ICS141: Discrete Mathematics for Computer Science I

Dept. Information & Computer Sci., University of Hawaii

Jan Stelovsky
based on slides by Dr. Baek and Dr. Still
Originals by Dr. M. P. Frank and Dr. J.L. Gross
Provided by McGraw-Hill

University of Hawaii

Lecture 16

Chapter 3. The Fundamentals

3.3 Complexity of Algorithms

3.4 The Integers and Division

University of Hawaii

3.3 Complexity of Algorithms

- An algorithm must always produce the correct answer, and should be efficient.
- How can the efficiency of an algorithm be analyzed?
- The ***algorithmic complexity*** of a computation is, most generally, a measure of how *difficult* it is to perform the computation.
- That is, it measures some aspect of the *cost* of computation (in a general sense of "cost").
  - Amount of resources required to do a computation.

Our focus

- Some of the most common complexity measures:
  - "Time" complexity: # of operations or steps required
  - "Space" complexity: # of memory bits required

University of Hawaii

Complexity Depends on Input

- Most algorithms have different complexities for inputs of different sizes.
  - *E.g.* searching a long list typically takes more time than searching a short one.
- Therefore, complexity is usually expressed as a *function* of the input size.
  - This function usually gives the complexity for the *worst-case* input of any given length.

Worst-, Average- and Best-Case Complexity

University of Hawaii

- A ***worst-case complexity*** measure estimates the time required for the most time consuming input of each size.
- An ***average-case complexity*** measure estimates the average time required for input of each size.
- An ***best-case complexity*** measure estimates the least time consuming input of each size.

Example 1: Max algorithm

University of Hawaii

- **Problem:**

  Find the *simplest form* of the *exact* order of growth (Θ) of the *worst-case* time complexity of the *max* algorithm, assuming that each line of code takes some constant time every time it is executed (with possibly different times for different lines of code).

Complexity Analysis of *max*

University of Hawaii

procedure $max(a_1, a_2, \ldots, a_n\text{: integers})$

$v := a_1$ $\quad t_1$

for $i := 2$ **to** n $\quad t_2$

if $a_i > v$ **then** $v := a_i$ $\quad t_3$

return v $\quad t_4$

- First, what is an expression for the *exact* total worst-case time? (Not its order of growth.)
  - t_1: once
  - t_2: $n - 1 + 1$ times
  - t_3 (comparison): $n - 1$ times
  - t_4: once

Complexity Analysis (*cont.*)

University of Hawaii

- Worst-case time complexity

$$t(n) = t_1 + t_2 + t_3 + t_4$$
$$= 1 + (n - 1 + 1) + (n - 1) + 1$$
$$= 2n + 1$$

  - In terms of the number of comparisons made

$$t(n) = t_2 + t_3$$
$$= (n - 1 + 1) + (n - 1)$$
$$= 2n - 1$$

# of comparisons

- Now, what is the simplest form of the exact (Θ) order of growth of $t(n)$?

$$\boldsymbol{t(n) = \Theta(n)}$$

Example 2: Linear Search

- In terms of the number of comparisons

procedure *linear_search* (x: integer,
$a_1, a_2, \ldots, a_n$: distinct integers)

$i := 1$

while $(i \le n \wedge x \ne a_i)$ t_{11} & t_{12}

$i := i + 1$

if $i \le n$ **then** *location* $:= i$ t_2

else *location* $:= 0$

return *location*

Linear Search Analysis

- Worst case time complexity:

$$t(n) = t_{11} + t_{12} + t_2$$

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

# of comparisons

- Best case: $t(n) = t_{11} + t_{12} + t_2 = 1 + 1 + 1 = \Theta(1)$

- Average case, if item is present:

$$t(n) = \frac{3 + 5 + 7 + \cdots + (2n+1)}{n} = \frac{2(1 + 2 + \cdots + n) + n}{n}$$

$$= \frac{2[n(n+1)/2]}{n} + 1 = n + 2 = \Theta(n)$$

Example 3: Binary Search

procedure *binary_search* (x:integer, $a_1, a_2, \ldots, a_n$: distinct integers, sorted smallest to largest)

$i := 1$

$j := n$

while $i < j$ **begin**

$\quad m := \lfloor (i + j)/2 \rfloor$

$\quad$**if** $x > a_m$ **then** $i := m + 1$ **else** $j := m$ $\qquad t_1$

end $\qquad t_2$

if $x = a_i$ **then** $location := i$ **else** $location := 0$ $\qquad t_3$

return *location*

Key question:

How many loop iterations?

Binary Search Analysis

University of Hawaii

- Suppose that n is a power of 2, *i.e.*, $\exists k$: $n = 2^k$.
- Original range from $i = 1$ to $j = n$ contains n items.
- Each iteration: Size $j - i + 1$ of range is cut in half.
  - Size decreases as 2^k, 2^{k-1}, 2^{k-2},…
- Loop terminates when size of range is $1 = 2^0$ ($i = j$).
- Therefore, the number of iterations is: $k = \log_2 n$

$$t(n) = t_1 + t_2 + t_3$$

$$= (k+1) + k + 1 = 2k + 2 = 2\log_2 n + 2 = \Theta(\log_2 n)$$

- Even for $n \neq 2^k$ (not an integral power of 2), time complexity is still the same.

Analysis of Sorting Algorithms

- Check out

  Rosen 3.3 Example 5 and Example 6

  for worst-case time complexity of bubble sort and insertion sort algorithms in terms of the number of comparisons made.

Bubble Sort Analysis

University of Hawaii

procedure *bubble_sort* ($a_1, a_2, \ldots, a_n$: real numbers with $n \geq 2$)

  for $i := 1$ **to** $n - 1$

    for $j := 1$ **to** $n - i$

      if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_1, a_2, \ldots, a_n$ is in increasing order}

- Worst-case complexity in terms of the number of comparisons: $\Theta(n^2)$

Insertion Sort

procedure *insertion_sort* ($a_1, a_2, \ldots, a_n$: real numbers; $n \geq 2$)

for $j := 2$ **to** n

begin

$i := 1$

while $a_j > a_i$

$i := i + 1$

m := a_j

for $k := 0$ **to** $j - i - 1$

$a_{j-k} := a_{j-k-1}$

$a_i :=$ m

end $\{a_1, a_2, \ldots, a_n$ are sorted in increasing order$\}$

- Worst-case complexity in terms of the number of comparisons: $\Theta(n^2)$

Common Terminology for the Complexity of Algorithms

University of Hawaii

TABLE 1 Commonly Used Terminology for the Complexity of Algorithms.

Complexity	*Terminology*
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	$n \log n$ complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

Computer Time Examples

- Assume that time = 1 ns (10^{-9} second) per operation, problem size = *n* bits, and #ops is a function of *n*.

#ops(n)	$n = 10$ (1.25 bytes)	$n = 10^6$ (125 kB)
$\log_2 n$	3.3 ns	19.9 ns
n	10 ns	1 ms
$n \log_2 n$	33 ns	19.9 ms
n^2	100 ns	16 m 40 s
2^n	1.024 µs	$10^{301,004.5}$
$n!$	3.63 ms	Ouch!

Review: Complexity

University of Hawaii

- Algorithmic complexity = *cost* of computation.
- Focus on *time* complexity for our course.
  - Although space & energy are also important.
- Characterize complexity as a function of input size:
  - Worst-case, best-case, or average-case.
- Use orders-of-growth notation to concisely summarize the growth properties of complexity functions.
- Need to know
  - Names of specific orders of growth of complexity.
  - How to analyze the order of growth of time complexity for simple algorithms.

Tractable *vs.* Intractable

- A problem that is solvable using an algorithm with at most polynomial time complexity is called ***tractable*** (or *feasible*).
  P is the set of all tractable problems.
- A problem that cannot be solved using an algorithm with worst-case polynomial time complexity is called ***intractable*** (or *infeasible*).
- Note that $n^{1,000,000}$ is *technically* tractable, but really very hard. $n^{\log \log \log n}$ is *technically* intractable, but easy. Such cases are rare though.

P *vs.* NP

University of Hawaii

- **NP** is the set of problems for which there exists a tractable algorithm for *checking a proposed solution* to tell if it is correct.
- We know that **P⊆NP**, but the most famous unproven conjecture in computer science is that this inclusion is *proper*.
  - *i.e.*, that **P⊂NP** rather than **P=NP**.
- Whoever first proves this will be famous!

(or disproves it!)

3.4 The Integers and Division

University of Hawaii

- Of course, you already know what the integers are, and what division is…
- **But:** There are some specific notations, terminology, and theorems associated with these concepts which you may not know.
- These form the basics of *number theory*.
  - *Number theory* is vital in many today important algorithms (hash functions, cryptography, digital signatures,…).

Divides, Factor, Multiple

- Let $a,b \in \mathbf{Z}$ with $a \neq 0$.
- **Definition:** $a|b \Leftrightarrow$ "a ***divides*** b" $\Leftrightarrow (\exists c \in \mathbf{Z}: b = ac)$
  "There is an integer c such that c times a equals b."
  - Example: $3|-12$ (**True**), but $3|7$ (**False**).
- If a divides b, then we say a is a ***factor*** or a ***divisor*** of b, and b is a ***multiple*** of a.
- E.g.: "b is even" $\Leftrightarrow 2|b$.

The Divides Relation

University of Hawaii

- **Theorem:** $\forall a,b,c \in \mathbf{Z}$:
  1. $a|0$
  2. $(a|b \wedge a|c) \rightarrow a|(b + c)$
  3. $a|b \rightarrow a|bc \;\; \forall c \in \mathbf{Z}$
  4. $(a|b \wedge b|c) \rightarrow a|c$

- **Corollary**: $\forall a,b,c \in \mathbf{Z}$
  - $(a|b \wedge a|c) \rightarrow a|(mb + nc),\ m,n \in \mathbf{Z}$

Proof of (2)

University of Hawaii

- Show $\forall a,b,c \in \mathbf{Z}: (a|b \wedge a|c) \rightarrow a|(b+c)$.
  - Let a, b, c be any integers such that $a|b$ and $a|c$, and show that $a|(b+c)$.
  - By definition of $|$, we know $\exists s\in\mathbf{Z}: b = as$, and $\exists t\in\mathbf{Z}: c = at$. Let s, t, be such integers.
  - Then $b + c = as + at = a(s + t)$, so $\exists u\in\mathbf{Z}: b + c = au$, namely $u = s + t$. Thus $a|(b+c)$.

The Division "Algorithm"

- It's really just a *theorem*, not an algorithm…
  - Only called an "algorithm" for historical reasons.
- **Theorem:** For any integer ***dividend*** a and ***divisor*** $d \in \mathbf{Z}^+$, there are **unique** integers ***quotient*** q and ***remainder*** $r \in \mathbf{N}$ such that $a = dq + r$ and $0 \leq r < d$. Formally, the theorem is:

$$\boxed{\forall a \in \mathbf{Z}, d \in \mathbf{Z}^+ : \exists! q, r \in \mathbf{Z} : 0 \leq r < d,\ a = dq + r}$$

- We can find q and r by: $q = \lfloor a/d \rfloor$, $r = a - dq$