CSE 373: Asymptotic Analysis

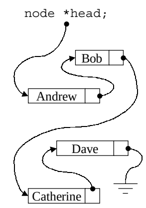
book chapter 2

Pete Morcos
University of Washington
3/31/00

http://www.cs.washington.edu/education/courses/cse373/00sp

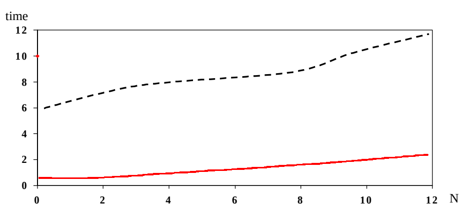
Quick Example

- Ops for linked lists:
  - add(char *newname)
  - remove(node *node_to_kill)
  - find(char *searchname)
  - removeAll(node *head_of_list)
  - getNext(node *current_node)
  - getPrev(node *current_node)
- What are the costs?

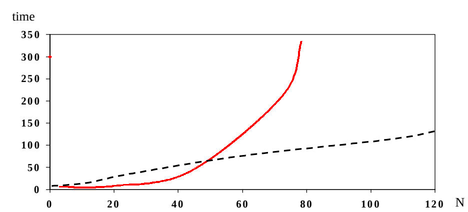


Picking an algorithm

- Which one is the best choice?



As N grows...



Asymptotic Behavior

- We're interested in the performance as $N \to \infty$, not the fluctuations at small N
- Given functions $T_1(N)$, $T_2(N)$, we need a way to decide which is the better choice
- Asymptotic behavior is most important
- Lower order behavior might matter in practice, especially if you are sure that small N will be common

Big-Oh Notation

- Formally:
  - $T(n) = O(f(n))$ iff. there are positive constants c and n_0 such that $T(n) \le c \cdot f(n)$ for all $n \ge n_0$
    - $\log n$, n, $2000\,n + \log n$ all $= O(n)$
  - $T(n) = \Omega(f(n))$ iff. there are positive constants c and n_0 such that $T(n) \ge c \cdot f(n)$ for all $n \ge n_0$
    - n^2, 2^n, $0.000001 \cdot n^{1.5}$ all $= \Omega(n)$
  - $T(n) = \Theta(f(n))$ iff. $T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$
- Can ignore constant factors. In sums, largest term overrides the rest (e.g. $O(n^2 + n \log n + n) = O(n^2)$)

Common Growth Rates

constant:	O(1)
"log log":	O(log(log *n*))
logarithmic:	O(log *n*)
"log squared":	O(log2 *n*)
linear:	O(*n*)
"n log n":	O(*n* · log *n*)
quadratic:	$O(n^2)$
cubic:	$O(n^3)$
exponential:	$O(2^n)$

$\left.\begin{array}{l} O(n) \\ O(n \cdot \log n) \\ O(n^2) \\ O(n^3) \end{array}\right\}$ polynomial time

Doing the analysis

- Treat all sequences of basic statements as O(1)
  - Even if it does 1,000,000 things, as long as that 1,000,000 is a constant and not a function of *n*, it's O(1)
- Conditionals: max of the alternatives
- Loops: if body is O(f(*n*)), loop is O(#iters*f(*n*))
- Function calls: not a single statement! Check each one to see if it depends on *n*.
- Recursive calls: trickier, depends on how much progress each call makes

Example

```
for (i=0; i<n; i++)
        for (j=0; j<i; j++)
                printf("hello\n");
```

- Outer loop is easy, O(n) iterations
- Inner loop changes each time!
- What is overall cost?

- How about:

```
for (i=n; i>=1; i/=2)
        for (j=0; j<i; j++)
                printf("hello\n");
```


Analyzing Recursion

- Consider a function to add an array recursively:
  - `int add() { return` *`first-element`* `+ add(`*`rest-of-array`*`) }`
  - Addition is O(1). What is cost of recursive call?
- We can say that the time to add, T(n), is:
  - O(1) if n = 1
  - T(n-1) + O(1) if n > 1
- Obviously, T(n) = T(n-1) + O(1) = O(n)
- This is called a *recurrence relation.*

Recurrence Relations

- Commonly seen relations are
  - $T(n) = T(n-1) + \Theta(1)$ — $O(n)$
  - $T(n) = T(n-1) + \Theta(n)$ — $O(n^2)$
  - $T(n) = T(n/2) + \Theta(1)$ — $O(\log n)$
  - $T(n) = T(n/2) + \Theta(n)$ — $O(n \log n)$
- Note: these formulas are sensitive to constants.
  - $T(n) = 9\,T(n/3) + \Theta(n)$ is $O(n^2)$, not $O(n \log n)$!
- We'll see this again later in the course; you'll only need to know a few specific examples.

Summary

- Usually care about asymptotic behavior
  - Low-*n* behavior can be important in practice
- Analyze both time and space costs this way
- Can get different results depending on whether you consider
  - best case
  - worst case
  - average case
  - most common case