CMSC 341-04
Lecture 3

Announcements

Project 1
Project Preview tonight

Mileage Example

Problem:

John drives his car, how much gas does he use?

Complexity

How many resources will it take to solve a problem of a given size?

- time
- space

Expressed as a function of problem size (beyond some minimum size)

- how do requirements grow as size grows?

Problem size

- number of elements to be handled
- size of thing to be operated on

Growth Functions

Constant

$f(n) = c$

ex: getting array element at known location
trying on a shirt
calling a friend for fashion advice

Linear

$f(n) = cn$ [+ possible lower order terms]

ex: finding particular element in array (sequential search)
trying on all your shirts
calling all your n friends for fashion advice

Growth Functions (cont)

Quadratic

$f(n) = cn^2$ [+ possible lower order terms]

ex: sorting all the elements in an array (using bubble sort)
trying all your shirts (n) with all your ties (n)
having conference calls with each pair of n friends

Polynomial

$f(n) = cn^k$ [+ possible lower order terms]

ex: looking for maximum substrings in array
trying on all combinations of k separates (n of each)
having conferences calls with each k-tuple of n friends

Growth Functions (cont)

Exponential

f(n) = c^n [+ possible lower order terms

ex: constructing all possible orders of array elements

Logarithmic

f(n) = logn [+ possible lower order terms]

ex: finding a particular array element (binary search)

trying on all Garanimal combinations

getting fashion advice from n friends using phone tree

Asymptotic Analysis

What happens as problem size grows really, really large? (in the limit)

- constants don't matter
- lower order terms don't matter

Analysis Cases

What particular input (of given size) gives worst/best/average complexity?

Mileage example: how much gas does it take to go 20 miles?

- Worst case: all uphill
- Best case: all downhill, just coast
- Average case: "average terrain"

Cases Example

Consider sequential search on an unsorted array of length n, what is time complexity?

Best case:

Worst case:

Average case:

Complexity Bounds

Upper bound (big O):

$T(n) = O(f(n))$ if

$T(n) \leq cf(n)$ for some constants c, n_0 and $n \geq n_0$

Lower bound (omega):

$T(n) = \Omega(g(n))$ if

$T(n) \geq cg(n)$ for some constants c, n_0 and $n \geq n_0$

"Exact" bound (theta):

$T(n) = \Theta(h(n))$ if

$T(n) = O(h(n))$ and $T(n) = \Omega(h(n))$

"Greater" bound (little o):

$T(n) = o(p(n))$ if

$T(n) = O(p(n))$ and $T(n) \neq \Theta(p(n))$

Simplifying Assumptions

1. If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$
2. If $f(n) = O(kg(n)$ for any $k > 0$, then $f(n) = O(g(n))$
3. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$,
   then $f_1(n)+f_2(n) = O(\max(g_1(n),g_2(n)))$
4. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$,
   then $f_1(n)f_2(n) = O(g_1(n)\, g_2(n))$

Example

Code:

```
a = b;
```

Complexity:

Example

Code:

```
sum = 0;
for (i=1; i <=n; i++)
    sum += n;
```

Complexity:

Example

Code:

```
sum = 0;
for (j=1; j<=n; j++)
    for (i=1; i<=j; i++)
          sum++;
for (k=0; k<n; k++)
    A[k] = k;
```

Complexity:

Example

Code:

```
sum1 = 0;
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
          sum1++;
```

Complexity:

Code:

```
sum2 = 0;
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
          sum2++;
```

Complexity:

Example

Code:

```
sum1 = 0;
for (k=1; k<=n; k*=2)
    for (j=1; j<=n; j++)
          sum1++;
```

Complexity:

Code:

```
sum2 = 0;
for (k=1; k<=n; k*=2)
    for (j=1; j<=k; j++)
          sum2++;
```

Complexity: