DISMATH
Discrete Mathematics
Methods of Proof, Algorithms, and Number Theory

De La Salle University

September 2014

Overview

- Methods of Proof

Overview

- Methods of Proof
- Algorithms

Overview

- Methods of Proof
- Algorithms
- The Growth of Functions

Overview

- Methods of Proof
- Algorithms
- The Growth of Functions
- Complexity of Algorithms

Overview

- Methods of Proof
- Algorithms
- The Growth of Functions
- Complexity of Algorithms
- Integers and Division

Overview

- Methods of Proof
- Algorithms
- The Growth of Functions
- Complexity of Algorithms
- Integers and Division
- Number Theory and Applications

Why Study Proofs?

Computing systems are doing so much:



How can we guarantee they work?

Why Study Proofs?

Why not just testing?

- Integrates well with programming
- No new languages, tools required
- Conclusive evidence for bugs

Why Study Proofs?

Why not just testing?

- Integrates well with programming
- No new languages, tools required
- Conclusive evidence for bugs

Because…

- Difficult to assess coverage
- Cannot demonstrate absence of bugs
- No guarantees for safety-critical systems

Why Study Proofs?

Formal Verification

1. SOFTWARE:
If you want to debug a program beyond a doubt, prove that it's bug-free! Deduction and proof provide universal guarantees.

2. HARDWARE:
Proof-theory has recently also been shown to be useful in discovering bugs in pre-production hardware.

Methods of Proof

- Direct Proof

Methods of Proof

- Direct Proof
- Proof by Contraposition (Indirect)

Methods of Proof

- Direct Proof
- Proof by Contraposition (Indirect)
- Vacuous and Trivial Proof

Methods of Proof

- Direct Proof
- Proof by Contraposition (Indirect)
- Vacuous and Trivial Proof
- Proof by Contradiction (Indirect)

Methods of Proof

- Direct Proof
- Proof by Contraposition (Indirect)
- Vacuous and Trivial Proof
- Proof by Contradiction (Indirect)
- Proof by Equivalence

Direct Proof

- In a conditional statement $p \rightarrow q$, assume that p is true and use definitions and previously proven theorems, to show that q must also be true.

Direct Proof

- In a conditional statement $p \rightarrow q$, assume that p is true and use definitions and previously proven theorems, to show that q must also be true.
- Ex. Give a direct proof of the theorem:
  "If n is an odd integer, then n^2 is odd."

Proof by Contraposition

- We take $\neg q$ as a hypothesis, and using definitions, and previously proven theorems, we show that $\neg p$ must follow.

Proof by Contraposition

- We take $\neg q$ as a hypothesis, and using definitions, and previously proven theorems, we show that $\neg p$ must follow.
- Ex. Prove that if n is an integer and $3n + 2$ is odd, then n is odd.

☞Exercise

- Prove that if $n = ab$, where a and b are positive integers, then $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$.

Vacuous and Trivial Proofs

- Vacuous proof
  Show that p is false, because $p \rightarrow q$ must be true when p is false.
- $\neg p \rightarrow (p \rightarrow q)$

Vacuous and Trivial Proofs

- Vacuous proof
  Show that p is false, because $p \rightarrow q$ must be true when p is false.
- $\neg p \rightarrow (p \rightarrow q)$
- Trivial proof
  Show that q is true, it follows that $p \rightarrow q$ must also be true.
- $q \rightarrow (p \rightarrow q)$

☞ Exercise

- Prove the statement: If there are 30 students enrolled in this course this semester, then $6^2 = 36$.

Exercise

- Prove the statement: If there are 30 students enrolled in this course this semester, then $6^2 = 36$.
- Prove the statement. If 6 is a prime number, then $6^2 = 30$.

☞ Exercise

- Prove that if n is an integer and n^2 is odd, then n is odd.

Proof by Contradiction

- Show that assuming $\neg p$ is true leads to a contradiction.

Proof by Contradiction

- Show that assuming $\neg p$ is true leads to a contradiction.
- Ex. Prove that $\sqrt{2}$ is irrational by giving a proof by contradiction.

☞ Exercise

- Give a proof by contradiction of the theorem "If $3n + 2$ is odd, then n is odd."

Proof by Equivalence

- To prove a theorem that is a biconditional statement, $p \leftrightarrow q$, we show that $p \rightarrow q$ and $q \rightarrow p$ are both true.

Proof by Equivalence

- To prove a theorem that is a biconditional statement, $p \leftrightarrow q$, we show that $p \rightarrow q$ and $q \rightarrow p$ are both true.
- $(p \leftrightarrow q) \leftrightarrow [(p \rightarrow q) \wedge (q \rightarrow p)]$

Exercise

- Prove the theorem
  "If n is a positive integer, then n is odd if and only if n^2 is odd."

Exercise

- Show that these statements about the integer n are equivalent:

  P_1 : n is even.

  P_2 : $n-1$ is odd.

  P_3 : n^2 is even.

Exercise

- T/F: "Every positive integer is the sum of the squares of two integers".

Algorithm

Algorithm

- Algorithm
  a finite set of precise instructions for performing a computation or for solving a problem.

Algorithm

- Algorithm
  a finite set of precise instructions for performing a computation or for solving a problem.
- Ex. Describe an algorithm for finding the maximum (largest) value in a finite sequence of integers.

Finding the Maximum Element Algorithm

1. Set the temporary maximum equal to the first integer in the sequence.

Finding the Maximum Element Algorithm

1. Set the temporary maximum equal to the first integer in the sequence.
2. Compare the next integer in the sequence to the temporary maximum, if larger, set the temporary maximum equal to this integer.

Finding the Maximum Element Algorithm

1. Set the temporary maximum equal to the first integer in the sequence.
2. Compare the next integer in the sequence to the temporary maximum, if larger, set the temporary maximum equal to this integer.
3. Repeat the previous step if there are more integers in the sequence.

Finding the Maximum Element Algorithm

1. Set the temporary maximum equal to the first integer in the sequence.
2. Compare the next integer in the sequence to the temporary maximum, if larger, set the temporary maximum equal to this integer.
3. Repeat the previous step if there are more integers in the sequence.
4. Stop when there are no integers left in the sequence.

Pseudocode

- high-level description of an algorithm that uses the structural conventions of a programming language, but is intended for human reading.

Pseudocode

- high-level description of an algorithm that uses the structural conventions of a programming language, but is intended for human reading.
- computer programs can be produced in any computer language using the pseudocode description as a starting point.

Finding the Maximum Element Pseudocode

PROCEDURE max(a_1, a_2, ..., a_n : integers)

  max $= a_1$

  for $i = 2$ to n

    if $max < a_j$ then $max = a_j$

{Output: max is the largest element}

Preconditions and Postconditions

- Preconditions
  statements that describe valid input

Preconditions and Postconditions

- Preconditions
  statements that describe valid input
- Ex. $(a_1, a_2, \ldots, a_n) \in \mathcal{Z}$

Preconditions and Postconditions

- Preconditions
  statements that describe valid input
- Ex. $(a_1, a_2, \ldots, a_n) \in \mathcal{Z}$
- Postconditions
  conditions that the ouput should satisfy when the program has run

Preconditions and Postconditions

- Preconditions
  statements that describe valid input
- Ex. $(a_1, a_2, \ldots, a_n) \in \mathcal{Z}$
- Postconditions
  conditions that the ouput should satisfy when the program has run
- Ex. Output: max is the largest element

Properties of Algorithm

- Input
  An algorithm has input values from a specified set.

Properties of Algorithm

- Input
  An algorithm has input values from a specified set.
- Output
  From each set of input values an algorithm produces output values from a specified set.

Properties of Algorithm

- Input
  An algorithm has input values from a specified set.
- Output
  From each set of input values an algorithm produces output values from a specified set.
- Definiteness
  The steps of an algorithm must be defined precisely.

Properties of Algorithm

- Correctness
  An algorithm should produce the correct output values for each set of input values.

Properties of Algorithm

- Correctness

  An algorithm should produce the correct output values for each set of input values.
- Finiteness

  An algorithm should produce the desired output after a finite number of steps.

Properties of Algorithm

- Correctness

  An algorithm should produce the correct output values for each set of input values.
- Finiteness

  An algorithm should produce the desired output after a finite number of steps.
- Generality

  The procedure should be applicable for all problems of the desired form, not just for a particular set of input.

Finding the Maximum Element Algorithm Sample Program

Searching Algorithms

- The problem of locating an element in an ordered list.

Searching Algorithms

- The problem of locating an element in an ordered list.
- Locate an element x in a list of distinct elements $a_1, a_2, \ldots, a_n$, or determine that it is not in the list.

Searching Algorithms

- The problem of locating an element in an ordered list.
- Locate an element x in a list of distinct elements $a_1, a_2, \ldots, a_n$, or determine that it is not in the list.
- Solution: the location of the term in the list that equals x (that is, i is the solution if $x = a_i$) and is 0 if x is not in the list.

Linear Search Algorithm

PRECONDITION: Linear search (x : integer, a_0, a_1,. . . , a_n : distinct integers)

- 1. Compare x and a_0. If $x = a_0$, location $= 0$, else proceed to next element.

POSTCONDITION: Output the location.

Linear Search Algorithm

PRECONDITION: Linear search (x : integer, a_0, a_1,. . . , a_n : distinct integers)

- 1. Compare x and a_0. If $x = a_0$, location $= 0$, else proceed to next element.
- 2. Repeat step 1 while a match has not been found and there are still elements.

POSTCONDITION: Output the location.

Linear Search Algorithm

PRECONDITION: Linear search (x : integer, a_0, a_1,. . . , a_n : distinct integers)

- 1. Compare x and a_0. If $x = a_0$, location $= 0$, else proceed to next element.
- 2. Repeat step 1 while a match has not been found and there are still elements.
- 3. Output the location if a match is found, else location = -1 signifying not found.

POSTCONDITION: Output the location.

Linear Search Pseudocode

PROCEDURE linear search
(PRECONDITION: x : integer, a_0, a_1, . . ., a_{n-1} : distinct integers)

$i = 0$

while ($i < n$ and $x \neq a_i$)

$i = i + 1$

if $i < n$ then $location = i$

else $location = -1$

{POSTCONDITION: location is the subscript of the term that equals x , or is –1 if x is not found}

Linear Search Sample Program

Binary Search Algorithm

PRECONDITION: Binary search (x : integer, a_0, a_1,. . . , a_{n-1} : sorted integers)

- 1. Compare x to the middle term of the list. If x is larger, choose the upper half, else choose the lower half.

POSTCONDITION: Output the location.

Binary Search Algorithm

PRECONDITION: Binary search (x : integer, a_0, a_1,. . . , a_{n-1} : sorted integers)

- 1. Compare x to the middle term of the list. If x is larger, choose the upper half, else choose the lower half.
- 2. Repeat step 1 while a match has not been found and there are still elements.

POSTCONDITION: Output the location.

Binary Search Algorithm

PRECONDITION: Binary search (x : integer, a_0, a_1,. . . , a_{n-1} : sorted integers)

- 1. Compare x to the middle term of the list. If x is larger, choose the upper half, else choose the lower half.
- 2. Repeat step 1 while a match has not been found and there are still elements.
- 3. Output the location if a match is found, else location = -1 signifying not found.

POSTCONDITION: Output the location.

Example

- Search for 19 in the list
  1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

Binary Search Pseudocode

PRECONDITION: Binary search (x : integer, a_0, a_1,. . . , a_{n-1} : sorted integers)

$i = 0$ {i is left endpoint of search interval}

$j = n - 1$ {j is right endpoint of search interval} while $i < j$

{

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

if $x > a_m$ then $i = m + 1$

else $j = m$

}

if $x = a_i$ then $location = i$

else $location := -1$

POSTCONDITION: Output the location.

Binary Search Sample Program

Sorting Algorithms

- The problem of putting elements in increasing order.

Sorting Algorithms

- The problem of putting elements in increasing order.
- Given a list of elements of a set, sort in increasing order.

Sorting Algorithms

- The problem of putting elements in increasing order.
- Given a list of elements of a set, sort in increasing order.
- Solution: bubble sort, insertion sort, etc.

Bubble Sort Algorithm

PRECONDITION: Bubble Sort (a_1, a_2,..., a_n : real numbers with $n \geq 2$)

- 1. Successively compare adjacent elements.

POSTCONDITION: Output the elements a_1, a_2,..., a_n in increasing order.

Bubble Sort Algorithm

PRECONDITION: Bubble Sort (a_1, a_2,. . . , a_n : real numbers with $n \geq 2$)

- 1. Successively compare adjacent elements.
- 2. Interchange elements if they are in the wrong order.

POSTCONDITION: Output the elements a_1, a_2,. . . , a_n in increasing order.

Bubble Sort Algorithm

PRECONDITION: Bubble Sort (a_1, a_2,. . . , a_n : real numbers with $n \geq 2$)

- 1. Successively compare adjacent elements.
- 2. Interchange elements if they are in the wrong order.
- 3. Repeat until there are elements.

POSTCONDITION: Output the elements a_1, a_2,. . . , a_n in increasing order.

Bubble Sort Algorithm

PRECONDITION: Bubble Sort (a_1, a_2,. . . , a_n : real numbers with $n \geq 2$)

- 1. Successively compare adjacent elements.
- 2. Interchange elements if they are in the wrong order.
- 3. Repeat until there are elements.
- 4. Output the list of elements in increasing order.

POSTCONDITION: Output the elements a_1, a_2,. . . , a_n in increasing order.

Bubble Sort Pseudocode

PROCEDURE bubble Sort
(PRECONDITION: a_1, a_2,..., 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}

{POSTCONDITION: Output the elements a_1, a_2,..., a_n in increasing order.}

Insertion Sort Algorithm

PRECONDITION: Insertion Sort (a_1, a_2,..., a_n : real numbers with $n \geq 2$)

- 1. Compare second element a_2 with the first element a_1.

POSTCONDITION: Output the elements a_1, a_2,..., a_n in increasing order.

Insertion Sort Algorithm

PRECONDITION: Insertion Sort (a_1, a_2,. . . , a_n : real numbers with $n \geq 2$)

- 1. Compare second element a_2 with the first element a_1.
- 2. If a_2 is smaller, place it before a_1 else place it after a_1.

POSTCONDITION: Output the elements a_1, a_2,. . . , a_n in increasing order.

Insertion Sort Algorithm

PRECONDITION: Insertion Sort (a_1, a_2,..., a_n : real numbers with $n \geq 2$)

- 1. Compare second element a_2 with the first element a_1.
- 2. If a_2 is smaller, place it before a_1 else place it after a_1.
- 3. Repeat until there are elements.

POSTCONDITION: Output the elements a_1, a_2,..., a_n in increasing order.

Insertion Sort Algorithm

PRECONDITION: Insertion Sort (a_1, a_2,. . . , a_n : real numbers with $n \geq 2$)

- 1. Compare second element a_2 with the first element a_1.
- 2. If a_2 is smaller, place it before a_1 else place it after a_1.
- 3. Repeat until there are elements.
- 4. Output the list of elements in increasing order.

POSTCONDITION: Output the elements a_1, a_2,. . . , a_n in increasing order.

Insertion Sort Pseudocode

PROCEDURE Insertion Sort

PRECONDITION: a_1, a_2,$\ldots$, a_n : real numbers with $n \geq 2$

  for $j = 2$ to n

  {

    $i = 1$

  while $a_j > a_i$

      $i = i + 1$

    $m = a_j$

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

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

    $a_i = m$

  }

{POSTCONDITION: Output the elements a_1, a_2,$\ldots$, a_n in increasing order.}

Greedy Algorithm

- selects the best choice at each step, instead of considering all sequences of steps that may lead to an optimal. solution.

Greedy Algorithm

- selects the best choice at each step, instead of considering all sequences of steps that may lead to an optimal. solution.
- applied in optimization problems where a solution to the given problem either minimizes or maximizes the value of some parameter.

☞ Exercise

- Consider the problem of making n cents change with quarters, dimes, nickels, and pennies, and using the least total number of coins.

Greedy Change-Making Algorithm Pseudocode

PROCEDURE change
PRECONDITION: c_1, $c_2, \ldots,$ c_n values of denominations of coins, where $c_1 > c_2 > \ldots > c_n$; $n \in \mathcal{Z}^+$

    for $i = 1$ to r
    while $n \geq c_i$
        add a coin with value c_i to the change
        $n = n - c_i$
    endwhile
    endfor
    {POSTCONDITION: Output the minimum number of coins.}

Growth of Functions

- The growth of functions is often described using Big-O Notation.

The constants C and k in the definition of big-O notation are called <u>witnesses</u>.

Growth of Functions

- The growth of functions is often described using Big-O Notation.
- Definition: Let f and g be functions from $\mathcal{R} \to \mathcal{R}$; $f(x)$ is $O(g(x))$ if there are constants C and k such that

$$|f(x)| \leqslant C\,|g(x)|$$

whenever $x > k$.

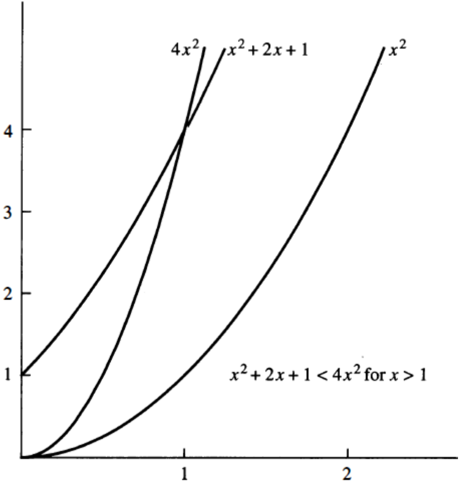
The constants C and k in the definition of big-O notation are called <u>witnesses</u>.

Example

- Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.

∘ A useful approach for finding a pair of witnesses is to first select a value of k for which the size of $|f(x)|$ can be readily estimated when $x > k$.

Illustration



Drill

- Show that $7x^2$ is $O(x^3)$.

Drill

- Show that $7x^2$ is $O(x^3)$.
- Show that n^2 is not $O(n)$.

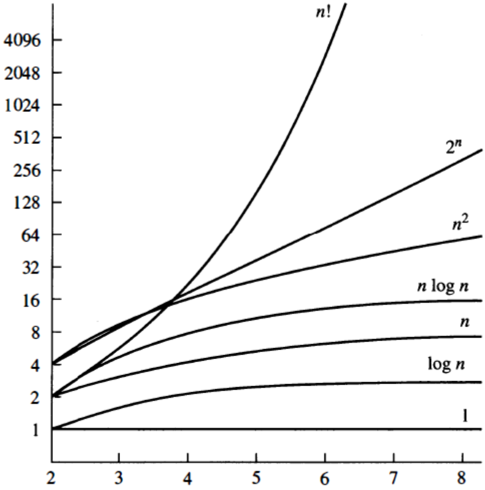
Drill

- How can big- O notation be used to estimate the sum of the first n positive integers?

Drill

- How can big- O notation be used to estimate the sum of the first n positive integers?
- Give big- O estimates for the factorial function and the logarithm of the factorial function.

Common Big-O Estimates



Big-Omega and Big-Theta Notation

- Big-O notation does not provide a lower bound for the size of $f(x)$.

Big-Omega and Big-Theta Notation

- Big-O notation does not provide a lower bound for the size of $f(x)$.
- For the lower bound, we use big-Omega (big-Ω) notation.

Big-Omega and Big-Theta Notation

- Big-O notation does not provide a lower bound for the size of $f(x)$.
- For the lower bound, we use big-Omega (big-Ω) notation.
- For the both lower and upper bound, we use big-Theta (big-Θ) notation.

Algorithm Time Complexity

- can be expressed in terms of the number of operations used by the algorithm when the input has a particular size.

Algorithm Time Complexity

- can be expressed in terms of the number of operations used by the algorithm when the input has a particular size.
- the number of comparisons will be used as the measure of the time complexity of the algorithm, because comparisons are the basic operations used.

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

Example

- Describe the time complexity of algorithm for finding the maximum element in a set.

PROCEDURE max(a_1, a_2, ..., a_n : integers)

  max $= a_1$

  for $i = 2$ to n

    if $max < a_j$ then $max = a_j$

{Output: max is the largest element}

Drill

- What is the worst-case complexity of the bubble sort in terms of the number of comparisons made?

PROCEDURE bubble Sort
(PRECONDITION: a_1, a_2,..., 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}

{POSTCONDITION: Output the elements a_1, a_2,..., a_n in increasing order.}

Division and Modulo Operator

- Let a be an integer and d a positive integer. Then there are unique integers q and r, with $0 \leq r < d$, such that $a = dq + r$.

Division and Modulo Operator

- Let a be an integer and d a positive integer. Then there are unique integers q and r, with $0 \leq r < d$, such that $a = dq + r$.

  $q = a \, div \, d \qquad r = a \mod d$

Division and Modulo Operator

- Let a be an integer and d a positive integer. Then there are unique integers q and r, with $0 \le r < d$, such that $a = dq + r$.

  $q = a\, div\, d \qquad r = a \mod d$

- In the equality given, d is called the divisor, a is called the dividend, q is called the quotient, and r is called the remainder.

Example

- What are the quotient and remainder when 101 is divided by 11 ?

Example

- What are the quotient and remainder when 101 is divided by 11 ?
- What are the quotient and remainder when -11 is divided by 3?

Modulo Equivalence

- If a and b are integers and m is a positive integer, then a is congruent to b modulo m if m divides $a - b$.

Modulo Equivalence

- If a and b are integers and m is a positive integer, then a is congruent to b modulo m if m divides $a - b$.
- We use the notation $a \equiv b(\mod m)$ to indicate that a is congruent to b modulo m.

$$a \equiv b \mod m \text{ iff. } m|(a-b)$$

Example

- Determine whether 17 is congruent to 5 modulo 6 and whether 24 and 14 are congruent modulo 6.

Applications

- Cryptology: the study of secret messages.
- Ex. Caesar Cipher: Messages are made secret by shifting each letter three letters forward in the alphabet.

Applications

- Cryptology: the study of secret messages.
- Ex. Caesar Cipher: Messages are made secret by shifting each letter three letters forward in the alphabet.
- Caesar's encryption method can be represented by the function f that assigns to the nonnegative integer p, $p \leq 25$, the integer f(p) in the set $\{0, 1, 2, ..., 25\}$ with

$$f(p) = (p + 3) \mod 26$$

Example

- What is the secret message produced from the message "MEET YOU IN DLSU" using the Caesar cipher?

Example

- What is the secret message produced from the message "MEET YOU IN DLSU" using the Caesar cipher?
- PHHW BRX LQ GOVY

Example

- What is the secret message produced from the message "MEET YOU IN DLSU" using the Caesar cipher?
- PHHW BRX LQ GOVY

$$f^{-1}(p) = (p-3) \mod 26$$

Representation of Integers

- Let b be a positive integer greater than 1. Then if n is a positive integer, it can be expressed uniquely in the form

$$n = a_k b^k + a_{k-1} b^{k-1} + ... + a_1 b + a_0$$

Representation of Integers

- Let b be a positive integer greater than 1. Then if n is a positive integer, it can be expressed uniquely in the form

$$n = a_k b^k + a_{k-1} b^{k-1} + ... + a_1 b + a_0$$

- where k is a nonnegative integer, $a_0, a_1, \ldots, a_k$ are nonnegative integers less than b, and $a_k \neq 0$.

Example

- What is the decimal expansion of the integer that has $(101011111)_2$ as its binary expansion?

Example

- What is the decimal expansion of the integer that has $(101011111)_2$ as its binary expansion?
- What is the decimal expansion of the hexadecimal expansion of $(2AEOB)_{16}$?

Base Conversion Algorithm

1. Divide n by b to obtain a quotient and remainder, $n = bq_0 + a_0$, $0 \leq a_0 < b$

Base Conversion Algorithm

1. Divide n by b to obtain a quotient and remainder, $n = bq_0 + a_0$, $0 \leq a_0 < b$
2. The remainder, a_0, is the rightmost digit in the base b expansion of n.

Base Conversion Algorithm

1. Divide n by b to obtain a quotient and remainder, $n = bq_0 + a_0$, $0 \leq a_0 < b$
2. The remainder, a_0, is the rightmost digit in the base b expansion of n.
3. Divide q_0 by b to obtain $q_0 = bq_1 + a_1$, $0 \leq a_1 < b$

Base Conversion Algorithm

1. Divide n by b to obtain a quotient and remainder, $n = bq_0 + a_0$, $0 \leq a_0 < b$
2. The remainder, a_0, is the rightmost digit in the base b expansion of n.
3. Divide q_0 by b to obtain $q_0 = bq_1 + a_1$, $0 \leq a_1 < b$
4. The remainder, a_1, is the second digit from the right in the base b expansion of n.

Base Conversion Algorithm

1. Divide n by b to obtain a quotient and remainder, $n = bq_0 + a_0$, $0 \leq a_0 < b$
2. The remainder, a_0, is the rightmost digit in the base b expansion of n.
3. Divide q_0 by b to obtain $q_0 = bq_1 + a_1$, $0 \leq a_1 < b$
4. The remainder, a_1, is the second digit from the right in the base b expansion of n.
5. Continue process until we obtain a quotient equal to zero.

Example

- Find the base 8, or octal, expansion of $(12345)_10$.

Example

- Find the base 8, or octal, expansion of $(12345)_{1}0$.
- Find the hexadecimal expansion of $(177130)_{1}0$.

Example

- Find the base 8, or octal, expansion of $(12345)_{1}0$.
- Find the hexadecimal expansion of $(177130)_{1}0$.
- Find the binary expansion of $(241)_{1}0$.

Base b Expansions Algorithm Pseudocode

PROCEDURE base b expansion
PRECONDITION: (n: positive integer);

$q = n$

$k = 0$

while $q \neq 0$

{

$a_k = q mod b$

$q = \lfloor q/b \rfloor$

$k = k + 1$

}

{POSTCONDITION: the base b expansion of n is $(a_{k-1}, \ldots, a_1, a_0)_b$}

Euclidean Algorithm

A method for computing the greatest common divisor of two integers.

- Let $a = bq + r$, where a, b, q, and r are integers. Then $gcd(a, b) = gcd(b, r)$.

Euclidean Algorithm

A method for computing the greatest common divisor of two integers.

- Let $a = bq + r$, where a, b, q, and r are integers. Then $gcd(a, b) = gcd(b, r)$.

  $gcd(a, b) = gcd(r_o, r_1) = gcd(r_1, r_2) = \ldots$
  $= gcd(r_{n-2}, r_{n-d} = gcd(r_{n_1}, r_n) = gcd(r_n, 0) = r_n.$

Euclidean Algorithm

A method for computing the greatest common divisor of two integers.

- Let $a = bq + r$, where a, b, q, and r are integers. Then $gcd(a, b) = gcd(b, r)$.

  $gcd(a, b) = gcd(r_o, r_1) = gcd(r_1, r_2) = \ldots$
  $= gcd(r_{n-2}, r_{n-d} = gcd(r_{n_1}, r_n) = gcd(r_n, 0) = r_n.$
- The greatest common divisor is the last nonzero remainder in the sequence of divisions.

Example

- Find the greatest common divisor of 414 and 662 using the Euclidean algorithm.

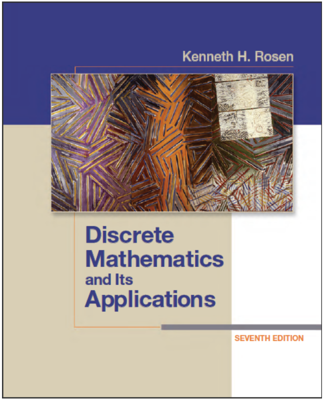
Euclidean Algorithm Pseudocode

PROCEDURE GCD
PRECONDITION: (a, b: positive integers; a>b);

```
    x = a
    y = b
    while y ≠ 0
{
        r = x mod y
        x = y
        y = r
}
    {POSTCONDITION: GCD(a, b) is x }
```

Reference:

Rosen, K.H. Discrete Mathematics and Its Applications (7 ed.), New York, McGraw-Hill



Thank you for your attention!