

INF102 Algorithms and Data Structures

Marc Bezem¹

¹Department of Informatics
University of Bergen

Fall 2015

INF102

- ▶ Lecturer: Marc Bezem, teaching assistants: NN
- ▶ Homepage: [INF102](#) (hyperlinks in red)
- ▶ Textbook: [Algorithms, 4th edition](#), R. Sedgewick and K. Wayne, Pearson, 2011
- ▶ Prerequisites: INF100 + 101 (\approx Ch. 1.1 + 1.2)
- ▶ Syllabus (pensum): Ch. 1.3–1.5, Ch. 2, Ch. 3, Ch. 4
- ▶ Exam: two or three compulsory exercises and a [written exam](#)
- ▶ Old exams: [2004–2013](#), [2014](#)
- ▶ Contents of these slides [here](#)

Didactical stuff

- ▶ Good textbook from USA: many pages, exercises etc.
- ▶ Average speed must be ca 50 pages p/w
- ▶ Lectures focus on the essentials
- ▶ Prepare yourself by reading in advance
- ▶ Workshops about selected exercises
- ▶ Test yourself by trying some exercises in advance
- ▶ If you can do the exercises (incl. compulsory), you are fine

Generic Bags, Queues and Stacks

- ▶ Generic programming in Java, example: **PolyPair**
- ▶ Bag, Queue and Stack are generic, iterable collections
- ▶ Queue and Stack: Ch. 9 in textbook INF100/1
- ▶ APIs include: `boolean isEmpty()` and `int size()`
- ▶ All three support adding an element
- ▶ Queue and Stack support removing an element (if any)
- ▶ FIFO Queue, LIFO Stack
- ▶ Dijkstra's Two-Stack Expression Evaluation **Movie**

Implementations

- ▶ `ResizingArray_Stack.java`
- ▶ Resizing takes time and space proportional to size
- ▶ `LinkedList_Stack.java`
- ▶ Pointers take space and dereferencing takes time
- ▶ Programming with pointers: make a picture
- ▶ `LinkedList_Queue.java`

Computation time and memory space

- ▶ Two central questions:
 - ▶ How long will my program take?
 - ▶ Will there be enough memory?
- ▶ Example: TheeSum
- ▶ Inner loop is important

Methods of Analysis

- ▶ Empirical:
 - ▶ Run program with randomized inputs, measuring time & space
 - ▶ Run program repeatedly, doubling the input size
 - ▶ Measuring time: **StopWatch**
 - ▶ Plot, or log-log plot and **linear regression**
- ▶ Theoretical:
 - ▶ Define a cost model by abstraction (e.g., array accesses, comparisons, operations)
 - ▶ Try to count/estimate/average this cost as function of the input (size)
 - ▶ Use $O(f(n))$ and $f(n) \sim g(n)$

ThreeSum, empirically

- ▶ Input sizes 1K, 2K, 4K, 8K take time 0.1, 0.8, 6.4 ,51.1 sec
- ▶ The log's are 3, 3.3, 3.6, 3.9 and -1, -0.1, 0.8, 1.71
- ▶ Linear regression gives $y \approx 3x - 10$
- ▶ $\lg(f(n)) = 3 \lg(n) - 10$ iff

$$f(n) = 10^{\lg(f(n))} = 10^{3 \lg(n) - 10} = n^3 * 10^{-10}$$

- ▶ Conclusion: cubic in the input size, with constant $\approx 10^{-10}$
- ▶ Strong dependence on input can be a problem
- ▶ Constant 10^{-10} depends on computer, exponent 3 does not

ThreeSum, theoretically

- ▶ Number of different picks of triples: $g(n) = n(n-1)(n-2)/6$
- ▶ Inner loop executed $g(n)$ times
- ▶ $g(n) = n^3/6 - n^2/2 + n/3$
- ▶ Cubic term $n^3/6$ wins for large n
- ▶ Computational model # array accesses: $n^3/2$
- ▶ Cost array access t sec: time $t * n^3/2$ sec

Big Oh, and \sim

- ▶ Q: 'wins for large n ' uhh???
- ▶ A: Big Oh, and \sim will clear this up
- ▶ Costs are positive quantities, so $f, g, \dots : \mathbb{N} \rightarrow \mathbb{R}^+$
- ▶ MNF130: $f(n)$ is $O(g(n))$ if there exist c, N such that $f(n) \leq cg(n)$ for all $n \geq N$
- ▶ Example: n^2 and even $99n^3$ are $O(n^3)$, but n^3 is not $O(n^{2.9})$
- ▶ INF102: $f(n) \sim g(n)$ if $1 = \lim f(n)/g(n)$
- ▶ If $f(n) \sim g(n)$, then $f(n)$ is $O(g(n))$ and $g(n)$ is $O(f(n))$
- ▶ Big Oh and \sim aim to capture 'order of growth'
- ▶ Big Oh abstracts from constant factors, \sim does not
- ▶ Large constant factors are important!

Important orders of growth

- ▶ constant: c ($f(n) = c$ for all n)
- ▶ linear: n (compare all for $n = 20$ sec)
- ▶ linearithmetic: $n \lg n$
- ▶ quadratic: n^2
- ▶ cubic: n^3
- ▶ exponential: 2^n
- ▶ general form: $an^b(\lg n)^c$

Examples

- ▶ Worst case: guaranteed, independent of input
- ▶ Average case: not guaranteed, dependent of input *distribution*
- ▶ Linked list implementations of Stack, Queue and Bag: all operations take constant time in the worst case
- ▶ Resizing array implementations of Stack, Queue and Bag: adding and deleting take linear time in the worst case (easy)
- ▶ Resizing array implementations of Stack, Queue and Bag: adding and deleting take on average constant time in the worst case (difficult)
- ▶ Special case of resizing array that is only growing:
12(2)3(4)56(8)789(16)10 ... 17(32), with (n) the new size.
Resizing to (n) costs $2n$.

ToC and topics of general interest

- ▶ Table of Contents on next slide (all items clickable)
- ▶ Practical stuff: slide 2

Introduction

Ch.1.3 Bags, Queues and Stacks

Ch.1.4 Analysis of Algorithms

Table of Contents