CS180 - Asymptotic Analysis

Note Title 9/7/2010

Announcements

- New Test file on website, so redownload!
- Don't forget comments on program 1 (~10%)
- New judging program for labs is working now
- Program 1 due Friday

Last time: How to measure speed of a program

Counting primitive operations

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

Counting operations:

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

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

1 operation

$n-1$ variable assignments & $n-1$ comparisons

1 comparison

1 variable assignment

1 memory access

(best case)

Sum: min: $1 + 2(n-1) + n-1 + 1 = 3n-1$

worst case: $(3n-1) + n-1 = 4n-2$

Asymptotic Notation (Ch. 3)

How important is exact number of computations?

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

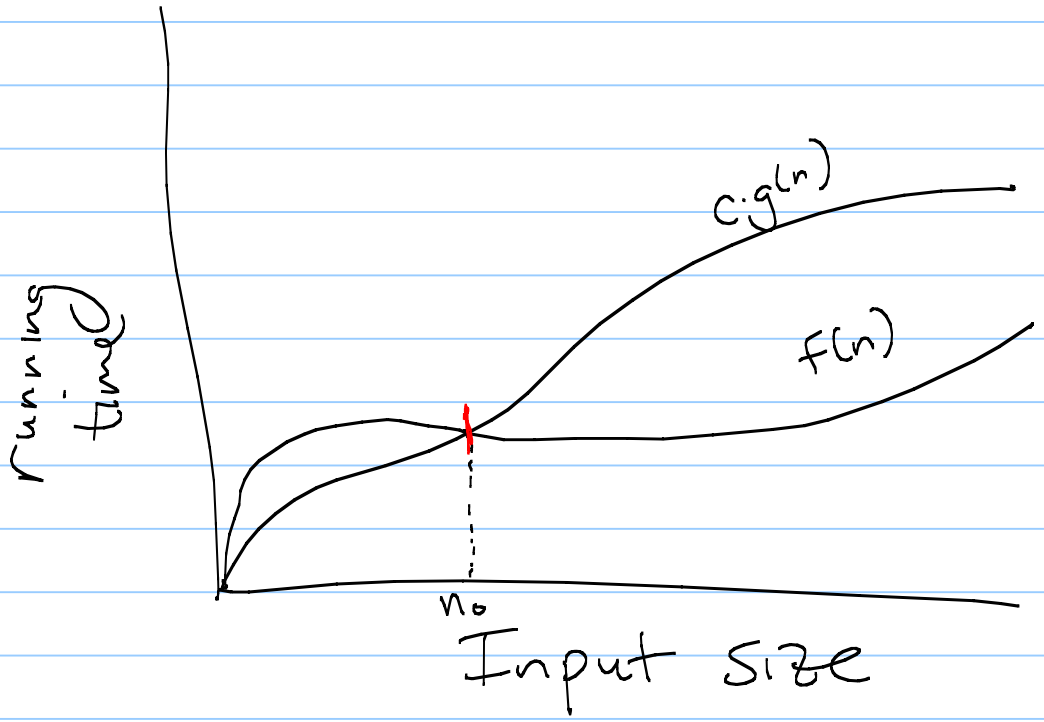
So we'll focus on big-picture, or how the running time grows in proportion to input size (usually n).

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 $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

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

Picture



Ex: $4n-2$ is $O(n)$. ($f(n)$, $g(n)$)

Why? Find c and n_0 s.t. $\forall n > n_0$,

$$4n-2 \leq c \cdot n$$

$$4n-2 \leq 4 \cdot n$$

Let $c = 4$ and $n_0 = 1$

$f(n) = 6n + 12$ is $O(n)$

Let $c = 7$ and $n_0 = 12$

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

linear time

```
Algorithm arrayMax(A, n):
    Input: An array A of n ≥ 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
```

We just counted $4n-2$ operations & showed $4n-2$ is $O(n)$!

Ex: $20n^3 + 10n\log_2 n + 5$

big-O? $O(n^3)$

Find c & n_0 s.t. $\forall n \geq n_0$,

$20n^3 + 10n\log n + 5 \leq c \cdot n^3$

Let $c = 20 + 10 + 5 = 35$

& $n_0 \geq 1$

So inequality holds

$\underbrace{20n^3}_{\leq 20n^3} + \underbrace{10n\log n}_{\leq 10n^3} + \underbrace{5 \cdot n^0}_{\leq 5n^3}$

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

is big-O of n^k.

Why?

$c = a_k + a_{k-1} + \cdots + a_0$

Ex: $2^{100} = O(1)$

Let $c = 2^{101}$ & $n_0 = 1$

You can choose any constant c.

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 $c > 0$.

$\log (n^5)$ is $O(\log n)$

$\|$

$5 \log n$

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 $n \geq 1$,

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

When might this come in handy?

for $i \leftarrow 1$ to n
  for $j = i$ to n
    addition

What is the running time of nested for loops?

$O(n^2)$

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...

Some more C++ items

Consider a function:

```
int min (int a, int b) {
    if (a<b)
        return a;
    else if (b<=a)
        return b;
}
```

Alternate:

```
int main(int a int b)
  { return (a<b? a : b); }
```

Seems handy...

Function templates:

this is a parameter list (only one here - called T)

```
template <typename T>
T min(T a, T b) {
    if (a<b)
        return a;
    else
        return b;
}
```

Important: Will work for any class, as long as "<" has been defined!

Class templates: a vector example

```
template <typename Object>
class BasicVector {
private:
    Object* a;          // array of elements
    int capacity;       // length of array a
public:
    BasicVector(int c=10) {   //constructor
        capacity = c;
        a = new Object[capacity]; //allocate storage
    }

    Object& elemAtRank(int r)  //access r^th element
        {return a[r];}
    :
```

Note: in C++, arrays are pointers!

Can always set an array using new,
or just put a pointer to first element.

Then pointer is address of first element,
so we can add to that number,
or just say pointer[index].

(Sec. 1.1.3)

Back to BasicVector: usage

```
BasicVector<int>    intvec(5); //vector of 5 ints
BasicVector<string> strvec(10); //vector of 10 strings

intvec.elementAtRank(3) = 8; //sets 4th element = 8
strvec.elementAtRank(7) = "hello"; //sets 8th elt = "hello"
```

Or even:

```
BasicVector<BasicVector<int> > myvec(5);
        //vector of 5 BasicVectors of integers

myvec.elementAtRank(2).elementAtRank(8) = 15;
        //  myvec[2][8] = 15
```