Asymptotic Analysis

CSE 373
Data Structures & Algorithms
Ruth Anderson
Winter 2009

Today's Outline

- **Announcements**
  - Assignment #1 due Thurs, Jan 15 at 11:45pm
- **Math Review**
  - **Exponents and Logs**
- **Asymptotic Analysis**

Comparing Two Algorithms

What we want

- Rough Estimate
- Ignores Details

Big-O Analysis

- Ignores "details"

Analysis of Algorithms

- Efficiency measure
  - how long the program runs — time complexity
  - how much memory it uses — space complexity
    - For today, we'll focus on time complexity only
- *Why analyze at all?*

Asymptotic Analysis

- Complexity as a function of input size n
  - $T(n) = 4n + 5$
  - $T(n) = 0.5\, n \log n - 2n + 7$
  - $T(n) = 2^n + n^3 + 3n$
- *What happens as n grows?*

Why Asymptotic Analysis?

- Most algorithms are fast for small n
  - Time difference too small to be noticeable
  - External things dominate (OS, disk I/O, …)
- BUT n is often large in practice
  - Databases, internet, graphics, …
- Time difference really shows up as n grows!

Big-O: Common Names

- constant: O(1)
- logarithmic: O(log n)
- linear: O(n)
- quadratic: $O(n^2)$
- cubic: $O(n^3)$
- polynomial: $O(n^k)$ (k is a constant)
- exponential: $O(c^n)$ (c is a constant > 1)

Exercise

2	3	5	16	37	50	73	75	126

```
bool ArrayFind(int array[], int n, int key){
  // Insert your algorithm here

}
```

What algorithm would you choose to implement this code snippet?

Analyzing Code

Basic Java operations	Constant time
Consecutive statements	Sum of times
Conditionals	Larger branch plus test
Loops	Sum of iterations
Function calls	Cost of function body
Recursive functions	Solve recurrence relation

Analyze your code!

Linear Search Analysis

```
bool LinearArrayFind(int array[],
                     int n,
                     int key ) {
  for( int i = 0; i < n; i++ ) {
      if( array[i] == key )
           // Found it!
           return true;
  }
  return false;
}
```

Best Case:

Worst Case:

Binary Search Analysis

```
bool BinArrayFind( int array[], int low,
                   int high, int key ) {
   // The subarray is empty
   if( low > high ) return false;

   // Search this subarray recursively
   int mid = (high + low) / 2;
   if( key == array[mid] ) {
       return true;
   } else if( key < array[mid] ) {
       return BinArrayFind( array, low,
                            mid-1, key );
   } else {
       return BinArrayFind( array, mid+1,
                            high, key );
}
```

Best case:

Worst case:

Solving Recurrence Relations

1. Determine the recurrence relation. What is the base case(s)?
2. "Expand" the original relation to find an equivalent general expression *in terms of the number of expansions*.
3. Find a closed-form expression by setting *the number of expansions* to a value which reduces the problem to a base case