Outline

Announcements:

- HW1 will be online today; Due: Tue, 9/25, 1:00 pm at 281 lockbox in 2240 EECS

Last time:

- Operation count
- FindMax, ComputeRank

Today:

- ComputeRank in $N \log N$ time
- BubbleSort
- Review of asymptotic analysis
- Empirical performance evaluation

Sugih Jamin (jamin@eecs.umich.edu)
Algorithm Design

Typical approach:

- define problem to be solved
- understand its complexity
- decompose it into smaller subtasks

Creative refinements:

- **think outside the box:**
  exploit any particular characteristics of the workload or problem
- don’t lose the forest for the tree:
  find the real performance bottleneck of the whole program,
  note coding cost
- optimize for the common case
Example of Not Exploiting Characteristics: Bubble Sort v.1

Sort array $a[20,30,15,10,35,25]$ by
“bubbling” the largest element to the right
See Fig. 15.3 of Preiss

How many CMPs does it take?
Example of Exploiting Characteristics: Bubble Sort v.2

Sort array $a[20,30,15,10,35,25]$ by “bubbling” the largest element to the right

Stop program as soon as array is sorted
(How do you know that the array is sorted?)

How many CMPs does it take in the worst case?
Taking Advantage of Workload Characteristics

What if array is already sorted initially?

- bubble1 still takes \( \frac{N(N-1)}{2} \) CMPs
- bubble2 takes \( N-1 \) CMPs

Implication: performance of a given alg. depends on the characteristics of the offered workload (initial values, input)

Conversely, depending on the expected workload, some algorithm/data structure may be more suitable than others
Algorithm Design

Typical approach:

- define problem to be solved
- understand its complexity
- decompose it into smaller subtasks

Creative refinements:

- think outside the box: exploit any particular characteristics of the workload or problem
- don’t lose the forest for the tree: find the real performance bottleneck of the whole program, note coding cost
- optimize for the common case
Don’t Lose the Forest . . .

Operations useful in maintaining an address book:

- search for a name
- insert a new name
- delete a name

How do you do insertion and deletion on an array?

How much does each operation cost, on average, for sorted and unsorted array?

Which algorithm to use (shall we keep the array sorted)?

Sugih Jamin (jamin@eecs.umich.edu)
The Language of Algorithm Analysis

How do you convince your boss that your solution is better than an existing solution?
Asymptotic Algorithm Analysis

An algorithm with complexity $f(n)$ (e.g., $f(n) = n$, $f(n) = n^2$, etc.), is said to be not slower than another algorithm with complexity $g(n)$ if $f(n)$ is bounded by $g(n)$ for large $n$ (See Fig. 4.1)

Commonly written as $f(n) = O(g(n))$ (read: $f(n)$ is big-Oh $g(n)$), a.k.a. the asymptotic (or big-Oh) notation

Definitions:

$f(n) = O(g(n))$ iff $\exists c > 0, n_o \geq 0 \mid \forall n, n \geq n_o, f(n) \leq c \cdot g(n)$,

$O(g(n)) = \{f(n) : \exists c > 0, n_o \geq 0 \mid \forall n, n \geq n_o, 0 \leq f(n) \leq c \cdot g(n)\}$,

and $c$ is a constant, i.e., it doesn’t change with $n$

So more accurately: $f(n) \in O(g(n))$
But NOT $f(n) \leq O(g(n))$
Big-Oh: Example

Algorithm A has time complexity $f(n) = 8n + 128$
Algorithm B has time complexity $g(n) = n^2$

Can we say that $f(n) = O(g(n))$?
Is $8n + 128 \leq c \cdot n^2$?

Let $c = 1$, clearly for $n = 8$, $f(n) > g(n)$.
At what value of $n_o$ is $g(n) > f(n)$, $\forall n \geq n_o$?
How about if $c = 2$ or $c = 4$? See Fig. 3.1.

Sugih Jamin (jamin@eecs.umich.edu)
Big-Oh: Summary

As long as there is a $c > 0$, and an $n_o \geq 0$, such that $c \cdot g(n) \geq f(n)$, for all $n \geq n_o$, you can say $f(n) = O(g(n))$.

In this case, $8n + 128 = O(n^2)$.

In other words, we only care about LARGE $n$, it doesn’t matter what $c$ is.

Also, asymptotically, $n^2 + k = O(n^2)$, why?

(Obviously, $c$ cannot be $10^{100}$ (one googol, the conjectured upper bound on the number of atoms in the observable universe)!)
Big-Oh: Summary (cont)

Big-Oh is NOT an exact notation

It compares the performance of two algorithms for LARGE input sizes

Big-Oh analysis is *language independent*
The Common Case: Empirical Performance

What if $n_o > \text{average } n$ or common case $n$?

To determine the common case performance, given known workload, run empirical performance measurement.

Note that common case performance is not necessary average case performance. (Why not?)

Pay attention to:

- system speed
- system load
- compiler optimization
Experiment Setup

Factors involved in running an empirical performance study:

- workload generator: does it generate realistic common cases?
- reducing system variability:
  - use the same compiler
  - use the same machine
  - run experiment around the same time of day
    (or use unloaded machine)
Experiment Setup (cont)

- call `gettimeofday()` before and after code of interest for timing information:

```
gettimeofday(&start,NULL);
for (i = 0; i < N; i++) {
    run_test_code();
}
gettimeofday(&end,NULL);
timing = (end-start)/N; /* PSEUDO-CODE! */
```

- repeat experiment several times with the same input and take the average or minimum.

Empirical analysis is also useful for complex algorithm or large software