COT 6405
ANLYSIS OF ALGORITHMS

Foundations – Analyzing Algorithms

Computer & Electrical Engineering and Computer Science Department
Florida Atlantic University

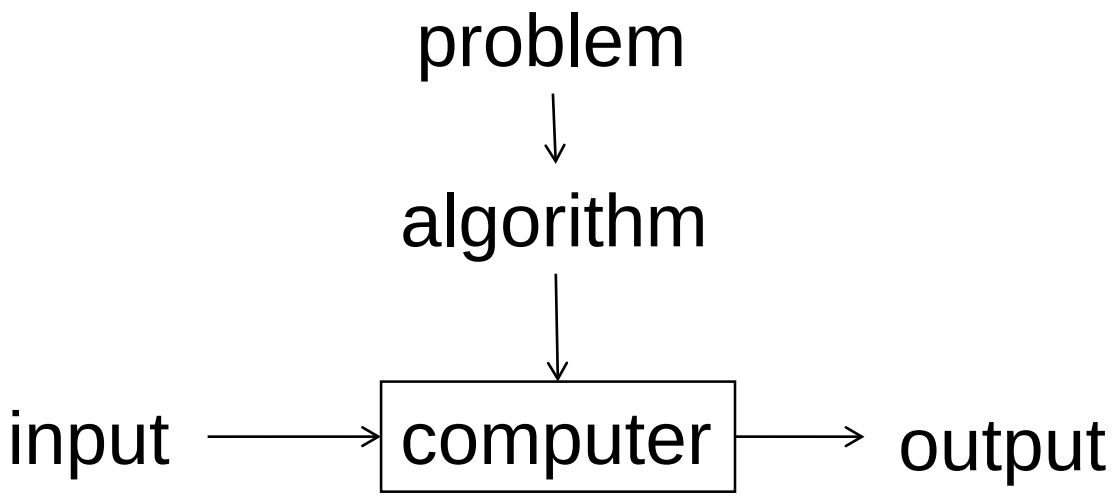
Outline

- What is an algorithm?
- Analyzing algorithms
- Which algorithm is more *efficient*?
- High-level problem classification

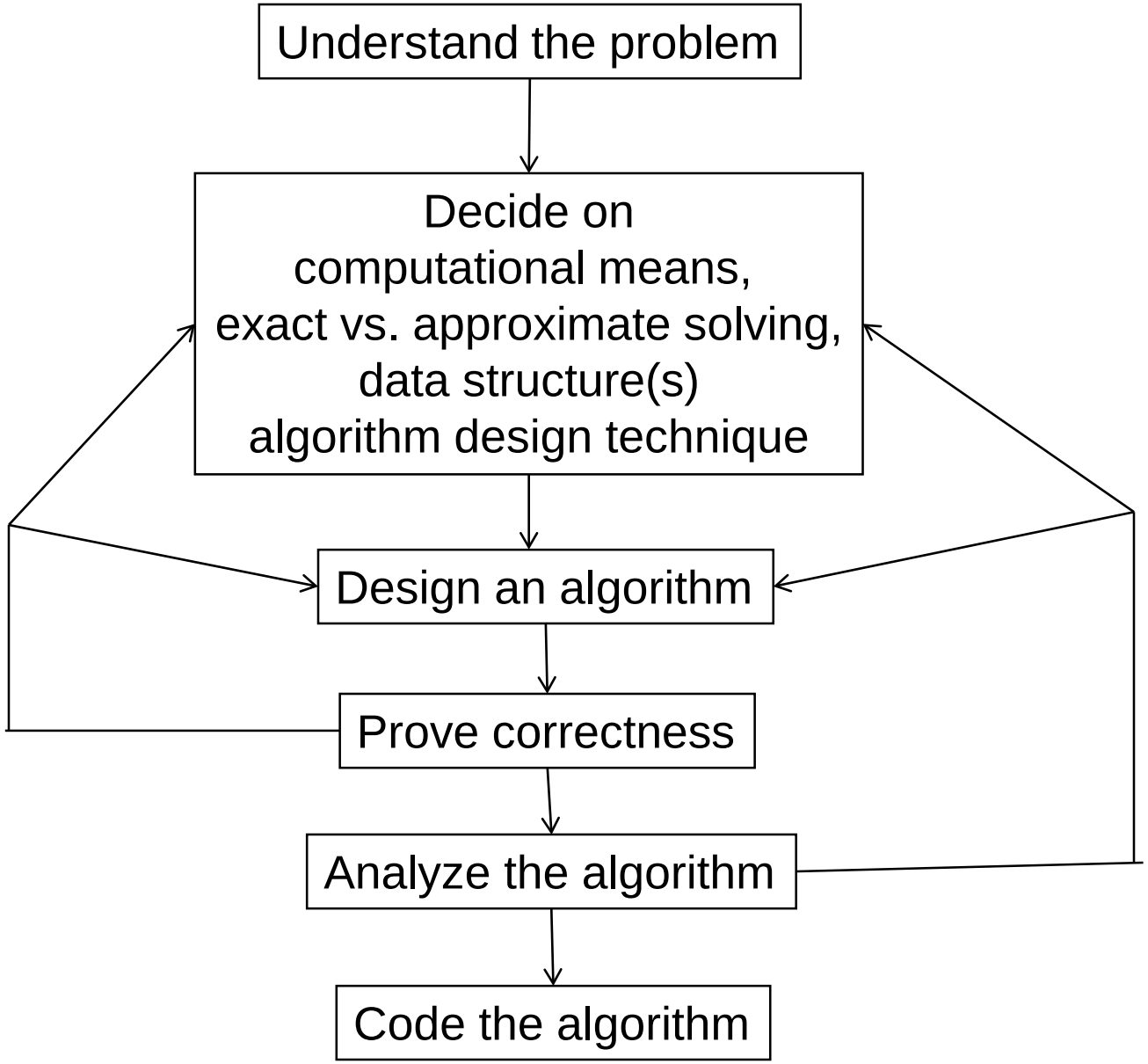
Reference: *Introduction to Algorithms*, 3rd edition, by T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, The MIT Press, 2009 (chapter 2.2)

What is an Algorithm?

Well-defined computational procedure that takes some value or set of values as input and produces some value or set of values as output.



Fundamentals of Algorithmic Problem Solving



Analyzing Algorithms

- Use pseudocode to describe algorithms
- Analyzing algorithms
  - Want to predict resources that the algorithm requires
    - Running time (or computational time)
    - Memory
    - Bandwidth
    - Hardware components
    - so on.

Random Access Machine (RAM) Model

Assumptions:

- Instructions executed one after another (e.g. sequential algorithms)
- Primitive instructions take a constant amount of time
  - *Arithmetic*: add, subtract, multiply, divide, remainder, floor, ceiling, shift left, shift right
  - *Data movement*: load, store, copy
  - *Control*: conditional/unconditional branch, subroutine call and return
- Uses integer and floating-point types

Computing the running time

- Express running time using *asymptotic notations* as a function of the *input size*
- Input size depends on the problem being studied
- Running time on a particular input is the number of primitive operations (steps) executed

  - Examples

When is an algorithm considered “efficient”?

- Platform-independent, instance-independent, and of predictive value with respect to increasing input sizes
- Think about the worst-case RT
- An algorithm is *efficient* if:
  - When implemented, it runs quickly on real input instances
  - Achieves qualitatively better worst-case performance, at analytical level, than brute force
- An algorithm is *efficient* if the worst-case running time is polynomial

Tractable vs intractable problems

- Problems that have worst-case polynomial-time algorithms are called **feasible** or **tractable**
- A problem that does not have a worst case polynomial time algorithm is called **intractable**
- A problem for which there is no algorithm is said to be **unsolvable**
  - Halting problem: given a Turing machine M and an input, will M eventually halt?
- **NP-complete problems**
  - solvable problems that have an undermined status: they are thought to be intractable, but none of them has been proved to be intractable
  - if one NP-complete problem has a polynomial-time algorithm, *all* of them will have polynomial-time algorithms
  - no polynomial-time algorithm discovered so far $\Rightarrow$ believed they are intractable