Module #5 - Algorithms

University of Florida
Dept. of Computer & Information Science & Engineering

COT 3100
Applications of Discrete Structures
Dr. Michael P. Frank

Slides for a Course Based on the Text
Discrete Mathematics & Its Applications
(5th Edition)
by Kenneth H. Rosen
Chapter 2: More Fundamentals

- §2.1: Algorithms (Formal procedures)
- §2.2: Complexity of algorithms
  - Analysis using order-of-growth notation.
- §2.3: The Integers & Division
  - Some basic number theory.
- §2.6: Matrices
  - Some basic linear algebra.
§2.1: Algorithms

• The foundation of computer programming.
• Most generally, an algorithm just means a definite procedure for performing some sort of task.
• A computer program is simply a description of an algorithm in a language precise enough for a computer to understand, requiring only operations the computer already knows how to do.
• We say that a program implements (or “is an implementation of”) its algorithm.
Module #5 - Algorithms

Algorithms You Already Know

• Grade school arithmetic algorithms:
  – How to add any two natural numbers written in decimal on paper using carries.
  – Similar: Subtraction using borrowing.
  – Multiplication & long division.
• Your favorite cooking recipe.
• How to register for classes at UF.
Some common programming languages:

- **Newer**: Java, C, C++, Visual Basic, JavaScript, Perl, Tcl, Pascal
- **Older**: Fortran, Cobol, Lisp, Basic
- Assembly languages, for low-level coding.

In this class we will use an informal, Pascal-like “pseudo-code” language.

You should know at least 1 real language!
Algorithm Example (English)

- Task: Given a sequence \( \{a_i\} = a_1, \ldots, a_n \), \( a_i \in \mathbb{N} \), say what its largest element is.
- Set the value of a temporary variable \( v \) (largest element seen so far) to \( a_1 \)'s value.
- Look at the next element \( a_i \) in the sequence.
- If \( a_i > v \), then re-assign \( v \) to the number \( a_i \).
- Repeat previous 2 steps until there are no more elements in the sequence, & return \( v \).
Executing an Algorithm

- When you start up a piece of software, we say the program or its algorithm are being *run* or *executed* by the computer.
- Given a description of an algorithm, you can also execute it by hand, by working through all of its steps on paper.
- Before ~WWII, “computer” meant a *person* whose job was to run algorithms!
Executing the Max algorithm

- Let \( \{a_i\} = 7, 12, 3, 15, 8 \). Find its maximum...
- Set \( v = a_1 = 7 \).
- Look at next element: \( a_2 = 12 \).
- Is \( a_2 > v \)? Yes, so change \( v \) to 12.
- Look at next element: \( a_2 = 3 \).
- Is \( 3 > 12 \)? No, leave \( v \) alone....
- Is \( 15 > 12 \)? Yes, \( v = 15 \)....
### Algorithm Characteristics

Some important features of algorithms:

- **Input.** Information or data that comes in.
- **Output.** Information or data that goes out.
- **Definiteness.** Precisely defined.
- **Correctness.** Outputs correctly relate to inputs.
- **Finiteness.** Won’t take forever to describe or run.
- **Effectiveness.** Individual steps are all do-able.
- **Generality.** Works for many possible inputs.
- **Efficiency.** Takes little time & memory to run.
procedure name(argument: type)
variable := expression
informal statement
begin statements end
{comment}
if condition then statement [else statement]
for variable := initial value to final value statement
while condition statement
procname(arguments)

Not defined in book:
return expression
**procedure** *procname*(arg: type)

- Declares that the following text defines a procedure named *procname* that takes inputs (*arguments*) named *arg* which are data objects of the type *type*.
  
  – Example:

  ```plaintext
  procedure maximum(L: list of integers)
  [statements defining maximum…]
  ```
variable : = expression

• An assignment statement evaluates the expression expression, then reassigns the variable variable to the value that results.
  – Example:
    \[ v : = 3x+7 \]  (If \( x \) is 2, changes \( v \) to 13.)

• In pseudocode (but not real code), the expression might be informal:
  – \( x : = \)the largest integer in the list \( L \)
Informal statement

- Sometimes we may write a statement as an informal English imperative, if the meaning is still clear and precise: “swap x and y”
- Keep in mind that real programming languages never allow this.
- When we ask for an algorithm to do so-and-so, writing “Do so-and-so” isn’t enough!
  – Break down algorithm into detailed steps.
begin statements end

- Groups a sequence of statements together:

\[
\begin{align*}
\text{begin} \\
\text{statement 1} \\
\text{statement 2} \\
\text{...} \\
\text{statement n} \\
\text{end}
\end{align*}
\]

- Allows sequence to be used like a single statement.

- Might be used:
  - After a `procedure` declaration.
  - In an `if` statement after `then` or `else`.
  - In the body of a `for` or `while` loop.
Module #5 - Algorithms

{comment}

- Not executed (does nothing).
- Natural-language text explaining some aspect of the procedure to human readers.
- Also called a remark in some real programming languages.
- Example:
  - {Note that v is the largest integer seen so far.}
if \textit{condition} then \textit{statement}

- Evaluate the propositional expression \textit{condition}.
- If the resulting truth value is \textbf{true}, then execute the statement \textit{statement}; otherwise, just skip on ahead to the next statement.
- Variant: \texttt{if cond then stmt1 else stmt2}
  Like before, but iff truth value is \textbf{false}, executes \textit{stmt2}.
while \textit{condition} \textit{statement}

- \textit{Evaluate} the propositional expression \textit{condition}.
- If the resulting value is \textbf{true}, then execute \textit{statement}.
- Continue repeating the above two actions over and over until finally the \textit{condition} evaluates to \textbf{false}; then go on to the next \textit{statement}. 
while condition statement

- Also equivalent to infinite nested ifs, like so:

```plaintext
if condition
begin
  statement
if condition
begin
  statement
  ...(continue infinite nested if’s)
end
end
```
for \texttt{var} := \texttt{initial} to \texttt{final} \texttt{stmt}

- \texttt{Initial} is an integer expression.
- \texttt{Final} is another integer expression.
- Repeatedly execute \texttt{stmt}, first with variable \texttt{var} := \texttt{initial}, then with \texttt{var} := \texttt{initial}+1, then with \texttt{var} := \texttt{initial}+2, etc., then finally with \texttt{var} := \texttt{final}.
- What happens if \texttt{stmt} changes the value that \texttt{initial} or \texttt{final} evaluates to?
for \( \text{var} : = \text{initial} \) to \( \text{final} \) \( \text{stmt} \)

- For can be exactly defined in terms of while, like so:

\[
\begin{align*}
\text{begin} \\
\quad \text{var} : = \text{initial} \\
\quad \text{while} \: \text{var} \leq \text{final} \\
\quad \quad \text{begin} \\
\quad \quad \quad \text{stmt} \\
\quad \quad \quad \text{var} : = \text{var} + 1 \\
\quad \quad \text{end} \\
\quad \text{end} \\
\end{align*}
\]
A procedure call statement invokes the named *procedure*, giving it as its input the value of the *argument* expression.

Various real programming languages refer to procedures as *functions* (since the procedure call notation works similarly to function application \( f(x) \)), or as *subroutines*, *subprograms*, or *methods*.
Max procedure in pseudocode

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

\[
v := a_1 \quad \{ \text{largest element so far} \}
\]

\[
\text{for } i := 2 \text{ to } n \quad \{ \text{go thru rest of elems} \}
\]

\[
\quad \text{if } a_i > v \text{ then } v := a_i \quad \{ \text{found bigger?} \}
\]

\{at this point \( v \)'s value is the same as the largest integer in the list\}

**return** \( v \)
Another example task

• Problem of searching an ordered list.
  – Given a list $L$ of $n$ elements that are sorted into a definite order (e.g., numeric, alphabetical),
  – And given a particular element $x$,
  – Determine whether $x$ appears in the list,
  – and if so, return its index (position) in the list.

• Problem occurs often in many contexts.
• Let’s find an efficient algorithm!
Search alg. #1: Linear Search

**procedure** linear search

(x: integer, a₁, a₂, …, aₙ: distinct integers)

i : = 1

while (i ≤ n ∧ x ≠ aᵢ)
    i : = i + 1

if i ≤ n then location : = i
else location : = 0

return location {index or 0 if not found}
Search alg. #2: Binary Search

- Basic idea: On each step, look at the *middle* element of the remaining list to eliminate half of it, and quickly zero in on the desired element.
Search alg. #2: Binary Search

procedure binary search
    (x:integer, \(a_1, a_2, \ldots, a_n\): distinct integers)
    \(i := 1\) \{left endpoint of search interval\}
    \(j := n\) \{right endpoint of search interval\}
    \textbf{while} \(i < j\) \textbf{begin} \{while interval has >1 item\}
      \(m := \lfloor (i+j)/2 \rfloor\) \{midpoint\}
      \textbf{if} \(x > a_m\) \textbf{then} \(i := m+1\) \textbf{else} \(j := m\)
    \textbf{end}
    \textbf{if} \(x = a_i\) \textbf{then} \textit{location} := i \textbf{else} \textit{location} := 0
    \textbf{return} \textit{location}
Practice exercises

• 2.1.3: Devise an algorithm that finds the sum of all the integers in a list. [2 min]

• procedure sum(a_1, a_2, …, a_n: integers)
  \[ s := 0 \quad \{ \text{sum of elems so far} \} \]
  for \( i := 1 \) to \( n \) \{ go thru all elems \}
  \[ s := s + a_i \quad \{ \text{add current item} \} \]
  \{ at this point \( s \) is the sum of all items \}
  return \( s \)
Review §2.1: Algorithms

- Characteristics of algorithms.
- Pseudocode.
- Examples: Max algorithm, linear search & binary search algorithms.
- Intuitively we see that binary search is much faster than linear search, but how do we analyze the efficiency of algorithms formally?
- Use methods of \textit{algorithmic complexity}, which utilize the order-of-growth concepts from §1.8.
Module #5 - Algorithms

Review: \textit{max} algorithm

\begin{verbatim}
procedure max(a_1, a_2, \ldots, a_n: integers)
    v := a_1 \hspace{1cm} \{largest element so far\}
    for i := 2 to n \hspace{1cm} \{go thru rest of elems\}
        if a_i > v then v := a_i \hspace{1cm} \{found bigger?\}
    \{at this point v’s value is the same as the largest integer in the list\}
    return v
\end{verbatim}
Review: Linear Search

procedure linear search
  (x: integer, a₁, a₂, …, aₙ: distinct integers)
i := 1
while (i ≤ n ∧ x ≠ aᵢ)
  i := i + 1
if i ≤ n then location := i
else location := 0
return location {index or 0 if not found}
Review: Binary Search

• Basic idea: On each step, look at the *middle* element of the remaining list to eliminate half of it, and quickly zero in on the desired element.
procedure binary search
   (x:integer, a_1, a_2, ..., a_n: distinct integers)
   i := 1  \{left endpoint of search interval\}
   j := n  \{right endpoint of search interval\}
   while i<j begin  \{while interval has >1 item\}
      m := \lfloor (i+j)/2 \rfloor  \{midpoint\}
      if x>a_m then i := m+1 else j := m
   end
   if x = a_i then location := i else location := 0
return location