CS2100 - Bg-O + Algorithms

Note Title 9/19/2011

Announcements

- Lab due today
- HW due Sunday
- Next HW up soon

A debugging PSA:

Don't be afraid to check things

- cout to be sure where things break
  eg: cout << "here1" << endl;
- check all relevant variables
- ask for help developing "tough" cases

Algorithm Analysis

How do we compare two programs?

- features – what do they do?
- speed
- size – system requirements

Speed

How fast an algorithm runs can be very dependent on variables in the system.

Examples:

- RAM - system specs
- input presented
  - identical
  - range of tests
- programming language

Primitive Operations

As a way to compare algorithms in a generic way, we instead count primitive operations.

↑ add, initialize variable, subtraction, print, ...

In addition, we (generally) only analyze the worst possible running time.

Why? · absolute guarantee
· easier

Comparing

OK, so we have the worst case # of operations - usually a function of n.

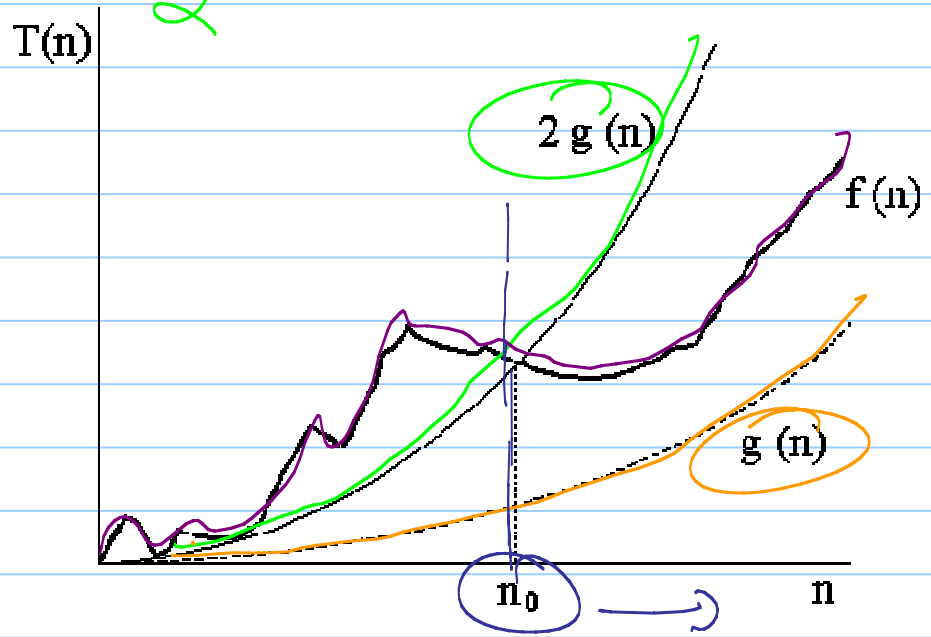
How to compare?

Big-O notation

Big-O

We say $f(n)$ is $O(g(n))$ if $\forall n > n_0$,

$\exists\ c > 0$ such that $f(n) \leq c \cdot g(n)$.



Ex: $5n$ is $O(n^2)$

$c = 5$: $\forall n > 5$, $5n \leq n^2$

Ex: $5 \cdot n$ is $O(n)$

$c = 6$ $\quad 5 \cdot n \leq 6 \cdot n$

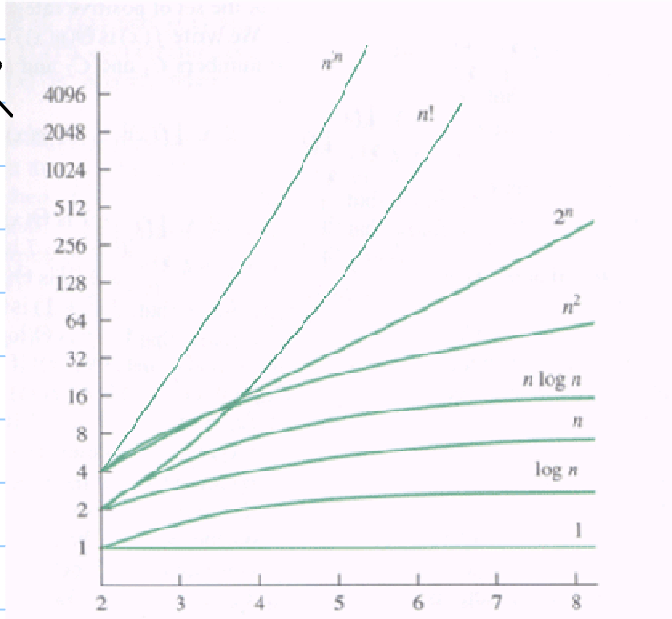
$n > 1$

Ex: $16n^2 + 52$ is $O(n^2)$

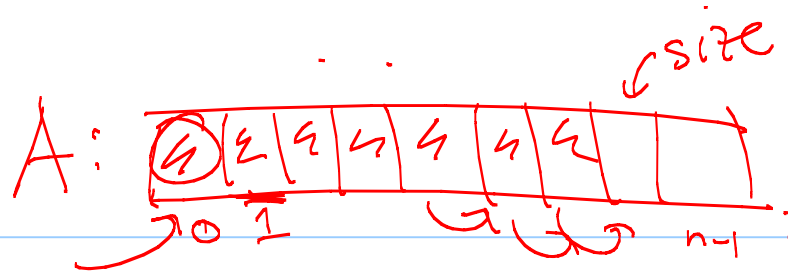
$c = 16 + 52$ ← chose inequality...

Functions we will use

1. $O(1)$ – constant time
2. $O(\log n)$ – logarithmic time
3. $O(n)$ – linear time
4. $O(n \log n)$
5. $O(n^2)$ – quadratic time
6. $O(n^3)$ – cubic time
7. $O(2^n)$ – exponential time



Algorithms



Claim: Inserting an element into the first spot in an array is $O(n)$ time.

declaring var = 1

```
for( int i = size; i > 0; i-- )
    A[i] = A[i-1];
A[0] = new thing;
```

comparison: once per loop, doing subtraction

$A[i] = A[i-1]$ ← 1 operation

2 op.

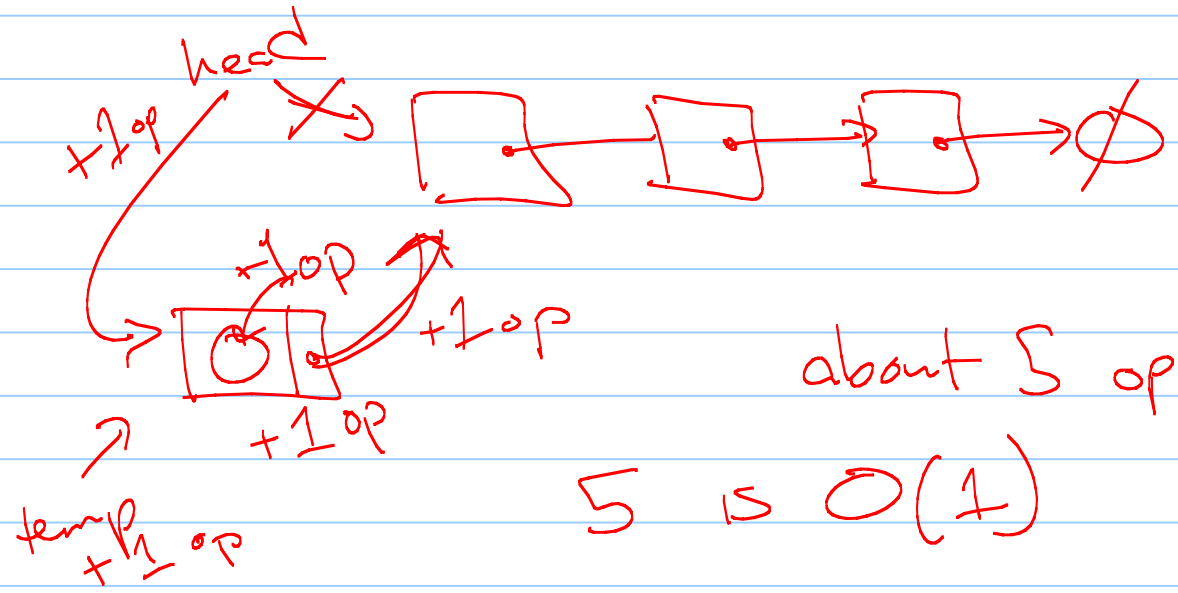
$$\text{total} = 1 + \sum_{i=1}^{size} (1+1+1) + 1$$

$$= \left(\sum_{i=1}^{size} 3\right) + 2 = 3 \cdot n + 2$$

$$= O(n)$$

(n = size of data)

Claim: Inserting at the beginning of a list is O(1) time.



about 5 operations

5 is O(1)

Common running times

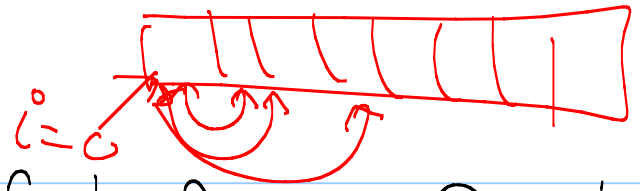
- A for loop which goes from $i=0$ to $n-1$ and reads to an array

```
for (int i=0; i<n; i++)
    cin >> array[i];
```

Analyze:

$$\sum_{i=0}^{n-1} (c) + 1 = O(n)$$

" $c \cdot n + 1$



Nested For loops : find if any 2 elements are identical

```
for (int i=0; i<n; i++)
    for (int j=i+1; j<n; j++)
        if (A[i] == A[j])
            cout << "Two items are the same" << endl;
```

Analyze:

$$\sum_{i=0}^{n}\left[\sum_{j=i+1}^{n} 1\right] = \sum_{i=0}^{n}(n-i)$$

$$= n+(n-1)+(n-2)+\cdots+1+0 = \sum_{i=0}^{n} i = \frac{n(n+1)}{2} = \binom{n}{2}$$

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

From here on out:

We'll analyze running time of the most common functions in every data structure.

Some will be easy:

Some harder:

(Note: Sometimes space too!)

Stack: a way to store a list of data

Ex: Web browser: store history for "back" button

Ex: Text editors: store previously used commands

The stack ADT:

Supports 2 main functions:

- push(e): add e to top of the stack
- pop(): remove e from the stack

Others

- top(): returns top element of the stack without removing it
- empty(): returns true if stack is empty
- size(): returns # of objects in the stack

The Standard template library

- Has iostream, string, etc.

- Also has basic data structures!
  (We'll be coding our own anyway.)

- See cplusplus.com for documentation...

Next time

How to implement our version?