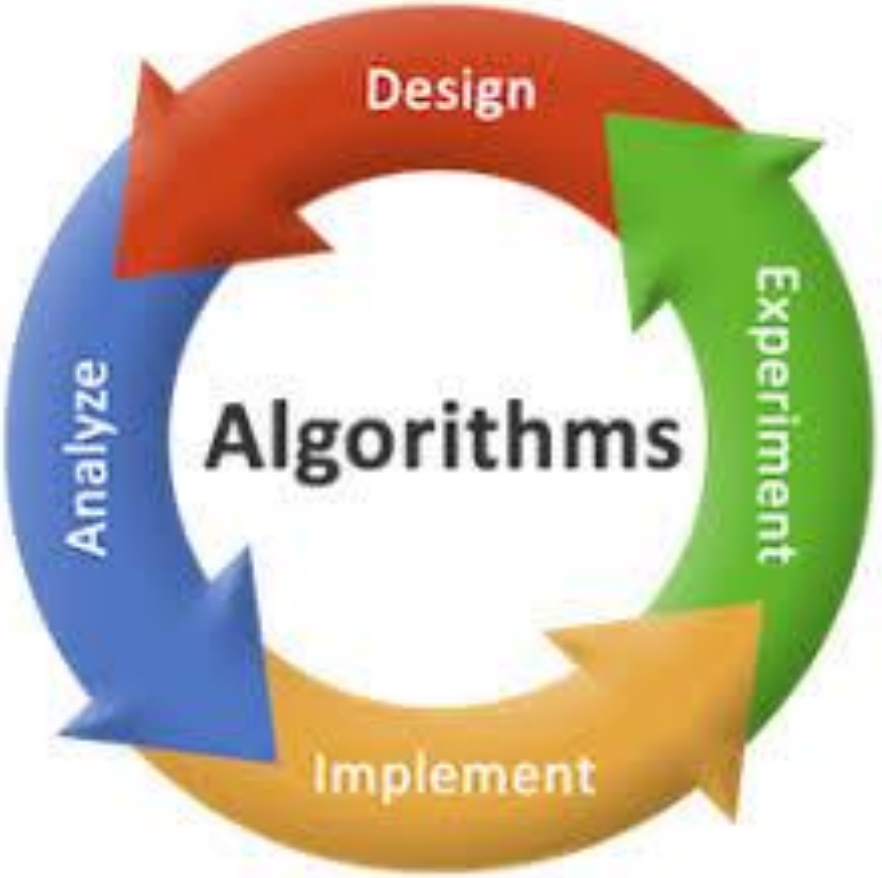
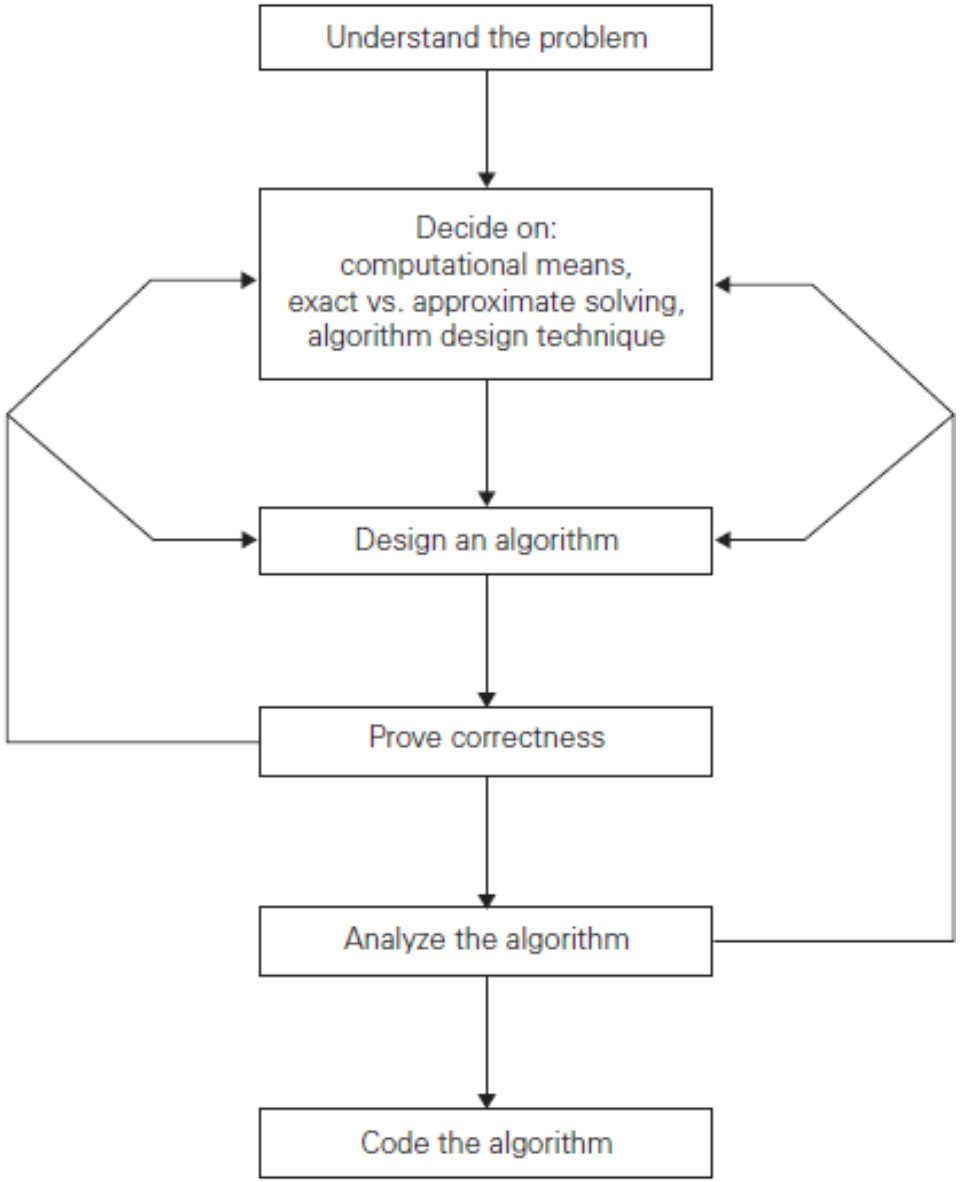


Design & Analysis of Algorithms

Lecture 3

Fundamentals of Algorithmic Problem Solving

Fundamentals of Algorithmic Problem Solving



Understanding the Problem

- **Problem Description**
  - few small examples by hand
  - think about special cases
  - ask questions again if needed

- **Inputs**
  - Your algorithm may work correctly for a majority of inputs but crash on some "**boundary**" value.
  - Remember that a **correct** algorithm is not one that works most of the time, but one that works correctly for all **legitimate inputs**.

Ascertaining the Capabilities of the Computational Device

- The vast majority of algorithms in use today are still destined to be programmed for a computer closely resembling the **Von Neumann** machine
  - **Random-Access Machine (RAM)**
  - Instructions are executed one after another, one operation at a time.
  - Accordingly, algorithms designed to be executed on such machines are called **Sequential Algorithms**.

Ascertaining the Capabilities of the Computational Device

- The central assumption of the RAM model does not hold for some newer computers that can execute operations concurrently, i.e., in **parallel**.

- Algorithms that take advantage of this capability are called **Parallel Algorithms**.

Ascertaining the Capabilities of the Computational Device

- Should you worry about the speed and amount of memory of a computer at your disposal?

- **NO**

- **Algorithms should be System Independent**

Choosing between Exact and Approximate Problem Solving

- **Exact Algorithms**
- **Approximate Algorithms**

WHY Approximate Algorithms?

- there are important problems that simply cannot be solved exactly for most of their instances;
  - **Extracting square roots, solving nonlinear equations, and evaluating definite integrals.**
- Available algorithms for solving a problem exactly can be unacceptably slow because of the problem's intrinsic complexity.

Algorithm Design Techniques

- An algorithm design technique (or "**strategy**" or "**paradigm**") is a general approach to solving problems **algorithmically** that is applicable to a variety of problems from different areas of computing.

- Algorithm design techniques make it possible to **classify** algorithms according to an underlying design idea

Designing an Algorithm and Data Structures

- Choosing data structures appropriate for the operations performed by the algorithm

Example:

- The **sieve of Eratosthenes** would run longer if we used a linked list instead of an array in its implementation (**Why?**).

the Sieve of Eratosthenes relies heavily on efficient memory access patterns and rapid element updates, both of which are better supported by arrays than by linked lists.

- **Google the Answer**

Methods of Specifying an Algorithm

- Two options that are most widely used nowadays for specifying algorithms.
  1. **Free** and also a **step-by-step** form (**flowchart**)
  2. **Pseudocode**: is a mixture of a natural language and programming language like constructs

Proving an Algorithm's Correctness

- You have to prove that the algorithm yields a required result for every legitimate input in a finite amount of time.

- **For example: the correctness of Euclid's algorithm**
  - for computing the greatest common divisor stems from the correctness of the equality **gcd(m, n)** = **gcd(n, m mod n)** the simple observation that the second integer gets smaller on every iteration of the algorithm, and the fact that the algorithm stops when the second integer becomes 0.

Analyzing an Algorithm

- After correctness, by far the most important is **efficiency**.

- Two kinds of algorithm efficiency:

1. **time efficiency**— indicating how fast the algorithm runs
2. **space efficiency**— indicating how much extra memory it uses.

- Other desirable characteristic of an algorithm are **simplicity**, **generality**.

Coding an Algorithm

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

- Of course, implementing an algorithm correctly is **necessary** but **not sufficient**:
- you would not like to diminish your algorithm's power by an inefficient implementation.

(Modern compilers do provide a certain safety net in this regard, especially when they are used in their code optimization mode.)

Algorithms and other technologies

- Algorithms, like computer hardware, are a technology.
- Total system performance depends on choosing efficient algorithms as much as on choosing fast hardware.
- Whether algorithms are truly that important on contemporary computers in light of other advanced (following) technologies?
  - hardware with high clock rates, pipelining, and superscalar architectures
  - easy-to-use and intuitive GUIs
  - object-oriented systems
  - local-area and wide-area networking
- **YES**

References

- **Chapter 1**: Anany Levitin, "Introduction to the Design and Analysis of Algorithms", Pearson Education, Third Edition, 2017

- **Chapter 2**: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "Introduction to Algorithms", MIT Press/PHI Learning Private Limited, Third Edition, 2012.

Homework

- Design an algorithm for computing $\lfloor \sqrt{n} \rfloor$ for any positive integer n. Besides assignment and comparison, your algorithm may only use the four basic arithmetical operations.

- Consider the problem of adding two n-bit binary integers, stored in two n-element arrays A and B. The sum of the two integers should be stored in binary form in an (n + 1)-element array C. State the problem formally and write pseudocode for adding the two integers.