CS314 - Big-O, induction & recurrences

Note Title 8/26/2013

Announcements

- See webpage for handouts / notes
- HW due next Wed. at start of class

Peasant multiplication

```
PEASANTMULTIPLY(x, y):
  prod ← 0
  while x > 0
      if x is odd
          prod ← prod + y
      x ← ⌊x/2⌋
      y ← y + y
  return p
```

Ex:

x	y	prod
123	456	0
→ 61	912	456
→ 30	1824	1368
→ 15	3648	—
7	7296	5016
3	14592	12312
1	29184	26904
0		+

- earliest written record is 1650 BC (& was already old!)
- still taught in Europe in late 1900's.

Why correct? (This is non-trivial!)

Key:

$$x \cdot y = \begin{cases} 0 & \text{if } x = 0 \\ \left\lfloor \frac{x}{2} \right\rfloor (y+y) & \text{if } x \text{ is even} \\ \left\lfloor \frac{x}{2} \right\rfloor (y+y) + y & \text{if } x \text{ is odd} \end{cases}$$

Harder : US constitution

> Representatives and direct Taxes shall be apportioned among the several States which may be included within this Union, according to their respective Numbers.... The Number of Representatives shall not exceed one for every thirty Thousand, but each State shall have at Least one Representative....

But how?

Today: Huntington-Hill method
- proposed in 1911
- adopted in 1941

(50 states, 435 representatives)

The algorithm:

```
ApportionCongress(Pop[1..n], R):
  PQ ← NewPriorityQueue
  for i ← 1 to n
      Rep[i] ← 1
      Insert(PQ, i, Pop[i]/√2)
      R ← R − 1

  while R > 0
      s ← ExtractMax(PQ)
      Rep[s] ← Rep[s] + 1
      Insert(PQ, s, Pop[s] / √(Rep[s](Rep[s]+1)))
      R ← R − 1

  return Rep[1..n]
```

A bad example:

BECOMEAMILLIONAIREANDNEVERPAYTAXES:
Get a million dollars.
Don't pay taxes.
If you get caught,
Say "*I forgot.*"

specific instructions!

Why?

In this class:

3 parts to <u>every</u> algorithm:

1. pseudocode

2. runtime analysis (mostly) - (big-O)

3. proof of correctness

+ (sometimes) 4. space

This week: why you should have paid attention in 135. (Math)

Topics:
- pigeonhole principle
- counting - combinations, permutations, ...
- probability
- proofs
- Logic
- graphs
- Sets

- big-O
- recursion + induction

Runtime:

What is big-O?

worst case running time

mathematical notion of upper bounding

Why use it?

Formal dfn:

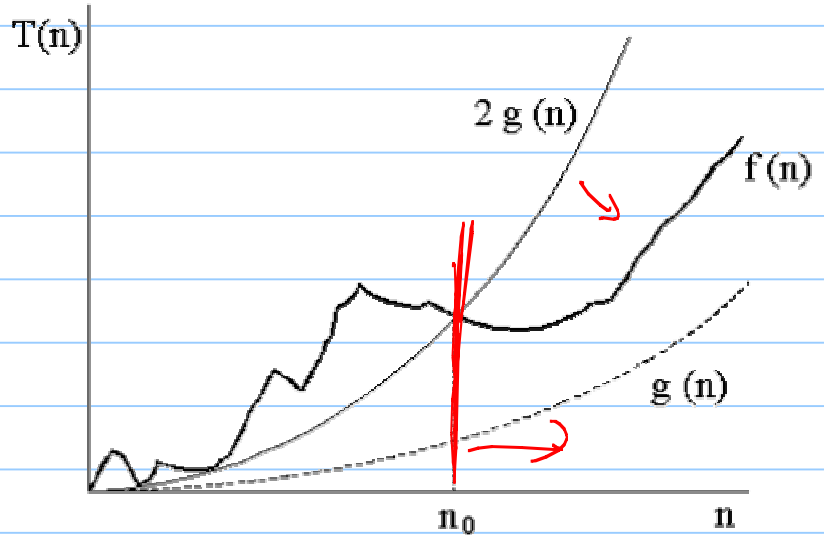
Let f & g be functions from $\mathbb{R} \to \mathbb{R}$ (or $\mathbb{Z} \to \mathbb{R}$). We say that:

$$f(n) = O(g(n))$$

if there exist constants C & n_0 such that

$$|f(n)| \leq C|g(n)|$$

for all $n > n_0$.



Ex: $f(x) = x^2 + 2x + 1$ is $O(x^2)$

proof: Need to find C and n_0

$$x^2 + 2x + 1 < x^2 + 2x^2 + x^2$$

(if $x > 0$)

$$= 4x^2$$

Let $C = 4$ & $n_0 = 1$

Then $x^2 + 2x + 1 \leq 4 \cdot x^2$

Thm: Let $f(x)$ be a polynomial,

So $f(x) = \sum_{i=0}^{n} a_i x^i = a_n x^n + a_{n-1} x^{n-1} + \cdots a_1 \cdot x + a_0$

where $a_0, a_1, \ldots, a_n \in \mathbb{R}$.

Then $f(x) = O(x^n)$.

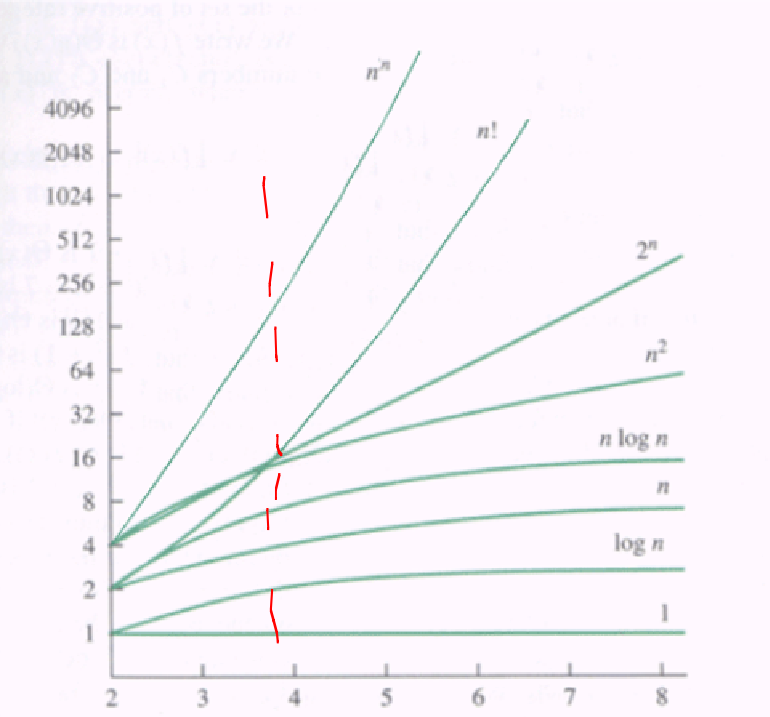
Central idea of proof: What c to use?

$$c = |a_n| + |a_{n-1}| + \cdots + |a_0|$$

Other useful functions:

(remember 180?)

$n^n >> n! >> 2^n \ldots$



Induction: Recursion's twin

A method of proving a statement which depends upon the statement holding for smaller values.

Can think of this as "automating" a proof:

true for $n = 1$

assume true for $< n$

Show also true at n

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

$= \sum_{i=1}^{n} \sum_{j=1}^{i} 1$

pf: base case: $n = 1$ $\quad \sum_{i=1}^{1} i = 1$

$\frac{(n+1)n}{2} = \frac{(1+1)1}{2} = 1$

IH: Assume for $k < n$, that $\sum_{i=1}^{k} i = \frac{(k+1)k}{2}$

IS: $\sum_{i=1}^{n} i = (1 + 2 + 3 + 4 + \cdots + (n-1) + n)$

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

$= \frac{n(n-1)}{2} + n = \frac{n(n-1)+2n}{2} = \frac{n(n-1+2)}{2}$ IH