Math 135 - Algorithms - Ch. 3.1, 3.3

Note Title 10/13/2010

Announcements

- HW due in 1 week
- No class next Monday

Last time

Basic programing structures:

- variables

  letters or words that store values.

  Ex: $a_1, a_2, \ldots, a_n$, x, number

- if statements

  Ex: if $(x > 1)$
  $x := x + 1$
  else
  $x := x - 1$

- for loops

$x := 0$
for ($i := 1$ to n)
    $x := x + i$

Suppose $n = 5$
$i = \cancel{1}\ \cancel{2}\ \cancel{3}\ \cancel{4}\ 5$
$x = \cancel{0}\ \cancel{1}\ \cancel{3}\ \cancel{6}\ \cancel{10}\ 15$

- while loops

$i := 0$
while ($i < n$)
    $i := i + 1$
    $x = x + i$

← repeat as long as $i < n$

While loop example:

Compute $\underbrace{\lceil \log_2 n \rceil}$ (given a number n).

power := 0
number := n

while (number > 1)
  number := number / 2
  power := power + 1

$n = 16$

← number = $\cancel{16}$ $\cancel{8}$ $\cancel{4}$ $\cancel{2}$ 1

power = $\cancel{0}$ $\cancel{1}$ $\cancel{2}$ $\cancel{3}$ 4

Pseudocode

Our goal is to show how to write a program (without actually programming).

Remember, a program is a sequence of instructions to tell a computer how to solve a problem.

Complexity

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

count (worst case) # of operations

Ex: What is worst case complexity of FindMax?

FINDMAX$(a_1, a_2, \ldots, a_n)$:

```
max := a_1
for i := 2 to n
    if max < a_i
        max := a_i
return max
```

1

repeat n-1 times

2

n-1 variable assignment

\# operations =

$$1 + (n-1)(2) + (n-1)$$

$$= 1 + 2n - 2 + n - 1$$

$$= \boxed{3n - 2} = O(n)$$

$a_1 \quad a_2 \quad a_3 \quad \cdots \quad a_n$

x

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
```

1

2 comparisons

repeats n times

1

1 comparison

1 of these happens

$$1 + n(2+1) + 1 + 1$$

$$= 3n + 3 = O(n)$$

What is the complexity of BubbleSort?

BUBBLE SORT$(a_1 \ldots a_n)$:

→ for $i := 1$ to $n-1$ ⎤ for loops translate to summations

→ for $j := 1$ to $n-i$ ⎦

4 [if $(a_j > a_{j+1})$ ← 1 comparison

swap a_j and a_{j+1} ← 3 operations

$$\sum_{i=1}^{n-1} \underbrace{\sum_{j=1}^{n-i} 4}_{\text{"} \underbrace{(4+4+4+\cdots+4)}_{n-i} = 4(n-i)} = \sum_{i=1}^{n-1} 4(n-i)$$

$n-(n-1)$

$$\sum_{i=1}^{n-1} 4(n-i) = \underbrace{\sum_{i=1}^{n-1} 4n}_{= 4n(n-1)} - \underbrace{\sum_{i=1}^{n-1} 4i}_{= 4\sum_{i=1}^{n-1} i}$$

$$4\sum_{i=1}^{n-1} i = \frac{4n(n-1)}{2}$$

$$= 4n(n-1) - 2n(n-1) = \underbrace{2n(n-1)}_{= 2n^2 - 2n = O(n^2)}$$

method #2

$$4\sum_{i=1}^{n-1}(n-i) = 4\left((n-1)+(n-2)+(n-3)+\dots+1\right) = 4\sum_{i=1}^{n-1} i$$

$$= 4\,\frac{(n-1)(n)}{2}$$

$a_1 \quad a_2 \; \dots \; a_j \; \dots \; a_n$

What is complexity of insertion sort?

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

$$= \sum_{j=2}^{n} (2j+3)$$

$$\vdots$$

$$\shortparallel$$

$$O(n^2)$$

repeats $n-1$ times

total j times

INSERTION SORT $(a_1 .. a_n)$:

```
for j := 2 to n
    1 → i := 1
        while a_j > a_i
            i := i+1
    1 → temp := a_j
        for k := j to i+1
            a_k := a_{k-1}
    1 → a_i := temp
```

Why is big-O a good idea?

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	*	*

$n \log n$ versus $5\, n \log n$

10×10^{-2}

So to analyze an algorithm's running time, we often use big-O analysis.

Rather than $16n-12$, just say $O(n)$.