# Growth of Functions
# & Complexity of Algorithms

CS280
Fall 2002

## How Do You Choose an Algorithm?

- Suppose you have two possible algorithms or data structures that basically do the same thing; which is better?
  - Faster?
  - Less space?
  - Easier to code?
  - Easier to maintain?
  - Required for homework?
- How do we measure the first two?

2

## Sample Problem: Searching

- Determine if a *sorted* array of integers contains a given integer
- 1st solution: Linear Search (check each element)

```
static boolean find (int[ ] a, int item) {
    for (int i = 0; i < a.length; i++) {
            if (a[i] == item) return true;
            }
    return false;
    }
```

- 2nd solution: Binary Search

```
static boolean find (int[ ] a, int item) {
    int low = 0;
    int high = a.length - 1;
    while (low <=  high) {
           int mid = (low+high)/2;
           if (a[mid] < item)
                      low = mid+1;
           else if (item < a[mid])
                      high = mid - 1;
           else return true;
           }
    return false;
    }
```

3

## Linear Search vs. Binary Search

- Which one is better?
  - Linear Search is easier to program
  - But Binary Search is faster… isn't it?
- How do we measure to show that one is faster than the other
  - Experiment?
  - Proof?
  - But which inputs do we use?
- Simplifying assumption #1: Use the *size* of the input rather than the input itself
  - For our sample search problem, the input size is *n* where *n-1* is the array size
- Simplifying assumption #2: Count the number of "*basic steps*" rather than computing exact times

4

## One Basic Step = One Time Unit

- Basic step:
  - input or output of a scalar value
  - accessing the value of a scalar variable, array element, or field of an object
  - assignment to a variable, array element, or field of an object
  - a single arithmetic or logical operation
  - method invocation (not counting argument evaluation and execution of the method body)
- For a conditional, we count number of basic steps on the branch that is executed
- For a loop, we count number of basic steps in loop body times the number of iterations
- For a method, we count number of basic steps in method body (including steps needed to prepare stack-frame)

5

## Runtime vs. Number of Basic Steps

- But isn't this cheating?
  - The runtime is not the same as the number of basic steps
  - Time per basic step varies depending on computer, on compiler, on details of code…
- Well… yes, it is cheating in a way
  - But the number of basic steps is *proportional* to the actual runtime
- Which is better?
  - $n$ or $n^2$ time?
  - 100 $n$ or $n^2$ time?
  - 10,000 $n$ or $n^2$ time?
- As n gets large, multiplicative constants become less important
- Simplifying assumption #3: Multiplicative constants aren't important

6

## Using Big-O to Hide Constants

- Roughly, f(n) is O(g(n)) means that f(n) grows like g(n) or slower

Definition: O(g(n)) is a *set*;
f(n) is a member of this set if and only if there exist constants C and k such that
$|f(n)| \leq C\,|g(n)|$, for all n>k

- We *should* write $f(n) \in O(g(n))$
- But by convention, we write f(n) = O(g(n)) or f(n) is O(g(n))

Claim: $n^2 + n$ is $O(n^2)$

We know $n < n^2$ for n >1

So $n^2 + n < 2\,n^2$ for n >1

So by definition,
$n^2 + n$ is $O(n^2)$
for C=2 and k=1

7

## Big-O Examples

Claim: 100 n + log n is O(n)

We know log n < n for n > 1
(because $n < 2^n$)

So 100 n + log n < 101 n
for n > 1

So by definition,
100 n + log n is O(n)
for C=101 and k=1

Claim: $\log_B n$ is O(log n)

Let k = log n

Then $n = 2^k$ and (the subscripts are too messy; switch to board)

Question: Which grows faster: sqrt(n) or log n?

8

## Simple Big-O Examples

- Let $f(n) = 3n^2 + 6n - 7$
  - Claim f(n) is $O(n^2)$
  - Claim f(n) is $O(n^3)$
  - Claim f(n) is $O(n^4)$
  - …
- g(n) = 4n log n + 34 n – 89
  - Claim g(n) is O(n log n)
  - Claim g(n) is $O(n^2)$
- $h(n) = 20 * 2^n + 40$
  - Claim h(n) is $O(2^n)$
- a(n) = 34
  - Claim a(n) is O(1)
- Only the *leading* term (the term that grows most rapidly) matters

9

## Problem-Size Examples

- Suppose we have a computing device that can execute 1000 operations per second; how large a problem can we solve?

| Complexity | 1 second | 1 minute | 1 hour |
|---|---|---|---|
| n | 1000 | 60,000 | 3,600,000 |
| n log n | 140 | 4893 | 200,000 |
| $n^2$ | 31 | 244 | 1897 |
| $3n^2$ | 18 | 144 | 1096 |
| $n^3$ | 10 | 39 | 153 |
| $2^n$ | 9 | 15 | 21 |

10

## Commonly Seen Time Bounds

| | | |
|---|---|---|
| O(1) | constant | excellent |
| O(log n) | logarithmic | excellent |
| O(n) | linear | good |
| O(n log n) | n log n | good |
| $O(n^2)$ | quadratic | OK |
| $O(n^3)$ | cubic | maybe OK |
| $O(2^n)$ | exponential | too slow |

11

## Related Notations

- Big-Omega

Definition: f(n) is a member of the set $\Omega(g(n))$ if and only if there exists constants C and k such that
$C\,|g(n)| < |f(n)|$, for all n>k

Read f(n) is $\Omega(g(n))$ as
f(n) is big-Omega of g(n)

- Big-Theta

Definition: f(n) is a member of the set $\Theta(g(n))$ if and only if f(n) is O(g(n)) and
f(n) is $\Omega(g(n))$

Read f(n) is $\Theta(g(n))$ as
f(n) is big-Theta of g(n) or
f(n) is of order g(n)

12

## Worst-Case/Average-Case Bounds

- We can't determine time bounds for all possible inputs of size *n*
- Simplifying assumption #4: Determine number of steps for either
  - worst-case or
  - average-case
- Worst-case
  - Determine how much time is needed for the *worst possible* input of size n
- Average-case
  - Determine how much time is needed *on average* for all inputs of size n

13

## Our Simplifying Assumptions

1. Use the *size* of the input rather than the input itself
2. Count the number of "*basic steps*" rather than computing exact times
3. Multiplicative constants aren't important (so we use big-O notation)
4. Determine number of steps for either
   - worst-case or
   - average-case

14

## Worst-Case Analysis of Searching

- Linear Search (check each element)

```
static boolean find (int[ ] a, int item) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == item) return true;
        }
    return false;
    }
```

For Linear Search, worst-case time is O(n)

For Binary Search, worst-case time is O(log n)

- Binary Search

```
static boolean find (int[ ] a, int item) {
    int low = 0;
    int high = a.length - 1;
    while (low <=  high) {
        int mid = (low+high)/2;
        if (a[mid] < item)
            low = mid+1;
        else if (item < a[mid])
            high = mid - 1;
        else return true;
        }
    return false;
    }
```

15

## Analysis of Matrix Multiplication

Code for multiplying n-by-n matrices A and B:

```
for (i = 0; i<n; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

- Worst-case time is $O(n^3)$
- Average-case time is also $O(n^3)$
- By convention, matrix problems are measured in terms of *n*, the number of rows and columns
  - Note that the input size is $2n^2$
  - If we let m be the input size then the time bound is actually $O(m^{3/2})$

16