Algorithms 2024: Introduction

(Based on [Manber 1989])

Yih-Kuen Tsay

September 3, 2024

1 About Algorithms

What They Are

- An **algorithm** is, broadly speaking, a *step-by-step* procedure for solving a problem or accomplishing some end.
- When it is meant for the (electronic) computer, each step in an algorithm should be realizable by *well-defined*, limited *primitive* operations that the computer understands.

  /* In this course, an algorithm is often described in an abstract high-level way, and yet each step in the description can be realized by one or more statements of some real programming language, which may then be translated into instructions for the computer to execute. */
- You actually have learned several algorithms during your school years. Can you name one?

  /* Euclid's algorithm and Gaussian elimination are probably the most notable. */
- Algorithm design is an important and usually the hardest part of programming (which consists in finding/devising a solution and translating it into a computer program).
- Better algorithms (designed once, used forever) save more time and money.

Development of an Algorithm

- We typically are given a problem statement, including input and output requirements, that is an abstract yet *accurate* and *precise* account of the problem to be solved and the properties of a satisfactory solution.

  /* Take the sorting problem as an example. You are given a sequence of numbers, which is the input, and asked to arrange the numbers in ascending order, which is the output. If instead of numbers pairs of numbers or strings are considered, it is not clear what "ascending order" means; that needs to be clarified and made precise. You may be further informed that the input sequence of numbers is represented as an array and the output should be stored in the same array. */
- The development of an algorithm involves the following tasks:
  1. Design (main subject of this course)
  2. Verification (or Proof of Correctness)

     /* The methods of verification include testing, formal verification, etc. */

3. Analysis

   /* The analysis mainly is to determine how fast the algorithm runs and how much storage is needed to run the algorithm, relative to the size/scale of the input. */

4. Implementation (in some programming language)

(May need to iterate.)

Main Concerns

- Why is algorithm design difficult?
  - Computers are different from humans; they are very fast and can handle much larger amounts of data.
  - Counterintuitive approaches may be needed, because of large problem scales.

    /* Intuitive algorithms that work well for small problem instances may be terrible for large problem instances. */
  - Better solutions, if worthwhile (with greater payoffs), may be more complicated.
- How do we approach it?

2 Our Emphasis

A Creative Approach to the Subject

- Emphasis of the creative side
  - not only memorizing solutions
  - but also learning to create by trying to create
- Induction as one central design method
  - to explain/understand the principles behind a design
  - to systematically guide the creation process

Design by Induction

- Design by induction draws analogies from proving theorems by *mathematical induction*.
- In a proof by induction, we do not prove a statement from scratch, but rather we show
  1. the correctness of the statement follows from that of the same statement for smaller instances and
  2. the correctness of the statement for a small base case.
- This suggests an approach to algorithm design that concentrates on *extending* solutions for smaller problem instances to solutions for larger ones.
- Induction may not solve every problem, but is very helpful.

  /* Some types of problems require essentially trying all possibilities, e.g., the 2^n possible truth assignments to n Boolean variables. One may still enumerate all the possibilities in terms of induction, but that does not really help get a more efficient solution.*/