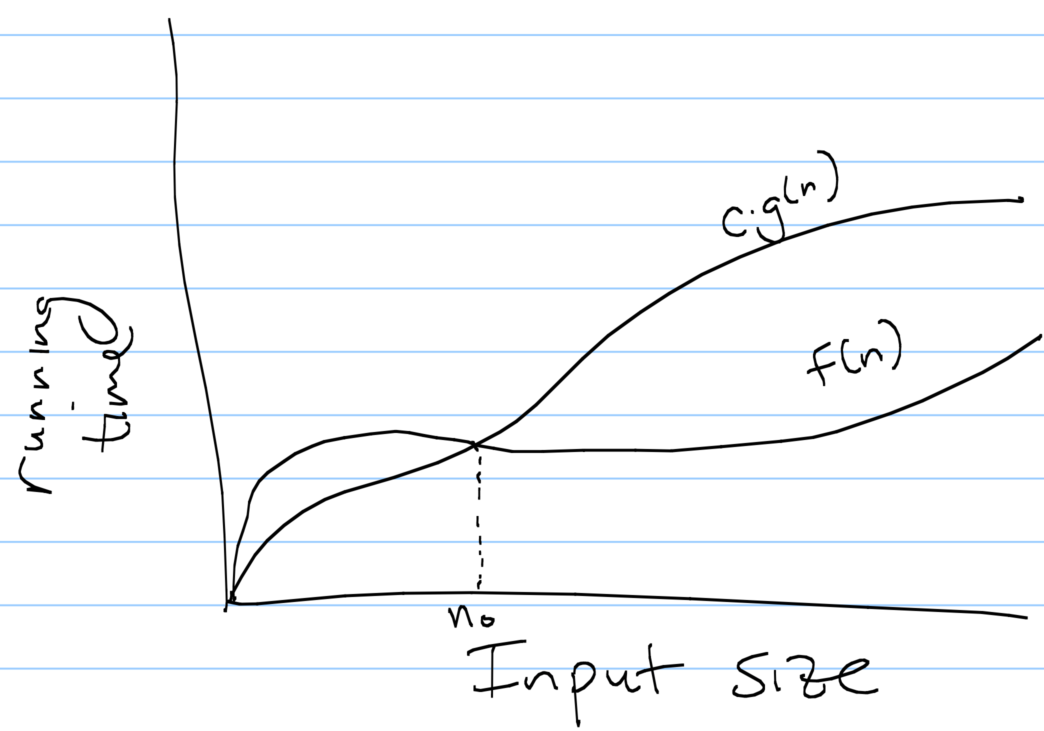
So we'll focus on big-picture, or how the running time grows in proportion to input size (usually n).

Formalize: Big-Oh notation

Let $f(n)$ and $g(n)$ be two functions from non-negative integers to reals.
We say $f(n)$ is $O(g(n))$ if there exists a constant c and integer $n_0 > 0$ such that $f(n) \le c \cdot g(n)$ for all $n \geq n_0$.

$f(n)$ is big-Oh of $g(n)$

Picture



__Ex:__

$\underline{4n+2}$ is $\underline{O(n)}$.

__Why?__ Find c & n_0:

Let $c=5$. Want $4n+2 < 5n$

Let $n_0 = 2$

if $n > n_0$, then $4n+2 < 5 \cdot n$

Let $c=100$ and $n_0=100$

$4n+2 < 100n$ if $n > 100$

Ex: running time of arrayMax is $O(n)$:

Algorithm arrayMax(A, n):
    Input: An array A of $n \geq 1$ numbers
    Output: The maximum element of A

```
currentMax ← A[0]
for i ← 1 to n-1
    if currentMax < A[i] then
        currentMax ← A[i]
return currentMax
```

Why? Just showed worst case running time is $O(n)$.

Ex: $20n^3 + 10n\log n + 5$

$O(n^3)$

Why?

$c = 35$ & $n_0 = 2$

$20n^3 + 10n\log n + 5 < 35n^3$

Any polynomial: $a_k n^k + a_{k-1} n^{k-1} + \dots + a_0$

$O(n^k)$

$25n^6 + \dots$

$O(n^6)$

<u>Ex</u>: $2^{100} = O(1)$

Let $c = 2^{100}$ & $n_0 = 1$

$$2^{100} \le 1 \cdot 2^{100}$$

p. 126 in book: Rules

Examples:

- If $d(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$, then $d(n)$ is $O(g(n))$.

- $\log n^c$ is $O(\log n)$ for any constant $c > 0$.

  $\log n^c = c \cdot \log n$

etc.

Useful things to remember:

- $\sum_{i=a}^{b} f(i) = f(a) + f(a+1) + \cdots + f(b)$

  (loops often produce these!)

- For any $n \geq 1$ and $0 < a \neq 1$:

  $$\sum_{i=0}^{n} a^i = 1 + a + \cdots + a^n = \frac{1-a^{n+1}}{1-a}$$

  and if $a < 1$, then $\sum_{i=0}^{\infty} a^i = \frac{1}{1-a}$

Another useful thing:

for any $n \geq 1$,

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

When might this come in handy?

2 nested loops

```
for i ← 1 to n
    for j ← 1 to n
        }
```

Logarithms (see p. 115)

- $\log_b(ac) = \log_b a + \log_b c$
- $\log_b(a/c) = \log_b a - \log_b c$
- $\log_b(a^c) = c \cdot \log_b(a)$
- $\log_b b^a = a$
- $b^{\log_c a} = a^{\log_c b}$
- $(b^a)^c = b^{ac}$
- $b^a b^c = b^{a+c}$

etc...