ALGORITHMS & ADVANCED DATA STRUCTURES (#1)

INTRODUCTION

ADAPTED FROM
CS 146 SJSU (KATERINA POTIKA)

Basic Data Structures

Data Structure	Search	Insertion	Deletion	Traversal
Array	O(N)	O(1)	O(N)	—
Ordered array	O(logN)	O(N)	O(N)	O(N)
Linked list	O(N)	O(1)	O(N)	—
Ordered linked list	O(N)	O(N)	O(N)	O(N)
Binary tree (average)	O(logN)	O(logN)	O(logN)	O(N)
Binary tree (worst case)	O(N)	O(N)	O(N)	O(N)
Balanced tree (average and worst case)	O(logN)	O(logN)	O(logN)	O(N)
Hash table	O(1)	O(1)	O(1)	—

Efficiency

- Same problem can have different algorithms that solve them
- Example:

  Sorting problem (sort n numbers)

Algorithm for sorting	Worst time
Insertion Sort	$c_1 n^2$
Merge Sort	$c_2 n \log n$

Difference in running time

- Sort 10 million numbers use:

1. Insertion Sort: with 10 billion instructions/sec machine and running time (instructions) $2n^2$

$$\frac{2(10^7)^2\ instructions}{10^{10}\ instructions/sec} = 20.000 \text{ sec } (5.5 \text{ h})$$

2. Merge Sort: with 10 million instructions/sec and running time $50n\log n$

$$\frac{50*10^7 log 10^7 \quad instructions}{10^7\ instructions/sec} = 1163 \text{ sec } (20 \text{ min})$$

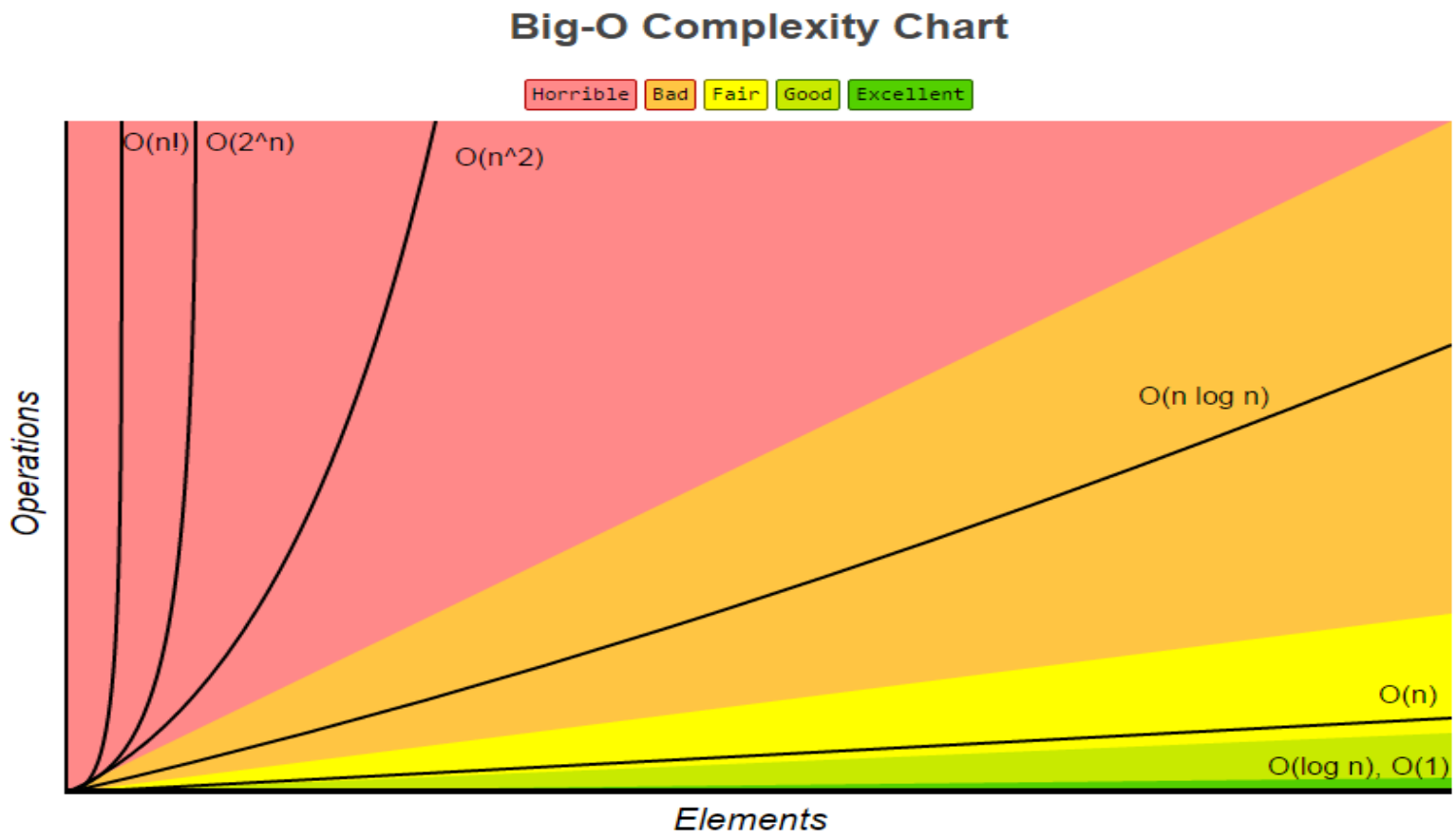
Asymptotic Performance

- In this course, we care most about *asymptotic performance*
  - How does the algorithm behave as the problem size gets very large?
    - Running time
    - Memory/storage requirements
    - Bandwidth/power requirements/logic gates/etc.

Asymptotic Notation

- Intuitive feel for asymptotic (big-O) notation:
  - *What does O(n) running time mean? O(n^2)? O(n lg n)?*
  - *How does asymptotic running time relate to asymptotic memory usage?*
- We will define this notation more formally and completely

Practical Complexity



bigocheatsheet.com

Analysis of Algorithms

- Analysis is performed with respect to a computational model
- We will usually use a generic uniprocessor random-access machine (RAM) – all operations cost const
  - All memory equally expensive to access
  - No concurrent operations
  - All reasonable instructions take unit time
    - Except, of course, function/method calls
  - Constant word size
    - Unless we are explicitly manipulating bits

Input Size

- Time and space complexity
  - This is generally a function of the **input size**
    - E.g., sorting, multiplication
  - How we characterize input size depends:
    - Sorting: number of input items
    - Multiplication: total number of bits
    - Graph algorithms: number of nodes & edges
    - Etc

Running Time

- Number of primitive steps that are executed
  - Except for time of executing a function/method call most statements roughly require the same amount of time
    - y = m * x + b
    - c = 5 / 9 * (t - 32)
    - z = f(x) + g(y)

Analysis

- Worst case
  - Provides an upper bound on running time
  - An absolute guarantee
- Average case
  - Provides the expected running time
  - Very useful, but treat with care: what is “average”?
    - Random (equally likely) inputs
    - Real-life inputs

Worst vs Average case Analysis

Worst case running time analysis

1. Gives an upper bound on the running time for any input (guarantee)
2. For some algorithms, worst case occurs fairly often
3. Average case as bad as worst case

Review: Induction

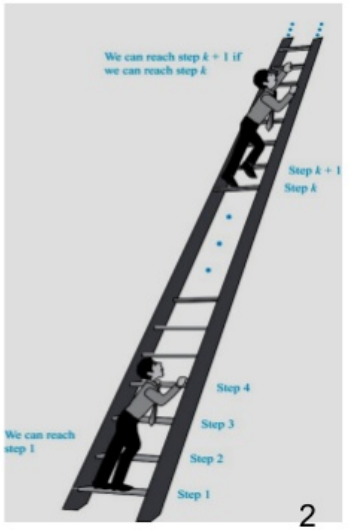
- Suppose
  - S(k) is true for fixed constant k
    - Often k = 0
  - S(n) ➔ S(n+1) for all n >= k
- Then S(n) is true for all n >= k

Climbing an Infinite Ladder

Suppose we have an infinite ladder:
1. We can reach the first rung of the ladder.
2. If we can reach a particular rung of the ladder, then we can reach the next rung.

From (1), we can reach the first rung. Then by applying (2), we can reach the second rung. Applying (2) again, the third rung. And so on. We can apply (2) any number of times to reach any particular rung, no matter how high up.

This example motivates proof by mathematical induction.



2

Proof By Induction

- Claim:S(n) is true for all n >= k
- Basis:
  - Show formula is true when n = k
- Inductive hypothesis:
  - Assume formula is true for an arbitrary n
- Step:
  - Show that formula is then true for n+1

Induction Example: Gaussian Closed Form

- Prove $1 + 2 + 3 + \ldots + n = n(n+1) / 2$
  - Basis:
    - If $n = 0$, then $0 = 0(0+1) / 2$
  - Inductive hypothesis:
    - Assume $1 + 2 + 3 + \ldots + n = n(n+1) / 2$
  - Step (show true for n+1):

$1 + 2 + \ldots + n + n+1 = (1 + 2 + \ldots + n) + (n+1)$

$= n(n+1)/2 + n+1 = [n(n+1) + 2(n+1)]/2$

$= (n+1)(n+2)/2 = (n+1)(n+1 + 1) / 2$

Induction Example: Geometric Closed Form

- Prove $a^0 + a^1 + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$ for all $a \neq 1$
  - Basis: show that $a^0 = (a^{0+1} - 1)/(a - 1)$

    $a^0 = 1 = (a^1 - 1)/(a - 1)$
  - Inductive hypothesis:
    - Assume $a^0 + a^1 + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$
  - Step (show true for n+1):

    $a^0 + a^1 + \ldots + a^{n+1} = a^0 + a^1 + \ldots + a^n + a^{n+1}$

    $= (a^{n+1} - 1)/(a - 1) + a^{n+1} = (a^{n+1+1} - 1)/(a - 1)$