

## CSE 326: Data Structures

### Asymptotic Analysis

Larry Snyder  
Autumn 2006

## Today's Outline

- Admin: Office hours, etc.
- **Asymptotic analysis**

2

## Office Hours, etc.

Larry Snyder	Wed 4:30-5:20,	CSE 584
Paul Pham	Thur 2:30-3:30,	CSE 002
Brian Ngo	Tues 2:30-3:30,	CSE 002

**Or by appointment.**

TODO : *Important!*

1. Subscribe to mailing lists if you haven't
2. Get started on the Project 1

3

## Project 1 – Sound Blaster!

### **Play your favorite song in reverse!**

Aim:

1. Implement stack ADT two different ways
2. Use to reverse a sound file

Due: Wed October 11,

Electronic: before lecture  
Hardcopy: in lecture

4

## 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?*

5

## 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?*

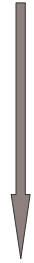
6

## 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!

7

## 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)

8

## 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

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

*Analyze your code!*

10

## 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:

11

## 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:

12

## 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

13

## Linear Search vs Binary Search

	Linear Search	Binary Search
Best Case		
Worst Case		

*So ... which algorithm is better?  
What tradeoffs can you make?*

14