

# Computational Complexity

CIS\*2520 Summer 2006

## Overview

- To this point we have focused on programming techniques
- What is the best or most appropriate algorithm (or method for representing data) for solving a given problem
  - searching for example
  - contiguous vs. linked
  - sequential vs. binary
- Note: if we have perfect information, and are implementing everything from scratch, we have completely flexibility to pick the ideal
  - in many cases our decisions are restricted by pre-existing constraints

CIS\*2520 - Summer 2006

D. McCaughan

## Program/Algorithm Efficiency

- How many resources are used by the program/algorithm?
  - TIME efficiency
    - how much time does it require
  - SPACE efficiency
    - how much memory/storage does it require
- A critical consideration in design
- NOTE: we are speaking of specific computational problems here, not interactive programs
- We will consider searching as an initial case study
  - we'll expand on these concepts as the course continues

CIS\*2520 - Summer 2006

D. McCaughan

## Program/Algo. Efficiency (cont.)

- For any computational problem there are many possible algorithms
  - which should we choose?
  - can we accurately predict how long an algorithm will take on an input?
  - can we bound how long it will take?
  - can we say that one algorithm will always be better than another?
- We wish to consider how time efficiency of an algorithm changes as:
  1. problem size grows
  2. we vary some property of the input for a fixed problem size
- Types of analysis: *empirical* or *analytical*

CIS\*2520 - Summer 2006

D. McCaughan

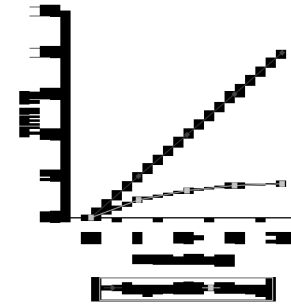
## Empirical Analysis

- Example: searching over an array of integers
- Problem size = number of integers (call this  $n$ )
  1. effect on efficiency as  $n$  increases
  2. for list of size  $n$ , vary the pre-existing order of integers in the list (i.e. sorted vs. random)
- Run program multiple times for different values of  $n$  recording execution times
  - graph results to see effect of problem size on time efficiency

CIS\*2520 - Summer 2006

D. McCaughan

## Empirical Analysis (cont.)



- Note: ratio between sequential and binary search gets progressively worse
  - they must differ in some fundamental way
  - sequential:
    - linearly proportional to size of input
    - growth rate is  $n$  for sequential search
  - binary
    - $\lg(16) = 4$
    - $\text{ceil}(\lg(100)) = 7$
    - growth rate is  $\lg(n)$  for binary search

CIS\*2520 - Summer 2006

D. McCaughan

## Analytical Analysis

- Empirical approach is fine, however it is inconvenient (infeasible in some cases) to implement and run experiments
  - we can do this analytically
- Goal in analyzing the efficiency of an algorithm
  - a function  $f(n)$  that expresses growth rate, where  $n$  is the problem size
  - want a “course” measure that captures the essence of algorithm efficiency
  - ideally want it to be independent of
    - machine used
    - language used
    - implementation details

CIS\*2520 - Summer 2006

D. McCaughan

## Analytical Analysis (cont.)

- Typically we count some key, elementary operation
  - comparisons
  - swaps
  - arithmetic operations
  - function calls
  - etc.
- In practice: concentrate only on the largest term and ignore constants, e.g.
  - $f(n) = 0.01n^2 + 999n + 3497$
  - we want to say what happens when  $n$  grows
  - in this case, the  $n^2$  term quickly dominates the other terms (we want to be independent of machine etc. so constants/coefficients are dropped)

CIS\*2520 - Summer 2006

D. McCaughan

## Big-Oh Notation

- Notation:
  - Let  $T(n)$  be the time an algorithm needs to solve a problem of size  $n$
- Definition:
  - $T(n)$  is in  $O(f(n))$  iff there is some constant  $c$  such that  $T(n) \leq c \cdot f(n)$  for all (sufficiently large)  $n$
- Note: if algorithm is in  $O(n^2)$ , it is also in  $O(n^3)$ , but we are interested in the “tightest bound”

CIS\*2520 - Summer 2006

D. McCaughan

## Example: Sequential Search

```
int location = 0;
while((lessThan(getKey(l->items[location]), target)
      && (location < l->item_count))
      location++);
if (isEqual(target, getKey(l->items[location])))
    return(getData(l->items[location]));
else
    return(null);
```

- For  $n = \text{item\_count}$ , number of comparisons (counting `lessThan` as a single comparison) in the worst case:
  - $2 \times n + 1 = 2n + 1 \rightarrow O(n)$

CIS\*2520 - Summer 2006

D. McCaughan

## Common Complexity Classes

- We encounter the following repeatedly in algorithm analysis
  - consider what happens in each as we double the problem size

$O(1)$	constant time (algorithm independent of problem size)
$O(\log_2 n)$	logarithmic time
$O(n)$	linear time
$O(n \log_2 n)$	“ $n \lg n$ ” time
$O(n^2)$	quadratic time
$O(n^3)$	cubic time
$O(c^n)$	exponential time (for some constant $c$ )

CIS\*2520 - Summer 2006

D. McCaughan

## Order of Complexity in Perspective

- Let us assume that we have some algorithm with a key operation that executes once per  $n$ 
  - Assume operation takes 1 microsecond -  $1/1000000$  sec.
- Focus of asymptotic complexity is the effect as  $n$  grows rather than the performance for any given  $n$ 
  - consider exponential time - for low  $n$  it can seem reasonable, but it grows *very* quickly
  - note that even if we assume the operation above took 1 nanosecond, the qualitative results are largely the same

CIS\*2520 - Summer 2006

D. McCaughan

## Complexity in Perspective

$n$	$O(\lg(n))$	$O(n)$	$O(n \lg(n))$	$O(n^2)$	$O(n^3)$	$O(2^n)$
10	3.3 ms	10 ms	33 ms	100 ms	1 ms	1 ms
20	4.3 ms	20 ms	86 ms	400 ms	8 ms	1 sec
30	4.9 ms	30 ms	147 ms	900 ms	27 ms	18 min
40	5.3 ms	40 ms	213 ms	1.6 ms	60 ms	13 days
50	5.6 ms	50 ms	282 ms	2.5 ms	125 ms	36 years
60	5.9 ms	60 ms	354 ms	3.6 ms	216 ms	since Cro-Magnon Man
70	6.1 ms	70 ms	429 ms	4.9 ms	343 ms	since last dinosaur
80	6.3 ms	80 ms	506 ms	6.4 ms	512 ms	two big bangs ago
90	6.5 ms	90 ms	584 ms	8.1 ms	729 ms	39 billion millenia
100	6.7 ms	100 ms	664 ms	10.0 ms	1 sec	402 trillion centuries
1 000	10.0 ms	1 ms	10 ms	1.0 sec	16.7 min	$3 \times 10^{298}$ years
10 000	13.0 ms	10 ms	132 ms	1.7 min	11.6 days	beginning of time
100 000	17.0 ms	100 ms	1.7 sec	2.8 hr	31.7 years	pre-time void

CIS\*2520 - Summer 2006

D. McCaughan

## Basic Rules for Big-Oh Notation

- Addition rule:
  - $O(f(n) + g(n)) = O(\max(f(n), g(n)))$
  - e.g.  $O(n^2 + n \lg(n)) = O(n^2)$
  - implication: complexity determined by complexity of slowest fragment
- Multiplication rule:
  - $O((f * g)(n)) = O(f(n) * g(n))$
  - e.g.  $O(n^2 * \lg(n)) = O(n^2 \lg(n))$
- Simple statements (assignment, comparison, etc.)
  - $O(1)$
  - not necessarily for arrays or involving function calls

CIS\*2520 - Summer 2006

D. McCaughan

## Basic Rules for Big-Oh Notation

- Sub-routines
  - time to set-up parameters + time to execute body
- if/then/else
  - time for test + time for worst case statement
- Iterative statements
  - sum time for body over all iterations
  - typically: # iterations \* (time for body)
- Sequences of statements
  - use addition rule

CIS\*2520 - Summer 2006

D. McCaughan

## On Counting Representative Operations

- We define the key statement(s) for complexity determination
  - it must be appropriate
  - we can usually justify ignoring what remains as the operations of interest are somehow the "essential" ones for the algorithm in question
  - e.g. searching
    - comparisons for a search are the essential operation
    - we can ignore assignments and mathematical operations as they will be a multiple of the comparisons anyway

CIS\*2520 - Summer 2006

D. McCaughan

## Best/Average/Worst Cases

- There are three major results we might be interested in:
  - BEST case
    - typically not useful
  - AVERAGE case
    - can be difficult to calculate this
    - what is average?
  - WORST case
    - gives an upper bound on time

CIS\*2520 - Summer 2006

D. McCaughan

## Analysis of Sequential Search

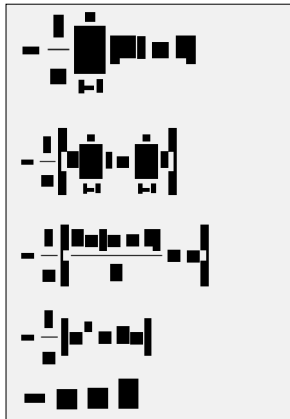
- Count comparisons; let size of list be  $n$
- **Best case:**
  - $2 + 1$  comparisons
  - $O(1)$
- **Worst Case:**
  - $2n + 1$  comparisons (i.e.  $O(1)$  loop body,  $n$  iterations)
  - $O(n)$
- **Average Case?**

CIS\*2520 - Summer 2006

D. McCaughan

## Avg. Case for Sequential Search

- Assume equal probability distribution for finding a key in a given location
  - *weighted average:*
    - sum of (probability of key at position  $i$ ) \* (operations if found at position  $i$ ), for all  $i$
- $f(n) = n + 2$
- $O(n)$
- Note: average == worst



CIS\*2520 - Summer 2006

D. McCaughan

## Analysis of Binary Search

- Note: BEST = AVERAGE = WORST (nature of algorithm)
- Count comparisons; let size of list be  $n$
- Background: Comparison Trees
  - also called "decision tree" or "search tree"
  - obtained, for an algorithm, by tracing the actions of the algorithm:
    - each comparison of keys is a **VERTEX** of the tree (circle), inside of which we put the index of the key we are comparing to the target
    - **BRANCHES** (lines) drawn down from the circle represent possible outcomes of the comparison and are labeled accordingly
    - when the algorithm terminates, we put either F (failure) or the location where the target is found at the end of the appropriate branch: **LEAF** (square)

CIS\*2520 - Summer 2006

D. McCaughan

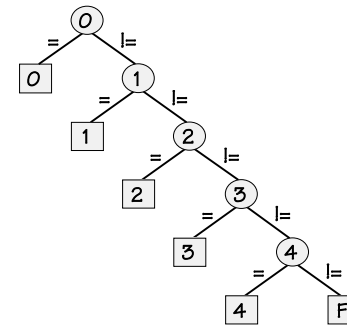
## Comparison Trees

- Terminology
  - external vertices** of a tree
    - leaves (end vertices)
  - internal vertices** of a tree
    - non-leaves (i.e. what's left)
  - level** of a vertex
    - # branches traversed to reach it (0, 1, ...)
  - height** of a tree
    - largest level that occurs
  - vertices immediately below vertex,  $v$ , are the **children** of  $v$
  - the vertex immediately above vertex,  $v$ , is the **parent** of  $v$
- Number of comparisons done in a search:
  - # of internal vertices traversed going from the ROOT down the appropriate path to a leaf

CIS\*2520 - Summer 2006

D. McCaughan

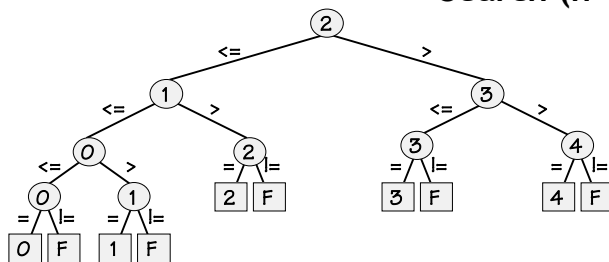
## Comparison Tree for Sequential Search (n=5)



CIS\*2520 - Summer 2006

D. McCaughan

## Comparison Tree for Binary Search (n=5)



- Note: we can make this even more compact
  - "fake" a 3-way comparison by having internal vertices terminate searches on equality (all leaves would be unsuccessful searches)

CIS\*2520 - Summer 2006

D. McCaughan

## 2-Trees

- Technically we are considering "2-trees" here
  - every vertex except the leaves have 2 children
  - also called "strictly binary tree"
- Some results on 2-Trees:
  - number of vertices on each level of a 2-Tree is at most twice the number on the level immediately above it
  - the number of vertices on level  $t$  is at most  $2^t$  for  $t \geq 0$

CIS\*2520 - Summer 2006

D. McCaughan

## Binary Search Analysis (cont.)

- Now: for binary search, both successful and unsuccessful searches terminate at leaves
  - there are thus  $2n$  leaves
- All leaves are on the same level or two adjacent levels
  - trivial proof by induction:
    - assume the claim is true (induction hypothesis)
    - obviously true for lists of size  $n=1$
    - when `binarySearch` divides a list, the sizes of the two halves differ by at most 1
    - by induction hypothesis these leaves are on same or adjacent levels .

CIS\*2520 - Summer 2006

D. McCaughan

## Binary Search Analysis (cont.)

- The height (# levels below the root) of the tree is the maximum number of key comparisons that the algorithm does
  - in this case, at most 1 more than the average number (as all leaves are on the same or different levels)
- From the above:
  - the height is the smallest integer,  $t$ , s.t.  $2^t \geq 2n$
  - take  $\log_2$  of both sides:  $t$  is approximately  $\log_2 n + 1$
- $O(\log_2 n)$

CIS\*2520 - Summer 2006

D. McCaughan

## Some Claims

- A similar approach using binary search with comparisons for equality at each iteration shows the added work to test for equality is only of benefit for very small lists
  - hurts as list size increases, although both are still  $O(\log_2 n)$
- Any search based on comparisons can do no better than  $O(\log_2 n)$ 
  - so can we break this “ $\log_2 n$ ” barrier?
- We can use comparison trees for higher order comparisons/searches
  - approach is similar and results are developed in the same way

CIS\*2520 - Summer 2006

D. McCaughan