Algorithms and Data Structures

Catherine Durso

cdurso@cs.du.edu

Algorithms

An algorithm is a well-defined procedure to produce output with a specified realtion to input.
Sorting is a typical example. There are many sorting algorithms with different strengths and weaknesses.

- Input is a sequence of numbers $< a_1, a_2, ...a_n >$
- Output is a permutation of the input $< a'_1, a'_2, ...a'_n >$ with $a'_i \leq a'_j$ whenever $i \leq j$.

Algorithm Terms

An *instance* of a problem is a specific input.
An algorithm solves a problem if, for any valid input, it eventually halts with the correct output. If this is the case, the algorithm is said to be *correct*.

Algorithms as Technology

There are many fields in which progress depends on the design of correct algorithms that run in a practical amount of time and require a practical amount of memory.

- Genomics, proteomics, bioinformatics
  - assembling DNA sequences from short sequences of nucleotides
  - determining possible geometries of proteins
  - genome comparisons
- Internet traffic management
  - making traffic fast, secure, and robust
- Cryptography

cont.

- Games
  - rendering graphics
  - controlling non-player characters
- Operations Research
  - allocating resources to optimize results
- Data Mining and Statistical Analysis
- Modeling and Simulation
  - predicting climate, fire, disease, economic conditions

Recent Innovations

The "35 under 35" is a list of 35 innovators under 35 assembled by **Technology Review Magazine** each year to identify individuals whose work is "likely to be influential for a very long time." Of the 35, 11 of the honorees work in computing. Of those, 9 used novel algorithms or data structures in their innovations.

The Innovators

- Ren Ng (Entrepreneur of the Year), age 32, developed a camera that takes pictures that can be digitally refocused after the image is captured.
- Rana el Kaliouby, age 34, assembled the world's largest database of facial expressions, and is developing technologies that allow automated assessment of a subject's mental state.
- John Hering, age 29, is developing detection of smartphone malware, using crowd-sourced threat finding, and a database of rogue apps.
- Hossein Rahnama, age 32, is developing practical context aware computing for mobile applications.
- Saikat Guha, age 30, has designed a practical method to target web advertising without violating privacy, and is designing algorithms that detect privacy violations.

more...

- Daniel Ek, age 29, developed Spotify, a cloud based provider of legal on-demand access to 16 million songs.
- Andreas Veltin, age 32, developed hardware and software for a super-fast camera that can capture the progress of a light ray.
- Burcin Becerik-Gerber, age 35, developed algorithms to harmonize work environment preferences of a building's occupants to reduce energy consumption.
- Drew Houston, age 29, overcame speed, scale, and memory problems to develop Dropbox.

The Challenge

Challenge: Design a correct algorithm with acceptable, or optimal, performance. The main performance metrics are speed and size of memory required.
Example: Would you sort an array by generating all possible permutations, then testing each to find a sorted permutation?

challenge, cont.

That approach requires the generation of $n!$ permutations. By the way, $100! \sim 10^{158}$.

Data Structures

A data structure is a way to organize and store data to facilitate the accesses and modifications. The data structure chosen or designed for a particular application should perform on the types of access and modification required by he application. A data structure may be implemented in multiple ways.
Examples:

- list
- stack
- queue
- binary search tree
- priority queue

Sample Problems

- Suppose you are given a set of real numbers, A, and a spacing size $s > 0$. The goal is to group the numbers into subsets $A_1, A_2, ...A_n$with the property that, for each subset, all the numbers in the subset are at least s apart. That is, $\forall i \in \{1, 2, ...n\} \forall a \in A_i \forall b \in A_i \left(a \neq b \Rightarrow |a - b| \geq s\right)$. Further, n should be minimal. By the way, this problem may seem artificial, but it arose in a practical problem.
- Find the maximal sequence possible for a given set of dominoes.

Chapter 2

Chapter 2 provides an overview of the analysis of an algorithm correctness and the analysis of an algorithms's asymptotic running time. The INSERTION SORT and MERGE SORT algorithms for sorting arrays are used as examples.
These algorithms are presented using pseudocode. The text's version of pseudocode uses indentation in place of nested brackets and uses "//" to introduce comments. Other conventions are detailed on pages 20 and 21.

Insertion Sort

```
Insertion-Sort(A)
1  for j=2 to A.length
2    key=A[j]
3    // Insert A[j] into the sorted sequence
4    i=j-1
5    while i>0 and A[i]>key
6      A[i+1]=A[i]
7      i=i-1
8    A[i+1]=key
```

Correctness

When you specify an algorithm in pseudocode but do not implement it, how can you verify that the algorithm is correct? Correctness becomes more difficult to verify when the algorithm involves iteration.

In this case, one approach is to use loop invariants. A loop invariant is a statement about the state of the program, including the data, each time control passes to the loop statement.

Loop Invariant

- The loop invariant should be true when control first passes to the loop. (*Initialization*)
- If the loop invariant was true before an iteration of the loop block, and the loop condition is true, then the loop invariant is true after the iteration, at the evaluation of the loop condition. (*Maintenance*)
- The loop invariant should give useful information about the configuration of the program when control exits the loop. (*Termination*)

For- loop Invariant

For now, assume that if the while-loop is passed an array A with $A[1, ..j-1]$ in ascending order, then after line 8, the array A is a permutation of the the input array, and $A[1, ..j]$ is in ascending order.

The loop invariant for the for-loop is "A is a permutation of the input array, and $A[1, ..j-1]$ is in ascending order."

Invariant Verification

Initialization j=2: The input array has not been changed. The claim that $A[1, ..j-1]$ in ascending order becomes the claim that $A[1, ..2-1] = A[1]$ is in ascending order is trivially true.

cont.

Maintenance: We are assuming that the effect of the while-loop with line 8, when passed an array with $A[1, ..j-1]$ in ascending order, is to return a permutation of that array with $A[1, ..j]$ in ascending order. Thus if the loop invariant and the loop condition are true at line 1, then at line 8, A is a permutation of the input array, and $A[1, ..j]$ is in ascending order. Thus the loop invariant remains true after the incrementation of j in the for-loop.

cont.

Termination: The for-loop terminates with the loop invariant true and $j = A.length + 1$. Thus $A[1, ..A.length]$ is in ascending order and is a permutation of the input array, as required.

While-loop

Each iteration of the while-loop compares the key, the value in $A[j]$, to the value in $A[i]$, and copies value in $A[i]$ one position up if $A[i] > key$. The goal is to verify the assumption used in analysis of the for-loop. Thus the following is a reasonable loop invariant: A, with key inserted into $A[i+1]$ is a permutation of the input array for the loop. The values in $A[1,..i]$ followed by the values in $A[i+2,..j]$ are the values in the input array in positions $1,..j-1$, in the input order. Further, $A[k] > key$ for $k \in \{i+2,..j\}$.

Initialization

At this point, $i = j - 1$, and the input array has not been changed. Thus A with key inserted into $A[i + 1] = A[j]$ is a permutation of the input array for the loop. The subarray $A[i + 2, ..j]$ is empty, so the values in $A[1, ..i]$ followed by the values in $A[i + 2, ..j]$ are the values in the input array in positions $1, ..j - 1$, in the input order. The set $\{i + 2, ..j\}$ is empty, so the comparison $A[k] > key$ for $k \in \{i + 2, ..j\}$ is trivially true.

Maintenance

Given that the loop condition is true and the loop is entered, $A[i] > key$. Line 6 copies $A[i]$ into $A[i+1]$. Thus the values in $A[1, ..i-1]$ followed by the values in $A[i+1, ..j]$ are the values in the input array in positions $1, ..j-1$, in the input order. Also, $A[k] > key$ for $k \in \{i+1, ..j\}$. In line 7, i is decremented. For this new i, the values in $A[1, ..i]$ followed by the values in $A[i+2, ..j]$ are the values in the input array in positions $1, ..j-1$, in the input order. Also, $A[k] > key$ for $k \in \{i+2, ..j\}$, as required.

Termination

The loop terminates with the loop condition true and $A[i] \leq key$. Line 8 copies key into $A[i+1]$. The loop invariant implies that $A[1, ..j]$ contains the values in in the input array in positions $1, ..j$, A is a permutation of the input array, and the values in $A[i+2, ..j]$ are greater that key, the value in position $i+1$. Assuming that $A[1, ..j-1]$ was in ascending order when the while-loop was entered, and A contains a permutation of the values in the input array for the function, $A[1, ..j]$ is in ascending order after line 8 and A is still a permutation of the argument of the function.