

# Algorithm Design Issues

---

- precise description of the problem.
- data analysis and creation of data descriptions.
- pseudo-code (general algorithm description)
- *modules*
  - scheme – the functions needed
  - OOP – objects needed
  - imperative languages: sequence of changes of state.
  - declarative languages: collection of declarations.

# Problem description

---

- In the business world this typically comes from the client.
  - often we need to help the client be precise!
- In research, the description is derived from hypotheses we hope to verify.
- In the classroom, this comes from the instructor
  - not always precise enough?

# Thought Exercises

---

- Develop a complete, precise, and concise description of the following problems:
  - we want to sort things.
  - we want to eliminate SPAM
  - we want to win at poker (or any game).
  - we want to verify (or disprove) that sorting the elements of a city-to-city cost matrix and then picking the smallest elements that cover all cities will provide an optimal tour. (TSP problem).

# Data analysis and Data Descriptions

---

- What data is provided as part of a problem instance? what form does it take?
- What data do we need to generate in order to model the problem?
- What are the *best* data structures for this application?
  - best might mean minimum memory usage.
  - best might mean leads to fastest algorithms.

# Pseudo Code

---

- Description of an algorithm in a mix of English and programming type expressions.
- Generally a mathematical description of the steps required to solve a problem.
- Some organizations develop formal pseudo-code languages (consistent problem description language within the organization).
  - code generators are possible.
  - this is almost programming...

# Correctness

---

- It should be possible to determine whether or not an algorithm is *correct* based on pseudo-code.
  - by *correct* we mean that the algorithm solves the problem for all possible inputs.
- Issues:
  - mathematical proof is sometimes possible.
  - *preconditions, postconditions, loop invariants*

# Pseudo-code example: Quicksort

---

*Quicksort(L):*

*given a list  $L = ( e_1 e_2 \dots e_n )$*

- 1. Create a list  $L_{\text{left}}$  that contains all elements  $e_i$  such that  $e_i < e_1$*
- 2. Create a list  $L_{\text{right}}$  that contains all elements  $e_i$  such that  $e_i > e_1$*
- 3. Create a list  $L_{\text{sorted}}$  by concatenating  $\text{Quicksort}(L_{\text{left}})$   $e_1$   $\text{Quicksort}(L_{\text{right}})$*

*Quicksort(L) =  $L_{\text{sorted}}$*

*Missing details? What about the base case?*

# Pseudo-Code Exercises

---

- Create a pseudo-code description of how you solved the TSP problem.
- pseudo-code for your tic-tac-toe player.
- bubblesort

# Modules

---

- The division of code in to separate *modules* is important!
  - minimize the interactions between modules whenever possible – simplifies testing and maintenance.
  - makes it possible for multiple people/groups to work on a single project.
  - generalization/abstraction of modules is often worthwhile/necessary.

# What is a module?

---

- Scheme: functions or collections of functions.
  - functional programming language
- Java, C++: Objects (collections of objects).
  - Object oriented programming.
- C: collections of C functions and data structures.
  - traditional (non-OOP) imperative programming.
- Prolog: collections of declarations
  - declarative programming.

# Module example we have seen

---

- Model – View – Controller (GUI programming).
  - we can consider the code that supports the view as separate from the code that supports the model.  
different modules.
  - We always need glue!
    - sometimes one module builds on another.
    - sometimes modules depend on each other.

# Module Exercise

---

- Create a brief description of the modules needed to create a student records management system.
- Functionality (functional description)
  - enter and change student records.
  - search for records
  - view records sorted by name, id, GPA, money owed, etc.