Lecture 1
Data Structures (DAT037)

Ramona Enache

November 3, 2014

Data Structures

Data Structures

A **data structure** is

- a way to store and organize information.
- optimized for access or modification of particular kinds

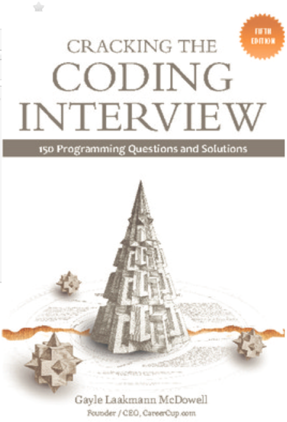
Why Data Structures ?

Near future:

- further study of algorithm and applications
- compilers, programming language technology
- operating systems, hardware verification
- **most programming-based courses** from now on!

Why Data Structures ?

More distant future:



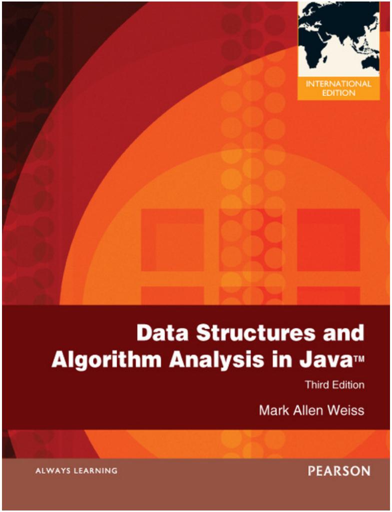
Teachers

- **Lecturer:** Ramona Enache
  `ramona . enache 'at' chalmers . se`
- **Assistants:**
  - Anton Ekblad
  - Inari Listenmaa
  - Olof Mogren
  - Andreas Widén

Homepage

Course Web Page

Course Book


INTERNATIONAL EDITION
Data Structures and Algorithm Analysis in Java™
Third Edition
Mark Allen Weiss
ALWAYS LEARNING
PEARSON

Labs

- Work in pairs
- Contact Ramona for:
  - help with pairing up
  - exceptions for working alone (need good reasons!)
  - deadline extensions for labs - **always before the actual deadline**
- last week of the course - you have an extra chance if you passed 2 out of 3 labs.

Partial exam

- **optional**
- December 1st
- 3 exercises
- the result from each of them can replace the result from the same exercise in the final exam

Uniform Cost Model

- each instruction takes 1 unit of time
- each variable takes 1 unit of memory
- infinite memory
- no cache, I/O
- only for theoretical purposes - measure **time** and **space** for programs!

Example

How many steps does the algorithm take for `n = a.length` stricly positive ?

Example

```
int min=a[0];
for (int i=1; i<a.length; i++)
     if (a[i] < min) min = a[i];
```

Think about best-case and worst-case scenarios!

Big O notation

- simplify complexity analysis
- approximate final result
- enough to judge the general performance of an algorithm

Big O notation

Let us consider, for a given algorithm:

- a function T(n) giving the number of steps for an input of size n
- the same procedure can be used for measuring space

We say "T(n) is O(f(n))" if $\exists n_0$ such that
a $\times$ f(n) $\leq$ T(n) $\leq$ b $\times$ f(n),
for 2 constants a and b, $\forall n \geq n_0$.

Why n_0 ?

- for a small n, the complexity difference is not noticeable
- for a small n, the constant fact matters more

Back to example

Best and worst-case scenario reduce to $\mathbf{O(n)}$

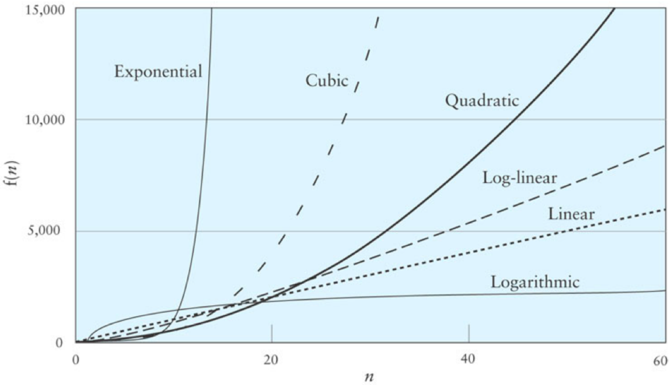
Big-O	Name
$O(1)$	Constant
$O(\log n)$	Logarithmic
$O(n)$	Linear
$O(n \log n)$	Log-linear
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(2^n)$	Exponential
$O(n!)$	Factorial

Growth rates

What happens if we double the input size from n to 2n ?
If an algorithm is...

- O(1), then it takes the same time as before
- O(log n), then it takes a constant amount of time more
- O(n), then it takes twice as long
- O(n log n), then it takes twice as long + a bit more
- O(n^2), then it takes 4 times more time

- If the algorithm is O(2^n), then adding one more item, makes the algorithm take 2 times more time.


15,000
10,000
5,000
0
f(n)
0
20
40
60
n
Exponential
Cubic
Quadratic
Log-linear
Linear
Logarithmic

Big O hierarchy

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$
Adding a term to another term which is lower in the hierarchy doesn't change the complexity

- $O(1) + O(\log n) = O(\log n)$
- $O(\log n) + O(n^k) = O(n^k)$ (if $k \geq 0$)
- $O(n^j) + O(n^k) = O(n^k)$ (if $k \geq j$)
- $O(n^k) + O(2^n) = O(2^n)$

More examples

What is the complexity of :

Example

```
for(int i=0; i<v.length; i++)
  for(int j=i+1; j<v.length; j++)
     if (v[i]>v[j]) return false;
 return true;
```

Exact answer: (n-1) + (n-2) + ...+1 = **n(n-1)/2**
Answer: $\mathbf{O}(n^2)$

More examples

What about the following algorithm that determines (in a non-optimal way) if there is at most one 0 in the matrix a of size n $\times$ n.

Example

```
for (int i=0; i<n; i++)
  for(int j =0; j<n; j++)
    if (a[i][j]==0) then
        for (int i1=0; i1<n; i1++)
          for (int j1=0; j1<n; j1++)
            if (a[i1][j1] ==0 && (i1 != i || j1 != j))
               then return 0;
return 1;
```

Answer: $O(n^4)$

What about $(n+1)^2(2^n+14)$?

Answer: $O(n^2 2^n)$

TO DO

- read from course book - **Chapter 2**
  - more about average-case complexity
  - non-polinomial complexities
- refresh your knowledge on **proofs by induction** – exercise session
- all these will make more sense later!
- find a **lab partner**!!