CS417 Module 1 Part A - Introduction

Athens State University

January 3, 2017

Outline

Contents

1 Introduction

What is an algorithm?

Algorithm A sequence of unambigous instructions for solving a problem; for example, for obtaining a required output for any legitimate input in a finite amount of time

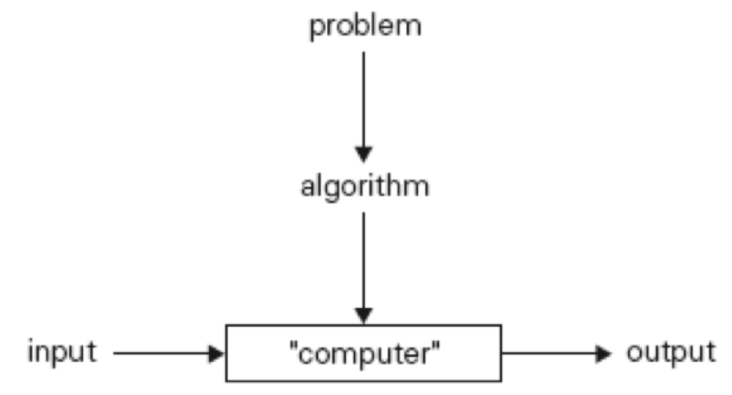


FIGURE 1.1 The notion of the algorithm.

Why study algorithms?

- Theoretical importance: core of computer science
- Practical importance
  - Building a toolkit of algorithms for known problems
  - Understanding how to design algorithms for new problems
  - Analyzing and comparing different algorithms for a problem

Two main issues

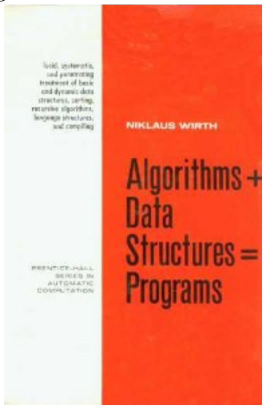
- How to design algorithms?
- How to analyze efficiency of algorithms?

What an algorithm is not

```
void insertion_sort(item s[], int n)
{
 int i,j;
 for (i=1; i<n; i++) {
  j=i;
  while ((j>0) && (s[j] < s[j-1])) {
    swap(&s[j],&s[j-1]);
    j = j-1;
  }
}
```

What an algorithm is not

[Wirth 1976]: Algorithms + Data Structures = Programs


NIKLAUS WIRTH
Algorithms + Data Structures = Programs
PRENTICE-HALL SERIES IN AUTOMATIC COMPUTATION

- Three desirable properties
  - Correct
  - Efficient
  - Easy to implement

What language do we use for an algorithm?

Use whatever is most clear and concise.
We all understand English (Spanish, Italian, Mandarin, . . .)
There are times when we must use flowcharts and *pseudocode*

English: Insertion sort

Sometimes the best language is the one we understand: *Insertion sort: the card game*

1. Start with an empty left hand and the cards face down.
2. Remove one card at a time from the table and put them in your left hand
3. Compare the card with each card in your left hand, from right to left. Put card in correct position.
4. At all times, the cards held in the left hand are sorted.

Pseudocode: Insertion sort

Algorithm 1: Insertion Sort

```
Input: A set s of items of length $n$
Output: The set s in sorted order
for $i \leftarrow 1 \ldots n$ do
    j ← i;
    while $j > 0$ and $s(j) < s(j-1)$ do
        swap s(j) and s(j-1);
        j ← j-1;
```

Pseudocode: Key points

- Similar to C, C++, Java.
- Human, not compilers
- Need to be careful with algorithms that use arrays
  - Do array indexes start with 0 or 1?
- Use similar control structures as with C/C++/Java

OK, why use this over natural language

- Why do we need this intermediate representation between natural language and code?

Pseudocode is an informal *high-level* description of an algorithm. Key is the point that it is an *intermediate* representation. So, we can, if required, go into greater detail about aspects of an algorithm than when one uses natural language to describe an algorithm.

As a design tool, pseudocode is an alternative to tools like flowcharts and UML. One should use the tool that makes the most sense for your situation.

One last note about notation

The heart of any algorithm is an *idea*. If your idea is not clearly revealed when you express an algorithm, then you are using too low-level a notation to describe it.

1.1 Algorithm Design Techniques

Algorithm Design Techniques

- Brute force
- Divide and conquer
- Transform and conquer
- Space and time tradeoffs
- Greedy approaches
- Dynamic programming
- Iterative improvement
- Backtracking
- Branch and bound

1.2 Important problem types

Important problem types

- Sorting
- Searching
- String processing
- Graph problems
- Combinatorial probblems
- Geometric problems
- Numerical problems

2 Analysis of Algorithms

What should you ask about an algorithm?

- Is the algorithm correct?
- How good is the algorithm?
  - time efficiency
  - space efficiency
- Does there exist a better algorithm?
  - Lower bounds
  - Optimality

Correctness

Correctness : an algorithm yields a required result for every legitimate input in a finite amount of time

Note: this is *proof* as in what you learned in Discrete Math

Easy for some algorithms, almost impossible for others

Many algorithms are searching for approximate rather exact answers

Correctness : Loop Invariant

Loop invariant criteria that must be true on each iteration of a loop

Using invariants to prove correctness

Initialization the invariant is true prior to the first iteration of the loop

Maintenance If the invariant is true before an iteration of the loop, it remains true before the next iteration

Termination When the loop terminates, the invariant plus the loop termination criteria provides the means to show the algorithm is correct

The concept of *invariant* has been extended in software engineering to the theory of *design by contract*. The idea is that components of a software system collaborate with each other on the basis of mutual obligations and benefits. These are expressed in terms of preconditions, postconditions, and invariants that must be asserted whenever state changes within a system.

3 Key Points

Key Points

- What is an algorithm?
- Why study algorithms?
- How do we express an algorithm?