DESIGN AND ANALYSIS OF ALGORITHMS

:

Dr Muhammad Aasim Qureshi

1

OVERVIEW

- Administrative stuff…
  - Announcements
- What is an algorithm?
- Systems for studying algorithms
- Lecture schedule overview

2

COURSE INFO

- Textbook:
  - Intro. to Algorithms, by Cormen, Leiserson, Rivest (McGraw Hill) 4th Ed. (or latest one)
    - Buy your own book before next class (Reading is compulsory)
- Reference
  - Computers and Intractability by Garey and Johnson
- Office Hours: will be announced on my office wall
- WhatsApp Group

3

GRADING

- 30%
  - Homework
  - Quizes
  - Assignments
  - Project/Research
- 30% Midterms
- 40% Final Term
- Class Participation

- Queries regarding Quizzes or Assignments will the entertained on the exact day
- In case of permission you must have it in black and white
- Keep all your assignments and quizzes safe for any ambiguity

4

WHAT IS THE COURSE ABOUT?

- In this course, we will study
  - Common *computational problems* and *algorithms*
  - Common strategies for designing algorithms
  - How to implement algorithms? => Using data structures
  - Advanced *data structures*
  - How to evaluate good algorithms and data structures => Analysis Techniques
  - Applying theory to problem solving

5

WHAT THE COURSE COVERS

- Algorithms in pseudo code
- Analysis tools
- Important data structures

What the course does not cover
- Implementation details using a particular language
- Debugging

- What you Need to Revise
  - Asymptotic Notations
  - Sorting Algorithms-Insertion, Selection, Bubble, Quick, Merge
  - Chapters 1, 2, 3, 4, 7, 22, 23, 24

Included in your Papers

Quiz Next Week

6

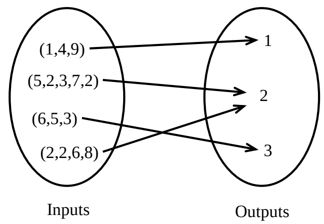
PROBLEM

- Why do we write programs?
  - to perform some specific tasks
  - to solve some specific problems
  - We will focus on "solving problems"
- What is a "problem"?
  - We can view a problem as a mapping of "inputs" to "outputs"

7

A PROBLEM IS A MAPPING

- Example: Find Minimum



8

DESCRIBING A PROBLEM

- How to describe a problem?
  - Input
    - Describe how an input looks like.
  - Output
    - Describe how an output looks like and how it relates to the input.

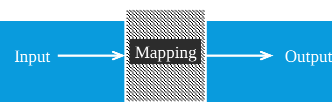
9

AN "INSTANCE" OF A PROBLEM

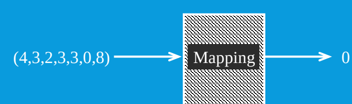
- An *instance* is an assignment of values to the input variables.
- Example:
  - An instance of the Find Minimum Problem
    - $N = 10$
    - $(a_1, a_2, \ldots, a_N) = (5,1,7,4,3,2,3,3,0,8)$

10

A PROBLEM IS A BLACK BOX

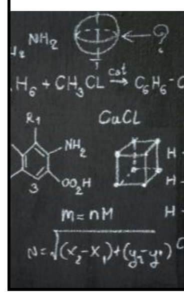


- Example: Find Minimum



11

EXAMPLE: SORTING



- **Input**: A sequence of N numbers $a_1 \ldots a_n$
- **Output**: the permutation (reordering) of the input sequence such that $a_1 \le a_2 \le \ldots \le a_n$.

12

HOW DO WE SOLVE A PROBLEM?

- Write a program that "implements" the mapping.
  - takes an *input* in and produces a correct *output*

Input → Program or *Algorithms* → Output

13

EXPRESSING ALGORITHMS

Three ways of expressing algorithms:

- Implementations
- Pseudo-code
- English

14

WHAT IS AN ALGORITHM?

- According to the *Academic American Encyclopedia:*
  - An algorithm is a procedure for solving a usually complicated problem by carrying out a precisely determined sequence of simpler, unambiguous steps. Such procedures were originally used in mathematical calculations (the name is a variant of algorism, which originally meant the Arabic numerals and then "arithmetic") but are now widely used in computer programs and in programmed learning.

15

ALGORITHMS

- We can write similar programs in other programming languages.
- We will not focus on a particular language.
- We will look at the algorithms behind the programs.
- Informally, an *algorithm* is a set of rules which gives a sequence of operations for solving a problem.

16

Data Structures

- What is a "data structure"?
  - a systematic way of organizing and manipulating data
- Two aspects
  - Data organization
  - Functions for manipulating the data

17

HOW TO JUDGE GOOD ALGORITHMS

- Efficiency
  - Time Complexity … How fast the program is?
  - Space Complexity
- Simplicity …
  - Program Complexity
    - Shorter better?: how about C++ programs??
  - Easy to maintain
- Correctness
  - formal method: related to program verification.
  - informal method: we usually rely on this method

18

CORRECTNESS

- For any algorithm, we must prove that it *always* returns the desired output for all legal instances of the problem.
- How about almost always correct?
- Example: Given an election of 100,000,000 votes, determine the winner.
- *Approach 1*: Count all 100,000,000 votes!
- *Approach 2*: Select a sample of 1000,000 votes and declare the winner of the sample as the winner of the election.
- ==> with high probability, approach 2 will determine the winner correctly! But not always.

19

WHAT IS "EFFICIENCY"?

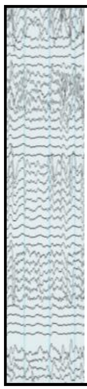
IN WHAT WAYS CAN WE COMPARE ALGORITHMS?

20

HOW DO WE MEASURE *SPEED*?

- Compare by processor?
- By compiler?
- What about optimization level?
- "Why not use a super-computer?"

21



THE RAM MODEL

- RAM model represents a "generic" implementation of t algorithm
- Each "simple" operation (+, -, =, if, call) takes exactly 1
- Loops and subroutine calls are not simple operations, b depend upon the size of the data and the contents of a subroutine. We do not want "sort" to be a single step operation.
- Each memory access takes exactly 1 step.

Other Models

- logn model: An n bit operation takes O(logn)

22

THE PROBLEM OF SORTING

Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \le a'_2 \le \cdots \le a'_n$.

Example:

Input: 8 2 4 9 3 6

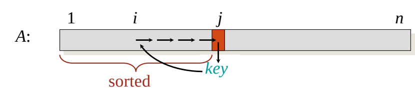
Output: 2 3 4 6 8 9

23

INSERTION SORT

"pseudocode"

```
INSERTION-SORT (A, n)      ⊳ A[1 . . n]
   for j ← 2 to n
        do key ← A[ j]
           i ← j – 1
           while i > 0 and A[i] > key
                do A[i+1] ← A[i]
                   i ← i – 1
           A[i+1] = key
```



24

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6

25

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6

26

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6

2 8 4 9 3 6

27

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6

2 8 4 9 3 6

28

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

29

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

30

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6

31

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6

32

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6

33

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6

34

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6
2 3 4 6 8 9 *done*

35

RUNNING TIME

- The running time depends on the input: an already sorted sequence is easier to sort.
- Major Simplifying Convention: Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
  - ➢ $T_A(n)$ = time of A on length n inputs
- Generally, we seek upper bounds on the running time, to have a guarantee of performance.

36

Types of Analysis

- Worst case
  - Provides an upper bound on running time
  - An absolute guarantee that the algorithm would not run longer, no matter what the inputs are
- Best case
  - Provides a lower bound on running time
  - Input is the one for which the algorithm runs the fastest

$$\boxed{Lower\ Bound \leq Running\ Time \leq Upper\ Bound}$$

- Average case
  - Provides a prediction about the running time
  - Assumes that the input is random

37

37

Worst-case: (usually) — $T(n)$ = maximum time of algorithm on any input of size n.

Average-case: (sometimes) — $T(n)$ = expected time of algorithm over all inputs of size n. Need assumption of statistical distribution of inputs.

Best-case: (NEVER) — Cheat with a slow algorithm that works fast on *some* input.

38

MACHINE-INDEPENDENT TIME

What is insertion sort's worst-case time?

BIG IDEAS:

- ***Ignore machine dependent constants,*** otherwise impossible to verify and to compare algorithms
- Look at ***growth*** of $T(n)$ as $n \to \infty$.

"Asymptotic Analysis"

39

HOW DO WE COMPARE ALGORITHMS?

- We need to define a number of objective measures.

(1) Compare execution times?

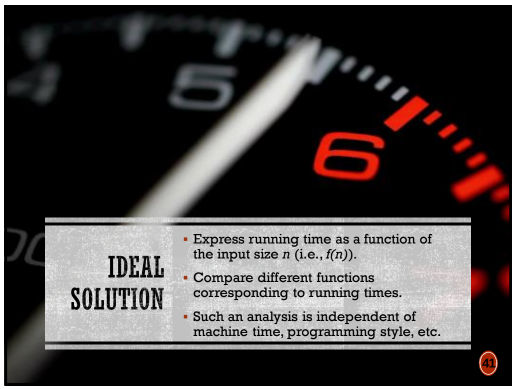
Not good: times are specific to a particular computer !!

(2) Count the number of statements executed?

Not good: number of statements vary with the programming language as well as the style of the individual programmer.

40

40



IDEAL SOLUTION

- Express running time as a function of the input size n (i.e., $f(n)$).
- Compare different functions corresponding to running times.
- Such an analysis is independent of machine time, programming style, etc.

41

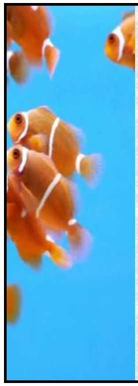
41

ASYMPTOTIC ANALYSIS

- To compare two algorithms with running times $f(n)$ and $g(n)$, we need a **rough measure** that characterizes **how fast each function grows.**
- *Hint:* use *rate of growth*
- Compare functions in the limit, that is, **asymptotically!** (i.e., for large values of n)

42

42



RATE OF GROWTH

- Consider the example of buying *elephants* and *goldfish:*
  - **Cost**: cost_of_elephants + cost_of_goldfish
  - **Cost** ~ cost_of_elephants (approximation)
- The low order terms in a function are relatively insignificant for **large** n

$$n^4 + 100n^2 + 10n + 50 \sim n^4$$

i.e., we say that $n^4 + 100n^2 + 10n + 50$ and n^4 have the same **rate of growth**

43

43

ASYMPTOTIC NOTATION

- O notation: asymptotic "less than":
  - f(n)=O(g(n)) implies: f(n) "≤" g(n)
- Ω notation: asymptotic "greater than":
  - f(n)= Ω (g(n)) implies: f(n) "≥" g(n)
- Θ notation: asymptotic "equality":
  - f(n)= Θ (g(n)) implies: f(n) "=" g(n)

44

44

BIG-O NOTATION

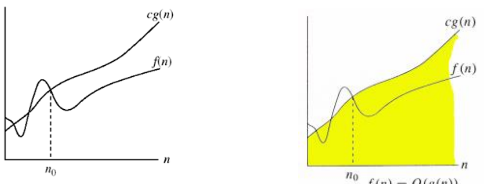
- We say $f_A(n)=30n+8$ is *order n*, or O (n)
  It is, at most, roughly *proportional* to *n*.
- $f_B(n)=n^2+1$ is *order* n^2, or $O(n^2)$. It is, at most, roughly proportional to n^2.
- In general, any $O(n^2)$ function is faster- growing than any $O(n)$ function.

45

45

BIG OH ("O")NOTATION

$O(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$.



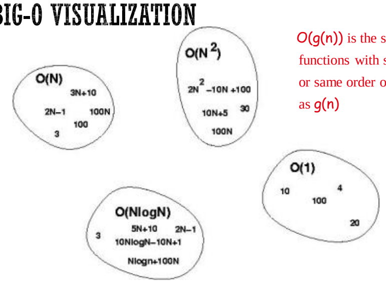
$f(n) = O(g(n))$

$g(n)$ is an ***asymptotic upper bound*** for $f(n)$.

46

46

BIG-O VISUALIZATION



O(g(n)) is the set of functions with smaller or same order of growth as g(n)

47

47

EXAMPLES

- $2n^2 = O(n^3)$: $2n^2 \le cn^3 \Rightarrow 2 \le cn \Rightarrow c = 1$ and $n_0 = 2$
- $n^2 = O(n^2)$: $n^2 \le cn^2 \Rightarrow c \ge 1 \Rightarrow c = 1$ and $n_0 = 1$
- $1000n^2+1000n = O(n^2)$:

$1000n^2+1000n \le 1000n^2+ n^2 = 1001n^2 \Rightarrow c=1001$ and $n_0 = 1000$

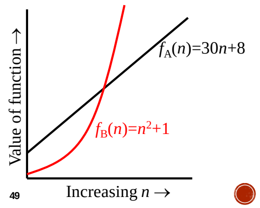
$n = O(n^2)$: $n \le cn^2 \Rightarrow cn \ge 1 \Rightarrow c = 1$ and $n_0 = 1$

48

48

VISUALIZING ORDERS OF GROWTH

- On a graph, as you go to the right, a faster growing function eventually becomes larger...



49

49

MORE EXAMPLES …

- $n^4 + 100n^2 + 10n + 50$ is $O(n^4)$
- $10n^3 + 2n^2$ is $O(n^3)$
- $n^3 - n^2$ is $O(n^3)$
- constants
  - 10 is $O(1)$
  - 1273 is $O(1)$

50

50

MORE EXAMPLES

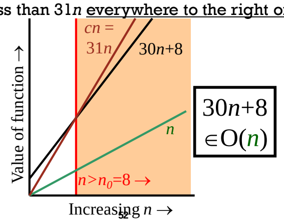
- Show that $30n+8$ is $O(n)$.
  - Show $\exists c,n_0: 30n+8 \leq cn, \forall n>n_0$.
    - Let $c=31$, $n_0=8$. Assume $n>n_0=8$. Then $cn = 31n = 30n + n > 30n+8$, so $30n+8 < cn$.

51

51

BIG-O EXAMPLE, GRAPHICALLY

- Note $30n+8$ isn't less than n *anywhere* ($n>0$).
- It isn't even less than $31n$ *everywhere*.
- But it *is* less than $31n$ everywhere to the right of $n=8$.



$\boxed{30n+8 \in O(n)}$

52

52

NO UNIQUENESS

- There is no unique set of values for n_0 and c in proving the asymptotic bounds
- Prove that $100n + 5 = O(n^2)$
  - $100n + 5 \leq 100n + n = 101n \leq 101n^2$

    for all $n \geq 5$

    $n_0 = 5$ and $c = 101$ is a solution
  - $100n + 5 \leq 100n + 5n = 105n \leq 105n^2$

    for all $n \geq 1$

    $n_0 = 1$ and $c = 105$ is also a solution

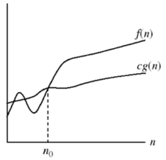
Must find **SOME** constants c and n_0 that satisfy the asymptotic notation relation

53

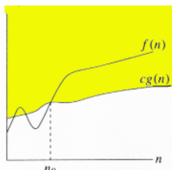
53

BIG OMEGA ("Ω")

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}.$$



$g(n)$ is an ***asymptotic lower bound*** for $f(n)$.



$f(n) = \Omega(g(n))$

$\Omega(g(n))$ is the set of functions with larger or same order of growth as $g(n)$

54

54

$\Rightarrow$ contradiction: n cannot be smaller than a consta

EXAMPLES

- $5n^2 = \Omega(n)$

  $\exists\ c, n_0$ such that: $0 \le cn \le 5n^2$ $\Rightarrow$ $cn \le 5n^2 \Rightarrow c = 1$ and $n_0 = 1$
- $100n + 5 \ne \Omega(n^2)$

  $\exists$ c, n_0 such that: $0 \le cn^2 \le 100n + 5$

  $100n + 5 \le 100n + 5n$ ($\forall$ $n \ge 1$) $= 105n$ $\quad cn^2 \le 105n$

  $\Rightarrow n(cn - 105) \le 0$

  Since n is positive $\Rightarrow cn - 105 \le 0$ $\quad \Rightarrow n \le 105/c$

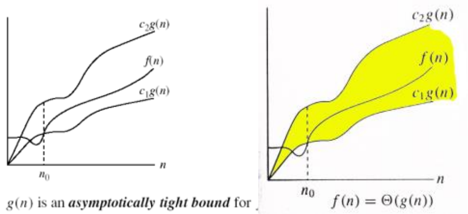
- $n = \Omega(2n)$, $n^3 = \Omega(n^2)$, $n = \Omega(\log n)$

55

55

ASYMPTOTIC NOTATIONS (CONT.)

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$.



$g(n)$ is an ***asymptotically tight bound*** for

$\Theta(g(n))$ is the set of functions with the same order of growth as $g(n)$

56

56

EXAMPLES

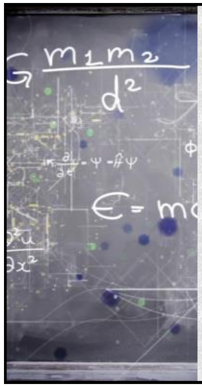
- $n^2/2 - n/2 = \Theta(n^2)$
  - $\frac{1}{2}n^2 - \frac{1}{2}n \le \frac{1}{2}n^2\ \forall n \ge 0 \quad \Rightarrow \quad c_2 = \frac{1}{2}$
  - $\frac{1}{2}n^2 - \frac{1}{2}n \ge \frac{1}{2}n^2 - \frac{1}{2}n * \frac{1}{2}n\ (\forall n \ge 2) = \frac{1}{4}n^2 \quad \Rightarrow \quad c_1 = \frac{1}{4}$

- $n \ne \Theta(n^2)$: $c_1 n^2 \le n \le c_2 n^2$

  $\Rightarrow$ only holds for: $n \le 1/c_1$

57

57



EXAMPLES

- $6n^3 \ne \Theta(n^2)$: $c_1 n^2 \le 6n^3 \le c_2 n^2$

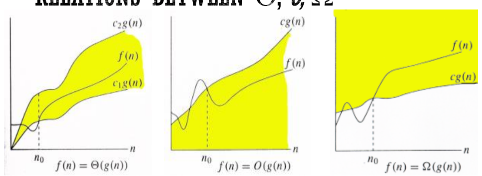
  $\Rightarrow$ only holds for: $n \le c_2/6$
- $n \ne \Theta(\log n)$: $c_1 \log n \le n \le c_2 \log n$

  $\Rightarrow c_2 \ge n/\log n$, $\forall$ $n \ge n_0$ – impossible

58

58

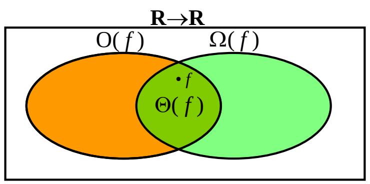
RELATIONS BETWEEN Θ, O, Ω



59

RELATIONS BETWEEN DIFFERENT SETS

- Subset relations between order-of-growth sets.



60

60

What does all this mean?

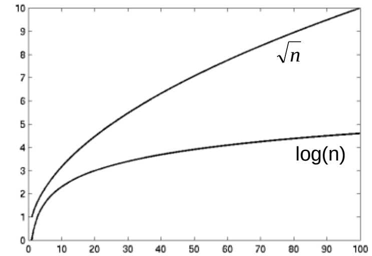
- $3n^2 - 100n + 6 = O(n^2)$ because $3n^2 > 3n^2 - 100n + 6$
- $3n^2 - 100n + 6 = O(n^3)$ because $0.0001n^3 > 3n^2 - 100n + 6$
- $3n^2 - 100n + 6 \neq O(n)$ because $c \times n < 3n^2$ when $n > c$

- $3n^2 - 100n + 6 = \Omega(n^2)$ because $2.99n^2 < 3n^2 - 100n + 6$
- $3n^2 - 100n + 6 \neq \Omega(n^3)$ because $3n^2 - 100n + 6 < n^3$
- $3n^2 - 100n + 6 = \Omega(n)$ because $10^{10}!\, n < 3n^2 - 100n + 6$

- $3n^2 - 100n + 6 = \Theta(n^2)$ because both O and Ω
- $3n^2 - 100n + 6 \neq \Theta(n^3)$ because O only
- $3n^2 - 100n + 6 \neq \Theta(n)$ because Ω only

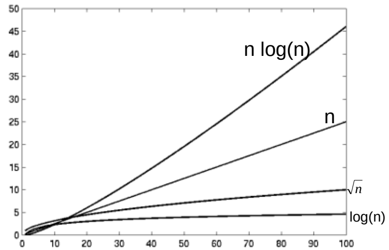
61

Complexity Graphs



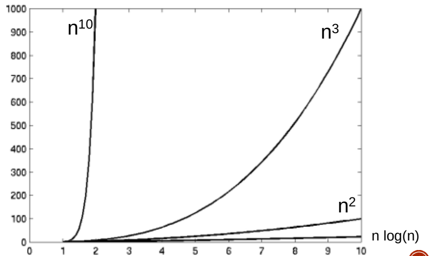
62

Complexity Graphs



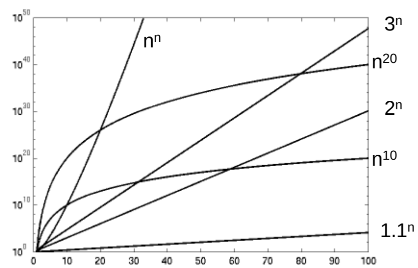
63

Complexity Graphs



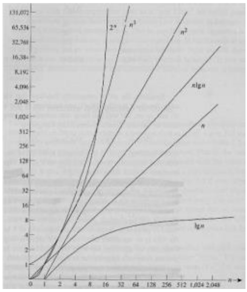
64

Complexity Graphs (log scale)



65

Common orders of magnitude



66

Common orders of magnitude

Table 1.4 Execution times for algorithms with the given time complexities

n	$f(n) = \lg n$	$f(n) = n$	$f(n) = n \lg n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$
10	0.003 μs*	0.01 μs	0.033 μs	0.1 μs	1 μs	1 μs
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	8 μs	1 ms†
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	27 μs	1 s
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	64 μs	18.3 min
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	125 μs	13 days
10^2	0.007 μs	0.10 μs	0.664 μs	10 μs	1 ms	4×10^{13} years
10^3	0.010 μs	1.00 μs	9.966 μs	1 ms	1 s	
10^4	0.013 μs	10 μs	130 μs	100 ms	16.7 min	
10^5	0.017 μs	0.10 ms	1.67 ms	10 s	11.6 days	
10^6	0.020 μs	1 ms	19.93 ms	16.7 min	31.7 years	
10^7	0.023 μs	0.01 s	0.23 s	1.16 days	31,709 years	
10^8	0.027 μs	0.10 s	2.66 s	115.7 days	3.17×10^7 years	
10^9	0.030 μs	1 s	29.90 s	31.7 years		

*1 μs = 10^{-6} second.
†1 ms = 10^{-3} second.

67

67

LOGARITHMS AND PROPERTIES

- In algorithm analysis we often use the notation "log n" without specifying the base

Binary logarithm $\lg n = \log_2 n$

Natural logarithm $\ln n = \log_e n$

$\lg^k n = (\lg n)^k$

$\lg \lg n = \lg(\lg n)$

$\log x^y = y \log x$

$\log xy = \log x + \log y$

$\log \frac{x}{y} = \log x - \log y$

$a^{\log_b x} = x^{\log_b a}$

$\log_b x = \frac{\log_a x}{\log_a b}$

68

68

MORE EXAMPLES

- For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct.
  - $f(n) = \log n^2$; $g(n) = \log n + 5$
  - $f(n) = n$; $g(n) = \log n^2$
  - $f(n) = \log \log n$; $g(n) = \log n$
  - $f(n) = n$; $g(n) = \log^2 n$
  - $f(n) = n \log n + n$; $g(n) = \log n$
  - $f(n) = 10$; $g(n) = \log 10$
  - $f(n) = 2^n$; $g(n) = 10n^2$
  - $f(n) = 2^n$; $g(n) = 3^n$

$f(n) = \Theta(g(n))$

$f(n) = \Omega(g(n))$

$f(n) = O(g(n))$

$f(n) = \Omega(g(n))$

$f(n) = \Omega(g(n))$

$f(n) = \Theta(g(n))$

$f(n) = \Omega(g(n))$

$f(n) = O(g(n))$

69

69

PROPERTIES

- *Theorem:*

  $f(n) = \Theta(g(n)) \Leftrightarrow f = O(g(n))$ and $f = \Omega(g(n))$
- Transitivity:
  - $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
  - Same for O and Ω
- Reflexivity:
  - $f(n) = \Theta(f(n))$
  - Same for O and Ω
- Symmetry:
  - $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$
- Transpose symmetry:
  - $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$

70

70

ASYMPTOTIC NOTATIONS IN EQUATIONS

- On the right-hand side
  - $\Theta(n^2)$ stands for some anonymous function in $\Theta(n^2)$

  $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ means:

  There exists a function $f(n) \in \Theta(n)$ such that

  $2n^2 + 3n + 1 = 2n^2 + f(n)$
- On the left-hand side

  $2n^2 + \Theta(n) = \Theta(n^2)$

  No matter how the anonymous function is chosen on the left-hand side, there is a way to choose the anonymous function on the right-hand side to make the equation valid.

71

71

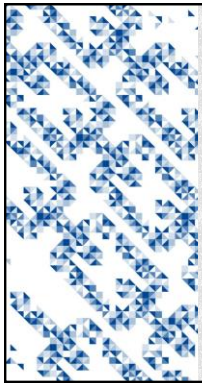
COMMON SUMMATIONS

- Arithmetic series: $\sum_{k=1}^{n} k = 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$
- Geometric series: $\sum_{k=0}^{n} x^k = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1} - 1}{x - 1} (x \neq 1)$
  - Special case: $|x| < 1$: $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$
- Harmonic series: $\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \ldots + \frac{1}{n} \approx \ln n$
- Other important formulas: $\sum_{k=1}^{n} \lg k \approx n \lg n$

  $\sum_{k=1}^{n} k^p = 1^p + 2^p + \ldots + n^p \approx \frac{1}{p+1} n^{p+1}$

72

72

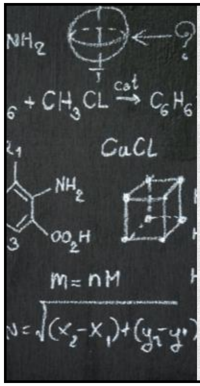


MATHEMATICAL INDUCTION

- A powerful, rigorous technique for proving that a statement S(*n*) is true for *every* natural number *n*, no matter how large.
- Proof:
  - **Basis step**: prove that the statement is true for n = 1
  - **Inductive step:** assume that *S*(n) is true and prove that *S*(n+1) is true for all n ≥ 1
- Find case n "within" case n+1

73

73



EXAMPLE

- Prove that: $2n + 1 \le 2^n$ for all $n \ge 3$
- **Basis step:**
  - $n = 3$: $2 * 3 + 1 \le 2^3 \Leftrightarrow 7 \le 8$ TRUE
- **Inductive step:**
  - Assume inequality is true for n, and prove it for (n+1):

$2n + 1 \le 2^n$ must prove: $2(n + 1) + 1 \le 2^{n+1}$

$2(n + 1) + 1 = (2n + 1) + 2 \le 2^n + 2 \le$

$\le 2^n + 2^n = 2^{n+1}$, since $2 \le 2^n$ for $n \ge 1$

74

74

THANKS

75

75

INSERTION SORT

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    do key ← A[j]
       ▷ Insert A[j] into the sorted
            sequence A[1 . . j − 1].
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

Cost	Times	Total

76

76

INSERTION SORT

INSERTION-SORT(A)	*cost*	*times*
for $j \leftarrow 2$ **to** $length[A]$	c_1	n
do $key \leftarrow A[j]$	c_2	$n - 1$
▷ Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n - 1$
$i \leftarrow j - 1$	c_4	$n - 1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow key$	c_8	$n - 1$

77

77

Θ-NOTATION

DEF:

$\Theta(g(n)) = \{ f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0 \}$

Basic manipulations:

- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

78

CASE STUDY: INSERTION SORT

Count the number of times each line will be executed:

```
                                   Num Exec.
for i = 2 to n                     (n-1) + 1
    key = A[i]                n-1
    j = i - 1                          n-1
    while j > 0 AND A[j] > key          ?
        A[j+1] = A[j]                   ?
        j = j -1                        ?
    A[j+1] = key                       n-1
```

79

MEASURING COMPLEXITY AGAIN

- The *worst case running time* of an algorithm is the function defined by the maximum number of steps taken on any instance of size n.
- The *best case running time* of an algorithm is the function defined by the minimum number of steps taken on any instance of size n.
- The *average-case running time* of an algorithm is the function defined by an average number of steps taken on any instance of size n.
- Which of these is the best to use?

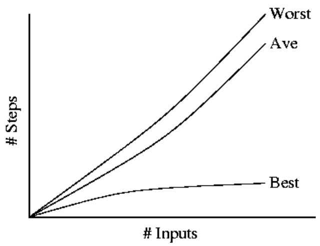
80

AVERAGE CASE ANALYSIS

- Drawbacks
  - Based on a probability distribution of input instances
  - How do we know if distribution is correct or not?
- Usually more complicated to compute than worst case running time
  - Often worst case running time is comparable to average case running time(see next graph)
  - Counterexamples to above:
    - Quicksort
    - simplex method for linear programming

81

BEST, WORST, AND AVERAGE CASE



82

WORST CASE ANALYSIS

- Typically much simpler to compute as we do not need to "average" performance on many inputs
  - Instead, we need to find and understand an input that causes worst case performance
- Provides guarantee that is independent of any assumptions about the input
- Often reasonably close to average case running time
- The standard analysis performed

83