Welcome to Analysis of Algorithms (CS583 - 002)

Amarda Shehu

Fall 2017
The Importance of Designing and Analyzing Algorithms

Instructor: Amarda Shehu
Office: ENG #4452
Email: amarda AT gmu.edu
Web: cs.gmu.edu/~ashehu

CS583 Hours
Class: W 4:30 - 7:10 am
Place: Art and Design Building 2003
Office Hours: W 2:30 - 4:30 pm

TA:
Email: minjeti AT gmu.edu
ENG#5321, M 5:00 - 6:00 pm
Class Information

Outline of Today’s Class

The Importance of Designing and Analyzing Algorithms
- The Pervasiveness of Algorithms in Our Society
- What does It Take to Design Useful Algorithms?
Why are we Here?

- In *Calculation with Hindu Numerals*, 825 A.D., Muhammad ibn Musa al-Khwarizmi introduced Indian decimal system.
- The book was translated into Latin in 12th century as *Algoritmi de numero Indorum*.
- *Algorithm* was introduced to refer to a procedure for calculations with numbers.
- Short answer: We are here to design and analyze algorithms - procedures to solve useful problems.

*Figure*: Soviet stamp for al-Khwarizmi’s 1200th birthday. ©wikipedia.
Pattern matching algorithms and information searching algorithms are fundamental to our ability to parse through an overwhelming amount of information.

Google was founded on the ability of two Stanford University Ph.D. students, Sergey Brin and Larry Page, to design a fast information searching algorithm, BackRub. They quit school after that.
The Importance of Designing and Analyzing Algorithms

Orientation Software: Google maps, GPS navigators

- Path from Rice University, Houston, Texas to George Mason University, Fairfax, Virginia
- Path finding algorithms can be found in portable GPS navigators
- Most versions of the algorithm work with a static map (static conditions on the ground)

Figure: Output of a path finding algorithm.
The Importance of Designing and Analyzing Algorithms

The Pervasiveness of Algorithms in Our Society

What does It Take to Design Useful Algorithms?

Exploration, Search and Rescue, and Motion Planning

Figure: Ron Li and his research team are developing algorithms to help the rovers, Spirit and Opportunity, to navigate and find a safe path to a winter resting area. ©NASA.

Figure: Erion Plaku at Rice University is developing algorithms that plan paths for car-like robots in cluttered environments. ©E. Plaku.
Simulating Molecular Properties for Drug Design

Figure: Successful docking of HIV protease with a small inhibitor ligand. ©A. R. Leach.

Figure: Simulating the ability of proteins like ubiquitin to change shape as needed to accommodate and dock with different partner molecules. ©A. Shehu.
What is an Algorithm?

- Recipe, computational procedure that transforms input into output, tool to solve well-defined problems, sequence of instructions

An algorithm is a finite sequence of unambiguous instructions for completing a task, that:
What is an Algorithm?

- Recipe, computational procedure that transforms input into output, tool to solve well-defined problems, sequence of instructions

An algorithm is a finite sequence of unambiguous instructions for completing a task, that:

- when given an initial state (≥ 0 inputs),
- proceeds through a well-defined series of successive states,
- eventually terminating in an end-state (≥ outputs)

Figure:
What is an Algorithm?

- Recipe, computational procedure that transforms input into output, tool to solve well-defined problems, sequence of instructions

**Figure:**

- An algorithm is a finite sequence of unambiguous instructions for completing a task, that:
  - when given an initial state ($\geq 0$ inputs),
  - proceeds through a well-defined series of successive states,
  - eventually terminating in an end-state ($\geq$ outputs)

- Examples of algorithms?
What is an Algorithm?

- Recipe, computational procedure that transforms input into output, tool to solve well-defined problems, sequence of instructions

Figure:

- An algorithm is a finite sequence of unambiguous instructions for completing a task, that:
  - when given an initial state ($\geq 0$ inputs),
  - proceeds through a well-defined series of successive states,
  - eventually terminating in an end-state ($\geq$ outputs)

- Examples of algorithms?
Why do we Write Algorithms?

Given a real problem:

- Model it as a well-defined computational problem
- Is the solution to the problem *computable*?
Given a real problem:

- Model it as a well-defined computational problem
- Is the solution to the problem *computable*?
- If it is, does there exist an *efficient* algorithm for it?
Why do we Write Algorithms?

Given a real problem:

- Model it as a well-defined computational problem
- Is the solution to the problem *computable*?
- If it is, does there exist an *efficient* algorithm for it?
- How do we come up with an efficient algorithm?
Why do we Write Algorithms?

Given a real problem:

- Model it as a well-defined computational problem
- Is the solution to the problem *computable*?
- If it is, does there exist an *efficient* algorithm for it?
- How do we come up with an efficient algorithm?
How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
- Can we improve the algorithm? On what do we improve?
How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
- Can we improve the algorithm? On what do we improve?
  - Efficiency?
How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
- Can we improve the algorithm? On what do we improve?
  - Efficiency?
    - Less space? Storage Concerns?
    - Shorter running time? Faster is always better.
How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
- Can we improve the algorithm? On what do we improve?
  - Efficiency?
    - Less space? Storage Concerns?
    - Shorter running time? Faster is always better.
  - Can we make the algorithm simpler?
How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
- Can we improve the algorithm? On what do we improve?
  - Efficiency?
    - Less space? Storage Concerns?
    - Shorter running time? Faster is always better.
  - Can we make the algorithm simpler?
    - Use a simpler data structure?
    - Allow developers to extend and generalize?
    - Aesthetics concerns?
How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
- Can we improve the algorithm? On what do we improve?
  - Efficiency?
    - Less space? Storage Concerns?
    - Shorter running time? Faster is always better.
  - Can we make the algorithm simpler?
    - Use a simpler data structure?
    - Allow developers to extend and generalize?
    - Aesthetics concerns?
Paradigms we Will See in this Class

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Space and time tradeoffs
- Dynamic Programming
- Greedy Approach
- Iterative improvement
- Backtracking
- Branch and bound