Welcome to Analysis of Algorithms (CS583 - 004)

Amarda Shehu

Spring 2019

Class Information

Tentative Syllabus

Information

Instructor: Amarda Shehu **amarda\AT\gmu.edu**
Place and Time: Innovation Hall 203, F 1:30-4:10 pm
Instructor Office Hours: ENGR #4452, F 11:30 am - 1:30 pm
TA Office Hours: Yang Yong (yyang29\AT\gmu.edu), ENGR #4456, W 4:00 - 6:00pm

Tentative Syllabus

Date	Topic	Chapters	Assignments
Jan25	Course Overview, Insertion Sort, Merge Sort	C1-3, [pdf], [pdf]	Self-eval. Quiz
Feb01	Asymptotic Notations, Bounding Functions	C3-5, [pdf]	Quiz
Sorting and Order Statistics			
Feb08	More on Bounding Functions, Bounding Recurrences	C3-5, [pdf]	Quiz, Hw1 Out
Feb15	More on Bounding Recurrences	C3-5, [pdf]	Quiz
Feb22	Average Time Analysis and Quicksort	C6-8, [pdf]	Quiz
Mar01	Heapsort, Comparison- vs. Linear-time Sorting	C6-8, [pdf]	Quiz, Hw1 Due
Mar08	Order Statistics	C9, [pdf]	Quiz
Mar22	**Exam 1**		Exam 1
Data Structures for Storing and Searching			
Mar29	Hashing	C11 [pdf]	Quiz
Apr05	Advanced Data Structures and Analysis	C12, C13 [pdf]	Quiz
Optimization and Graph Algorithms			
Apr12	Dynamic Programming, Greedy Algorithms	C15-16, [pdf]	Hw2 Out, Quiz
Apr19	Graph Representation, Uninformed Search, Applications	C22, [pdf], [pdf]	Quiz
Apr26	Informed Graph Search	C24-25, [pdf], [pdf]	Quiz
May03	MST, Maximum Flow	C23, C26 [pdf], [pdf]	Quiz, Hw2 Due
May10	**Exam 2**	Innovation Hall 203	1:30 pm -- 4:15 pm

Instructor: Amarda Shehu

Office: ENG #4452
Email: amarda AT gmu.edu
Web: cs.gmu.edu/∼ashehu

CS583 Hours

Class: F 1:30 - 4:10 pm
Place: Innovation Hall 203
Office Hours: F 11:30 am - 1:30 pm

TA:

Email: yyang29 AT gmu.edu
ENG#5321, W 4:00 - 6:00 pm

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 12^{th} 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 1200^{th} birthday. ©wikipedia.

Seach Engines: Google, Yahoo, Ask

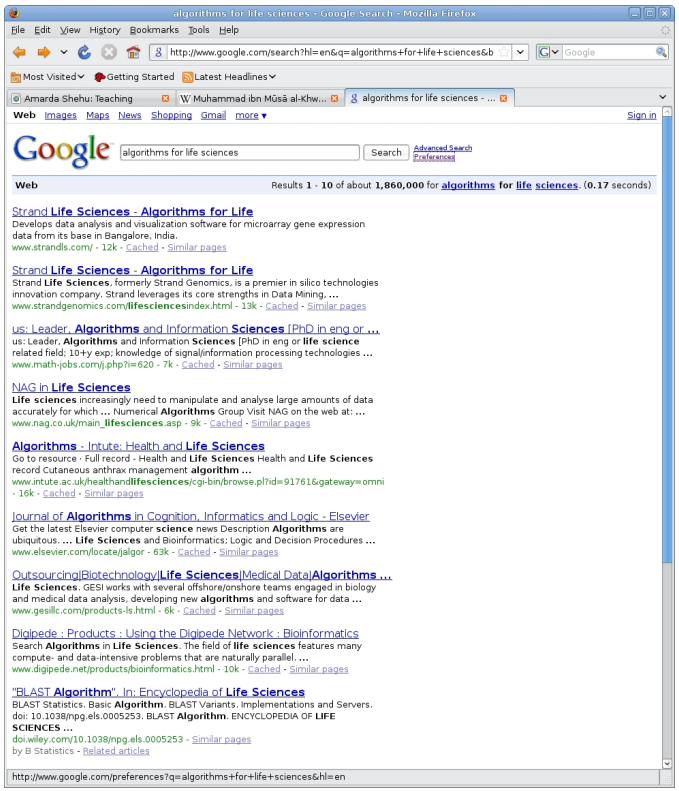

Figure: Searched for *algorithms for life sciences* in text in the web.

- 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*.

Orientation Software: Google maps, GPS navigators

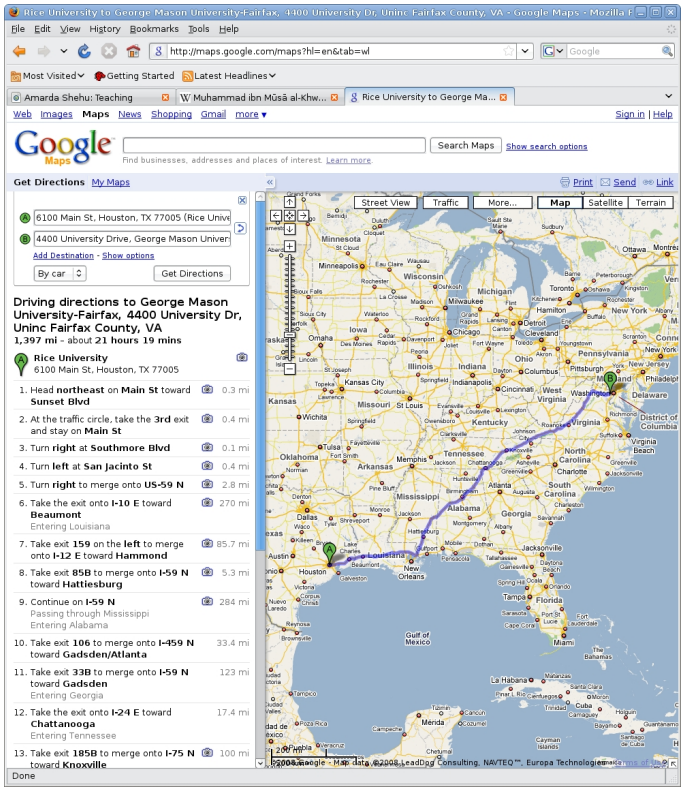

Figure: Output of a path finding algorithm.

- 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)

Exploration, Search and Rescue, and Motion Planning

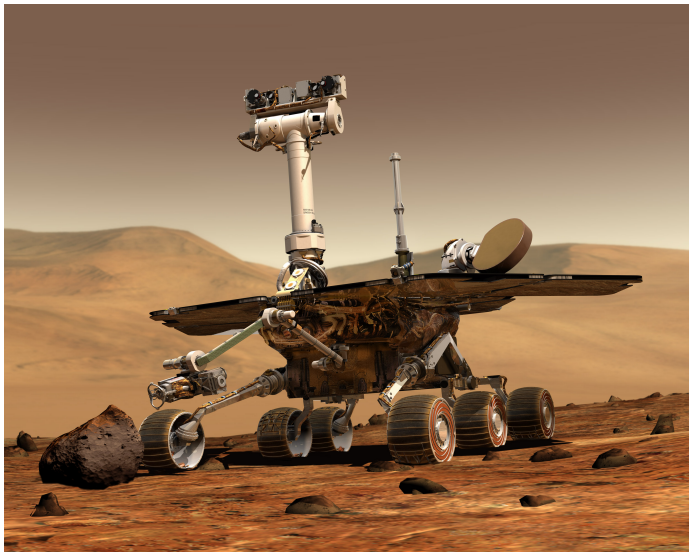

Figure: Ron Li and his research team are developing algorithms to help the rovers, Spirit and Opportunity, to navigate and find a safe path to a winter resting area. ©NASA.

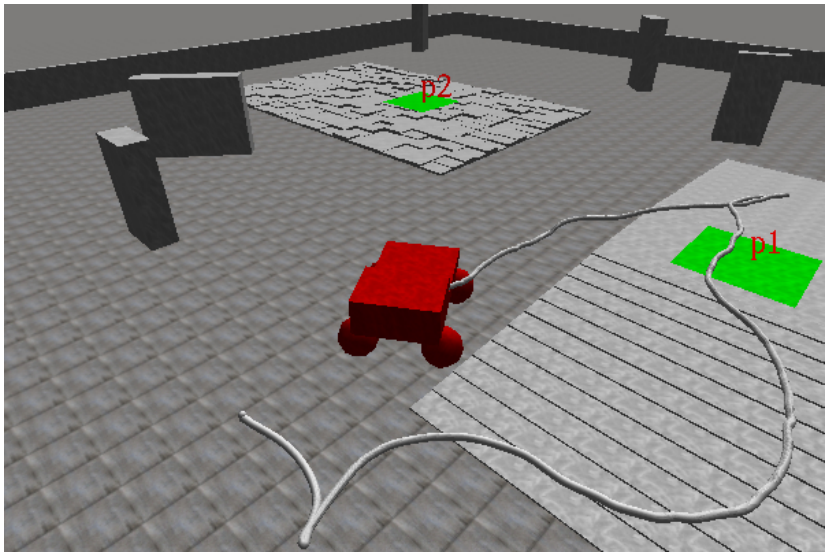

Figure: Erion Plaku at Rice University is developing algorithms that plan paths for car-like robots in cluttered environments. ©E. Plaku.

Simulating Molecular Properties for Drug Design

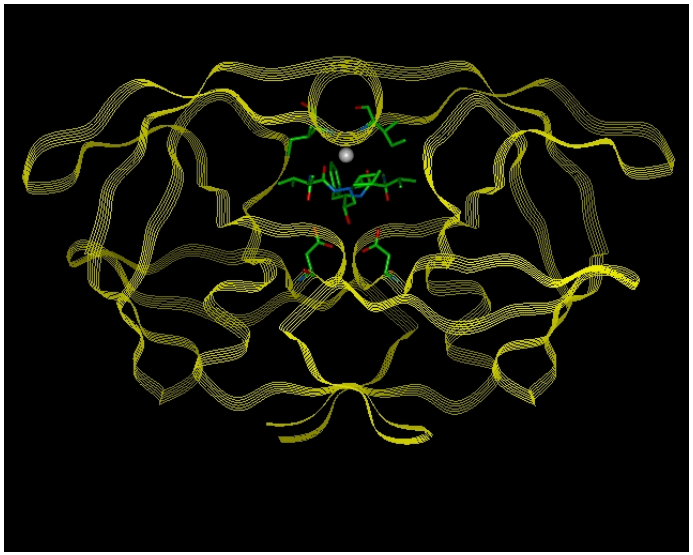

Figure: Successful docking of HIV protease with a small inhibitor ligand. ©A. R. Leach.

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

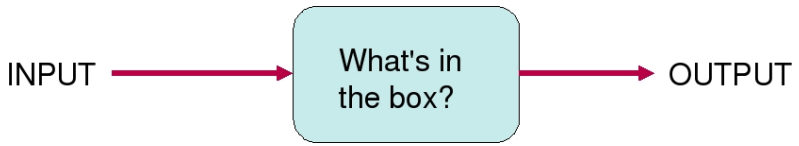

Figure:

- 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

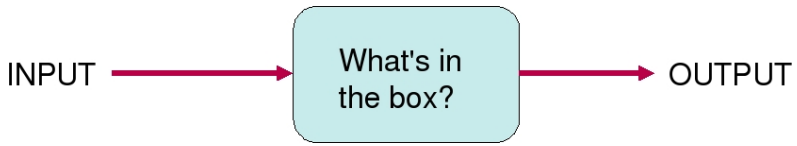

Figure:

- 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 ($\geq$ outputs)

What is an Algorithm?

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

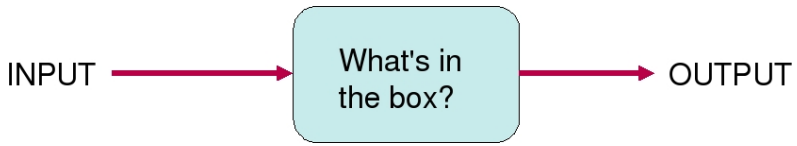

Figure:

- An algorithm is a finite sequence of unambiguous instructions for completing a task, that:
  - when given an initial state (≥ 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

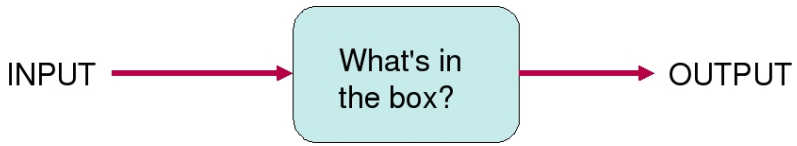

Figure:

- An algorithm is a finite sequence of unambiguous instructions for completing a task, that:
  - when given an initial state (≥ 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*?

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?

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