Math 135 - Algorithm Analysis

Note Title 10/4/2012

Announcements

- HW is up (?) over big-O and algorithms

Complexity

We define complexity or running time of an algorithm in terms of the number of operations.

Example of a single operation:

- adding two numbers
- loading or storing a value
  a = ...
- multiplication

worst case # of

Why count ^ operations?

Generally, the other way to compare the speed of 2 algorithms is benchmarking.

But this can vary even with identical algorithms.

Ex:
- system & hardware
- network
- programming language
- input

Here, n = size of input.

In addition, we usually use big-O.

Reason: Taking 3 versus 5 operations can vary between processors & programming languages.

In a loop, this is the difference between $3n$ and $5n$.

In reality, this is for the same algorithm!

So big-O lets us "hide" these details.

Ex: What is complexity of FINDMAX? (in 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
```

1 → max := a_1 : 1

for i := 2 to n : 1

if max < a_i : 1

max := a_i : 1

repeats n-1 times

return max : 1

$$1 + (n-1)[1+1+1] + 1 = 3n - 1 \text{ operations}$$

$$\left[\sum_{i=2}^{n} 3\right]$$

$$= \boxed{O(n)}$$

Ex: Linear Search?

```
LINEAR SEARCH (x, a_1, a_2, ..., a_n):
    i := 1
    while (i ≤ n and x ≠ a_i)
        i := i + 1
    if i ≤ n
        location := i
    else
        location := 0
    return location
```

1 → i := 1

3 → while (i ≤ n and x ≠ a_i)

1 ← i := i + 1

1 ← if i ≤ n

1 ← location := i

or

1 ← location := 0

n times

$1 + n(4) + 1 + 1 + 1 = O(n)$

Ex: Bubble Sort?

```
BUBBLESORT (a_1, ..., a_n):
    for i := n down to 2:
        for j := 1 to i-1:
            if a_j > a_{j+1}
                temp := a_j
                a_j := a_{j+1}
                a_{j+1} := temp
```

1

3

$i-1$

repeats $n-1$

$$\sum_{i=2}^{n} \left[\sum_{j=1}^{i-1} 5 \right] = \sum_{i=2}^{n} 5(i-1) = 5 \sum_{i=2}^{n} (i-1)$$

$$5 \sum_{i=2}^{n} (i-1) = 5 \left[\sum_{i=2}^{n} i - \sum_{i=2}^{n} 1 \right]$$

$$= 5 \left[\left(\frac{n(n+1)}{2} - 1 \right) - (n-1) \right]$$

$$= \boxed{O(n^2)}$$

Ex: Insertion Sort?

```
INSERTIONSORT(a_1, ..., a_n):
for j := 2 to n
    i := 1
    while a_j > a_i                 <= j times
        i := i+1
    temp := a_j                     repeats n-1 times
    for k := j to i+1               j - i times
        a_k := a_{k-1}
    a_i := temp
```

$$\sum_{j=2}^{n} \left[\sum_{i=1}^{j} 6 \right]$$

$$= \sum_{j=2}^{n} 6j$$

$$= 6 \sum_{j=2}^{n} j$$

$$= 6 \left[\frac{n(n+1)}{2} - 1 \right]$$

$$= 3n^2 + 3n - 1 = \boxed{O(n^2)}$$

Again, why big-O?

Rather than worrying about $6n - 64$, "just" say $O(n)$.

Worst-Case Time Complexity: The Big Picture

- On a computer running one billion operations per second ...

Input Size (n)	Time Complexity				
	n	$n \log_2 n$	n^2	n^3	2^n
10	$< .001$ second	$< .001$ second	$< .001$ second	$< .001$ second	$< .001$ second
20	$< .001$ second	$< .001$ second	$< .001$ second	$< .001$ second	.001 second
30	$< .001$ second	$< .001$ second	$< .001$ second	$< .001$ second	1 second
50	$< .001$ second	$< .001$ second	$< .001$ second	$< .001$ second	13 days
100	$< .001$ second	$< .001$ second	$< .001$ second	.001 second	4×10^{11} centuries
1000	$< .001$ second	$< .001$ second	.001 second	1 second	4×10^{282} centuries
100,000	$< .001$ second	.002 second	10 seconds	11.57 days	–
one million	.001 second	.02 second	1.67 minutes	32 years	–
ten million	.01 second	0.24 second	1.2 days	317 centuries	–
one billion	1 second	30 seconds	32 years	4×10^8 centuries	–
100 billion	1.67 minutes	1 hour	3171 centuries	4×10^{14} centuries	–