01 A—Intro

CS1102S: Data Structures and Algorithms

Martin Henz

January 13, 2010

Generated on Wednesday 13th January, 2010, 09:50

1. Getting Started
   - Goals
   - Structure and Material of Module
2. Overview of CS1102S
3. Algorithm Analysis

Goal of CS1102S

In CS1102S, we will work on basic skills for software practice and theory:

- *Data structures* as building blocks of programs
- *Algorithms* as solutions to computational problems
- *Path* from program text to executing solution
- *Tools* for software design, development and maintenance
- *Theory* of computation; analysis of algorithms

Java

Students of CS1101S have already a solid foundation of basic data abstraction and functional (algorithmic) abstraction.

Java

Students of CS1101S have already a solid foundation of basic data abstraction and functional (algorithmic) abstraction. CS1102S thus focuses on:

Java

Students of CS1101S have already a solid foundation of basic data abstraction and functional (algorithmic) abstraction. CS1102S thus focuses on:

- Specialized data structures as solutions to common computational problems

Java

Students of CS1101S have already a solid foundation of basic data abstraction and functional (algorithmic) abstraction. CS1102S thus focuses on:

- Specialized data structures as solutions to common computational problems
- Competency in Java (also for other SoC modules)

Java

Students of CS1101S have already a solid foundation of basic data abstraction and functional (algorithmic) abstraction. CS1102S thus focuses on:

- Specialized data structures as solutions to common computational problems
- Competency in Java (also for other SoC modules)
- Required background (paths, tools, theory) for software professionals

Structure of CS1102S

Wednesday lectures: 2 h; Data structures, algorithms, paths

Friday lectures: 1 h; Tools, theory, and other things

Tutorials: Discussing weekly assignments

Labs: Assisted sessions to practice software skills

Java crash course: next slide

Java Crash Course

Goals: Java basics, programming tools

Java Crash Course

Goals: Java basics, programming tools item[Structure:] 30 min (+) lecture, followed by ca 2 hours of lab time

Java Crash Course

Goals: Java basics, programming tools item[Structure:] 30 min (+) lecture, followed by ca 2 hours of lab time

Venue of lectures: First lecture: COM1/202

Java Crash Course

Goals: Java basics, programming tools item[Structure:] 30 min (+) lecture, followed by ca 2 hours of lab time

Venue of lectures: First lecture: COM1/202

Venue of lab time: PL1/PL2

Material for CS1102S

- IVLE: Discussion forum

Material for CS1102S

- IVLE: Discussion forum
- Textbook: Weiss: Data Structures and Algorithm Analysis in Java, 2nd Edition
  Available at COOP (under Central Library)

1. Getting Started
2. Overview of CS1102S
3. Algorithm Analysis

Overview of CS1102S

- Algorithm analysis
- Lists, Stacks, Queues
- Trees
- Hashing
- Priority Queues
- Sorting
- Graph Algorithms

Algorithm Analysis

Runtime analysis

Characterize runtime of algorithms, not programs

Algorithm Analysis

Runtime analysis

Characterize runtime of algorithms, not programs

Abstraction

Remove peculiarities of particular programming languages and computers

Lists, Stacks, Queues

Collections

Collections are data structures that contain a number of data items of a uniform type.

Lists, Stacks, Queues

Collections

Collections are data structures that contain a number of data items of a uniform type.

Access order

Lists, stacks and queues differ in the order in which the items are entered, accessed and removed.

Trees

Trees as data structures

Trees represent hierarchical information.

Trees

Trees as data structures

Trees represent hierarchical information.

A particular use of trees

Search trees provide easy access to a sorted collection of items.

Hashing

Problem

Keep track of large number of items, so that we can find them fast.

Hashing

Problem

Keep track of large number of items, so that we can find them fast.

Idea

Compute a *key*, that is used for entry, access and removal.

Priority Queues

Problem

Provide fast access to the smallest item in a collection.

Priority Queues

Problem

Provide fast access to the smallest item in a collection.

Idea

Keep the items in a tree, where you guarantee that the smallest item is at the top.

Sorting

Problem

Sort a given number of items in increasing order.

Sorting

Problem

Sort a given number of items in increasing order.

Solutions

Insertion sort, Shellsort, Heapsort, Mergesort, Quicksort

Graph Algorithms

Problem

Represent data items that are connected in interesting ways.

Graph Algorithms

Problem

Represent data items that are connected in interesting ways.

Applications

Shortest path, network flow, minimum spanning tree, depth first search

Motivation

Which functions grows faster?

$f(x) = 1000x$, or $g(x) = x^2$

Motivation

Which functions grows faster?

$f(x) = 1000x$, or $g(x) = x^2$

Intuition

g grows faster than f because *eventually* it will return larger values.

Motivation

Which functions grows faster?

$f(x) = 1000x$, or $g(x) = x^2$

Intuition

g grows faster than f because *eventually* it will return larger values.

No worries about constants

We would like to “overlook” when functions differ only by a constant factor.

Motivation

Which functions grows faster?

$f(x) = 1000x$, or $g(x) = x^2$

Intuition

g grows faster than f because *eventually* it will return larger values.

No worries about constants

We would like to “overlook” when functions differ only by a constant factor.
Example: $f(x) = 1000x$ grows in the same way as $g(x) = 2000x$.

Big Oh!

Definition

$T(N) = O(f(N))$ if there are positive constants c and n_0 such that $T(N) \leq cf(N)$ when $N \geq n_0$.

Big Oh!

Definition

$T(N) = O(f(N))$ if there are positive constants c and n_0 such that $T(N) \leq cf(N)$ when $N \geq n_0$.

Example

$T(N) = 1000N$
$f(N) = N^2$
$T(N) = O(f(N))$

Big Oh!

Definition

$T(N) = O(f(N))$ if there are positive constants c and n_0 such that $T(N) \leq cf(N)$ when $N \geq n_0$.

Example

$T(N) = 1000N$
$f(N) = N^2$
$T(N) = O(f(N))$

Notation

We often simply use the function definitions as in:

$$1000N = O(N^2)$$

Some more definitions

Big Oh

$T(N) = O(f(N))$ if there are positive constants c and n_0 such that $T(N) \leq cf(N)$ when $N \geq n_0$.

Some more definitions

Big Oh

$T(N) = O(f(N))$ if there are positive constants c and n_0 such that $T(N) \leq cf(N)$ when $N \geq n_0$.

Omega

$T(N) = \Omega(f(N))$ if there are positive constants c and n_0 such that $T(N) \geq cf(N)$ when $N \geq n_0$.

Some more definitions

Theta

$T(N) = \Theta(f(N))$ if and only if $T(N) = O(f(N))$ and $T(N) = \Omega(f(N))$.

Some more definitions

Theta

$T(N) = \Theta(f(N))$ if and only if $T(N) = O(f(N))$ and $T(N) = \Omega(f(N))$.

Little oh

$T(N) = o(f(N))$ if for all constants c there exists an n_0 such that $T(N) < cf(N)$ when $N > n_0$. This means: $T(N) = O(f(N))$ and $T(N) \neq \Theta(f(N))$.

Examples

- $1000 = O(1)$

Examples

- $1000 = O(1)$
- $1 = O(1000)$

Examples

- $1000 = O(1)$
- $1 = O(1000)$
- $1000 = \Omega(1)$

Examples

- $1000 = O(1)$
- $1 = O(1000)$
- $1000 = \Omega(1)$
- $1 = \Omega(1000)$

Examples

- $1000 = O(1)$
- $1 = O(1000)$
- $1000 = \Omega(1)$
- $1 = \Omega(1000)$
- $1000 = \Theta(1)$

Examples

- $1000 = O(1)$
- $1 = O(1000)$
- $1000 = \Omega(1)$
- $1 = \Omega(1000)$
- $1000 = \Theta(1)$
- $1 = \Theta(1000)$

Examples

- $1000N = O(N)$

Examples

- $1000N = O(N)$
- $N = O(1000N)$

Examples

- $1000N = O(N)$
- $N = O(1000N)$
- $1000N = \Omega(N)$

Examples

- $1000N = O(N)$
- $N = O(1000N)$
- $1000N = \Omega(N)$
- $N = \Omega(1000N)$

Examples

- $1000N = O(N)$
- $N = O(1000N)$
- $1000N = \Omega(N)$
- $N = \Omega(1000N)$
- $1000N = \Theta(N)$

Examples

- $1000N = O(N)$
- $N = O(1000N)$
- $1000N = \Omega(N)$
- $N = \Omega(1000N)$
- $1000N = \Theta(N)$
- $N = \Theta(1000N)$

Examples

- $N = O(N)$

Examples

- $N = O(N)$
- $N = O(N^2)$

Examples

- $N = O(N)$
- $N = O(N^2)$
- $N^2 = \Omega(N)$

Examples

- $N = O(N)$
- $N = O(N^2)$
- $N^2 = \Omega(N)$
- $\log N = O(N)$

Rule 1

If $T_1(N) = O(f(N))$ and $T_2(N) = O(g(N))$, then

- $T_1(N) + T_2(N) = O(f(N) + g(N))$
- $T_1(N) \cdot T_2(N) = O(f(N) \cdot g(N))$

Rule 2

If $T(N)$ is a polynomial of degree k, then $T(N) = \Theta(N^k)$.

Rule 3

$\log^k N = O(N)$ for any constant k.

Matters of Style

- Writing $T(N) = O(3N^2)$ is bad style. Why?

Matters of Style

- Writing $T(N) = O(3N^2)$ is bad style. Why?
  Because $T(N) = O(N^2)$ holds. The constant 3 does not matter!
- Writing $T(N) = O(N^2 + N)$ is bad style. Why?

Matters of Style

- Writing $T(N) = O(3N^2)$ is bad style. Why?
  Because $T(N) = O(N^2)$ holds. The constant 3 does not matter!
- Writing $T(N) = O(N^2 + N)$ is bad style. Why?
  Because $T(N) = O(N^2)$ holds. The low-order term N does not matter!

This Week

- Thursday Crash Course:
  - Languages and language processors
  -
  - Recursion and iteration
  -
  - Lists

This Week

- Thursday Crash Course:
  - Languages and language processors
  -
  - Recursion and iteration
  -
  - Lists
- Friday lecture: Running time calculations (Section 2.4)

This Week

- Thursday Crash Course:
  - Languages and language processors
  -
  - Recursion and iteration
  -
  - Lists
- Friday lecture: Running time calculations (Section 2.4)
- Friday Crash Course: Loops