Algorithmic Thinking: Why and What

Algorithmic Thinking

Luay Nakhleh

Department of Computer Science

Rice University

"A consensus is emerging that, this time around, throwing more differential equations at the problem won't cut it. **Mathematics shine in domains replete with symmetry, regularity, periodicity**—things often missing in the life and social sciences. Contrast a crystal structure (grist for algebra's mill) with the World Wide Web (cannon fodder for algorithms). **No math formula will ever model whole biological organisms, economies, ecologies, or large, live networks.** Will the algorithm come to the rescue? This is the next great hope. The algorithmic lens on science is full of promise—and pitfalls."

"Algorithmic thinking is likely to cause the most disruptive paradigm shift in the sciences since quantum mechanics."

—Bernard Chazelle

"A person well-trained in computer science knows how to **deal with algorithms**: how to **construct** them, **manipulate** them, **understand** them, **analyze** them. This knowledge is preparation for much more than writing good computer programs: it is a **general-purpose mental tool** that will be a definite aid to the understanding of other subjects, whether they be chemistry, linguistics, or music, etc. The reason for this may be understood in the following way: It has often been said that a person does not really understand something until after teaching it to someone else. **Actually, a person does not really understand something until after teaching it to a computer, i.e., expressing it as an algorithm...** An attempt to formalize things as algorithms leads to a much deeper understanding than if we simply try to comprehend things in the traditional way."

—Donald Knuth

Algorithmic Thinking: The Myth

Understand the problem

↓

Formulate it

↓

Design an algorithm

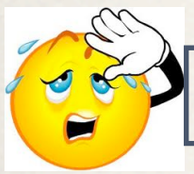
↓

Implement it

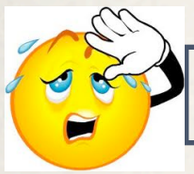
↓

Run the code and solve the original problem

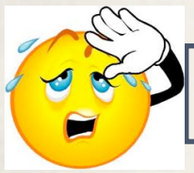
Algorithmic Thinking: The Reality



Understand the problem

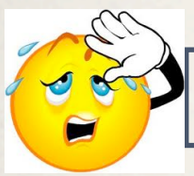


Formulate it



Design an algorithm

sometimes



Implement it

Run the code and solve the original problem

(1) Understanding the Problem

- Understand the description of the problem.
- What are the input/output?
- Do a few examples by hand.
- Think about special cases.

(2) Formulating the Problem

- Think about the data and the best way to represent them (graphs, strings, etc.)
- What mathematical criterion corresponds to the desired output?

(3) Designing an Algorithm

- Is the formulated problem amenable to a certain algorithm design technique (greedy, divide-and-conquer, etc.)?
- What data structures make sense to use?
- Is the algorithm correct?
- Is the algorithm efficient?
- If the problem is "too hard," do we want an approximation algorithm, a randomized algorithm, a heuristic, ...?

Algorithm Correctness

- The following message cannot be overemphasized: **Although tracing the algorithm's run on a few specific inputs can be a very worthwhile activity for understanding how the algorithm works, it cannot prove the algorithm's correctness conclusively.**

- To show that an algorithm is incorrect, you need just one instance of input for which the algorithm fails.

- To show that an algorithm is correct, we need to provide a conclusive mathematical proof of that (commonly using mathematical induction).

- Proof of correctness can be quite complex. We will discuss proofs and proof techniques in this course.

Algorithm Efficiency

- After correctness, the algorithm's efficiency is the most important quality.

- Time efficiency: how fast the algorithm runs

- Space efficiency: how much extra memory the algorithm needs

- There are theoretical tools to analyze an algorithm's efficiency. We will cover some of them in this course.

- Further, the actual running time of a specific implementation of the algorithm can be measured on a specific machine.

(4) Implementing the Algorithm

- Most algorithms are destined to be ultimately implemented as computer programs.

- Programming an algorithm presents both a peril and an opportunity.

- The peril lies in the possibility of making the transition from an algorithm to a program either incorrectly or very inefficiently.

- Special mathematical techniques have been developed for proving program correctness, but the power of these techniques is still limited to very small programs.

- As a practical matter, the validity of programs is still established by testing.

(4) Implementing the Algorithm

- Program correctness is necessary but not sufficient: You would not want to diminish your algorithm's power by an inefficient implementation.

- The opportunity lies in the possibility to better understand the algorithm and even to improve it.

(5) Solving the Original Problem

- Once the algorithm is implemented, it is run on data to solve the original problem.

- Sometimes, it is discovered after this entire process is followed, that the solution is not satisfactory. This might require going back to step (1) and repeating the entire process.