Math 135 – Complexity of Algorithms

Note Title 2/26/2010

Announcements

- HW due Monday
- Next HW out Mon/Tues, & due after break!
- Planning next midterm for week of March 29 – April 2 (probably March 31)

Last time - pseudo code (Ch. 3.1)

We often use pseudo code to write down computer algorithms.

Common programming concepts:

- if statements
- loops
- variables
- functions or procedures
- input/output

Complexity of Algorithms

Comparing which algorithms are "better" can be tricky.

Issues: how long does it take?

problem is this varies from computer to computer
varies from language to language

- input also matters

So:

We define complexity in terms of the number of operations.

Usually, an operation is:

- add 2 things (or subtract or multiply)
- compare 2 things
- set a variable equal to something

But still - how do we compare?

Last time, saw 2 searching algorithms, linear search & binary search.

One is not always better.

# of comparisons

Find(36): | 36 40 58 100 101 125 |

LS: 1
BS: 3

Find(36): | 1 11 25 36 41 42 100 |

LS: 4
BS: 1

So how can we compare worst case performance?

Count the maximum # of operations

Bounding runtime in terms of input size = n.

Ex: What is worst case complexity of FindMax?

comparisons = $n-1$

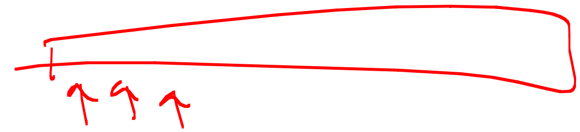
variable assignments:

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

worst case

```
FINDMAX(a_1, a_2, ..., a_n):
    max := a_1
    for i := 2 to n
        if max < a_i
            max := a_i
    return max
```

$O(n)$ time algorithm



Ex: What is worst case complexity of Linear Search?

```
LINEARSEARCH(x, a_1, ..., a_n):
- i := 1
  while (i ≤ n and x ≠ a_i)
->      i := i+1
  if i ≤ n
->      location := i
  else
->      location := 0
```

$\leq 2n+1$ comparisons

$\leq 1+n-1+1 = n+1$ variable assignments

$\leq n-1$ additions

$O(n)$ time algorithm

Ex: What is complexity of Bubble Sort?

Count # of comparisons

$$\sum_{i=1}^{n-1} \underbrace{\sum_{j=1}^{n-i} 1}_{\underbrace{1+1+1+\cdots+1}_{n-i}} = \sum_{i=1}^{n-1} (n-i) = (n-1)+(n-2)+(n-3)+\cdots+1$$

$$= \sum_{i=1}^{n-1} i = \boxed{\Theta(n^2)}$$

BUBBLE SORT$(a_1 \dots a_n)$:
  for $i := 1$ to $n-1$
    for $j := 1$ to $n-i$
      if $a_j > a_{j+1}$
        swap a_j and a_{j+1}

Ex: What is complexity of insertion sort?

worst case # of comparisons

$$\sum_{j=2}^{n} (j-1)$$

$$= 1 + 2 + 3 + \dots + n-1$$

$$= \Theta(n^2)$$

```
INSERTION SORT (a_1 .. a_n):
for j := 2 to n
    i := 1
    while a_j > a_i
        i := i+1
    temp := a_j
    for k := j to i+1
        a_k := a_{k-1}
    a_i := temp
```

Why is big-O a good justification?

TABLE 2 **The Computer Time Used by Algorithms.**

Problem Size	*Bit Operations Used*					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	3×10^{-9} s	10^{-8} s	3×10^{-8} s	10^{-7} s	10^{-6} s	3×10^{-3} s
10^2	7×10^{-9} s	10^{-7} s	7×10^{-7} s	10^{-5} s	4×10^{13} yr	*
10^3	1.0×10^{-8} s	10^{-6} s	1×10^{-5} s	10^{-3} s	*	*
10^4	1.3×10^{-8} s	10^{-5} s	1×10^{-4} s	10^{-1} s	*	*
10^5	1.7×10^{-8} s	10^{-4} s	2×10^{-3} s	10 s	*	*
10^6	2×10^{-8} s	10^{-3} s	2×10^{-2} s	17 min	*	*

$O(n^2)$ sorting alg $\rightarrow 2n^2$

versus $O(n \log n)$ sorting alg $\rightarrow 16\, n \log n$

The Halting Problem

Q: Can we write a program which accepts as input another program & input, then decides if the program will run forever or halt on that input.

(So if it contains infinite loop, will run forever, for example, & our program will say that.)

Note: Our program can't just run the input program.

Why?

Thm: The halting problem is undecidable.
(that is, no program to solve it can exist!)

pf: by contradiction