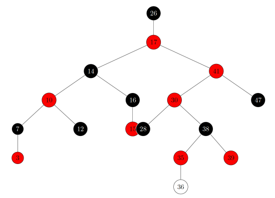


Introduction
Lecture 1

COSC 242 – Algorithms and Data Structures

Today’s outline

1. Course overview
2. Course goals
3. COSC 241 and 242
4. Algorithms and Analysis

Today's outline

1. Course overview
2. Course goals
3. COSC 241 and 242
4. Algorithms and Analysis

Teaching team

Dr. Steven R. Livingstone

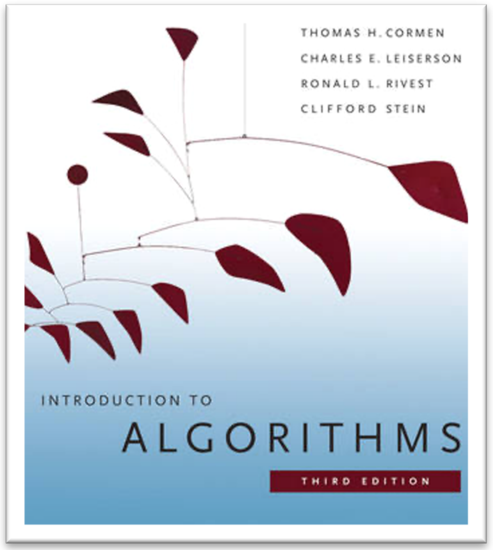
- Lectures
- Office: Owheo building, Room G.30
- Email: s.livingstone@otago.ac.nz

Mr. Iain Hewson

- Labs
- Office: Owheo building, Room G.37A
- Email: ihewson@cs.otago.ac.nz

Material covered

Textbook (recommended): Introduction to algorithms, by Cormen, Leiserson, Rivest, and Stein, any edition, MIT Press. I will use 3rd edition section numbers in class.



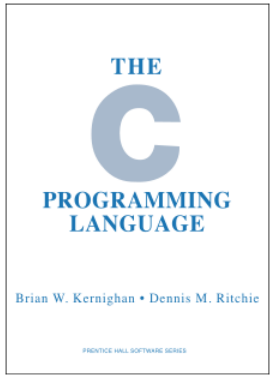
Other materials

Lecture notes, Lab book, and Assessment

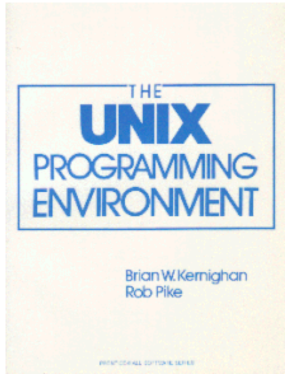
http://www.cs.otago.ac.nz/cosc242

Recommended readings

The C Programming Language, by Kernighan and Ritchie, Prentice Hall, 2nd Edition, 1988. Often called K & R, this classic is still widely regarded as the best C programming book.



The Unix Programming Environment by Kernighan and Pike



Course structure

COSC 242 focuses on learning theoretical concepts (lectures), which are then solidified through implementation (labs).

Lectures: Mondays and Thursdays, 11am-12pm, AUDIT and Archway 4

Labs: 2x hours per week

Tutorials: Mondays, 4-5pm, Quad 2. Starts in Week 2.

Assessment

Item	**Quantity**	**Form**	**Weight**
Lab assessment	1	Submission	2%
Practical test	3	Test	23%
Assignment	1	Submission	15%
Final Exam	1	Test	60%

Assessment: Labs & Practical tests

Labs and practical tests make up 25% of your final grade.

Lab 4 is an assignment-style submission.

Practical tests take place in the lab at a designated time & place.

Name	**Lab number**	**Weight**	**Form**	**Date**
Lab assessment	4	2%	Submission	Friday 17/07
Practical test	7	3%	Practical test	Tuesday 28/07
Practical test	13	8%	Practical test	Tuesday 18/08
Practical test	22	12%	Practical test	Fri 25/09 or Tues 29/09

Academic integrity

Academic integrity means being honest in your studying and assessments. It is the basis for ethical decision-making and behaviour in an academic context.

Academic integrity is informed by the values of honesty, trust, responsibility, fairness, respect and courage.

Students are expected to be aware of, and act in accordance with, the University's Academic Integrity Policy.

Academic Misconduct

Academic Misconduct, such as plagiarism or cheating, is a breach of Academic Integrity and is taken very seriously by the University.

Types of misconduct include: plagiarism, copying, unauthorised collaboration, taking unauthorised material into a test or exam, impersonation, and assisting someone else's misconduct.

A more extensive list of the types of academic misconduct and associated processes and penalties is available in the University's Student Academic Misconduct Procedures.

Academic responsibility

It is your responsibility to be aware of and use acceptable academic practices when completing your assessments.

To access the information in the Academic Integrity Policy and learn more, please visit the University's Academic Integrity website at www.otago.ac.nz/study/academicintegrity or ask at the Student Learning Centre or Library.

If you have any questions, ask Steven or Iain.

Academic integrity and misconduct resources

Academic Integrity Policy

www.otago.ac.nz/administration/policies/otago116838.html

Student Academic Misconduct Procedures

http://www.otago.ac.nz/administration/policies/otago116850.html

A brief guide for students

https://www.otago.ac.nz/study/otago703173.pdf

Academic integrity in COSC242

Acceptable and encouraged ✓

- Collaborating with other students on ideas.
- Giving some help to other students by helping them debug.
- All code you submit for any assessment is written by you. This excludes the scaffolding code provided by the teaching team.

Not acceptable ✗

- Taking over the keyboard and writing a piece of code for someone else.
- Sending someone a copy of your code.
- Copying code from another student, or from an online source.

Class Reps Wanted!

Are you?

✓ proactive, friendly and keen to contribute to your learning environment?

✓ A great communicator who can represent your peers?

What's in it for you?

→ FREE support

→ A reference letter from OUSA for your CV

→ Professional development training to boost your CV

→ FREE food

signs of the island being brought back to life anyway, I had

Volunteer to represent your class now!

Applications close July 17th

Scan the QR code above to register your interest in being a class rep or go to https://classrep.ousa.org.nz/login.php. If you are selected to be a class rep you will receive a confirmation email. If you have any issues or concerns throughout the semester email academic@ousa.org.nz or dione@ousa.org.nz

Lecture attendance

Lecture attendance is *expected*, but not required.

Attendance is the biggest predictor of university grades.

Research has repeatedly shown that attendance is more important than standardized admissions test scores (NCEA, ATAR/OP/SAT), study habits, and study skills [1, 2].

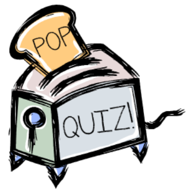
1 - https://goo.gl/x9v46X

2 - https://goo.gl/oQMPTz

Lecture conventions

Solo activity (by yourself)

Group activity (2-3 people)



Class question

Important slide

Today’s outline

1. Course overview
2. Course goals
3. COSC 241 and 242
4. Algorithms and Analysis

Course goals

1. Foster a mindset that focuses on the efficiency of your programs
2. Introduce more advanced algorithms and data structures than those you met in COSC241
3. Give you experience in building both data structures and algorithms from scratch
4. Learn a new programming language (C).

Goal 1: Mindset of efficiency

Why focus on efficiency?

- We're seeing the end of Moore's Law.
- Moore's Law - the doubling of the number of transistors on a chip every 2 years - meant that resources became plentiful and priority shifted from efficiency to ease of coding.
- CMOS, the best semiconductor technology, cannot be scaled down further, and Moore's Law is breaking down.
- That means we can’t expect hardware to pick up our slack.

Goal 2: Advanced algorithms and DS

Why know about more advanced A&DS?

- Because the more advanced algorithms that we'll cover in COSC242 allow for greater search efficiency.
- Examples of advanced data structures include RBTs and hash tables.
- Examples of advanced algorithms include dynamic programming.

Goal 3: Hands on experience

Why bother implementing things that already exist?

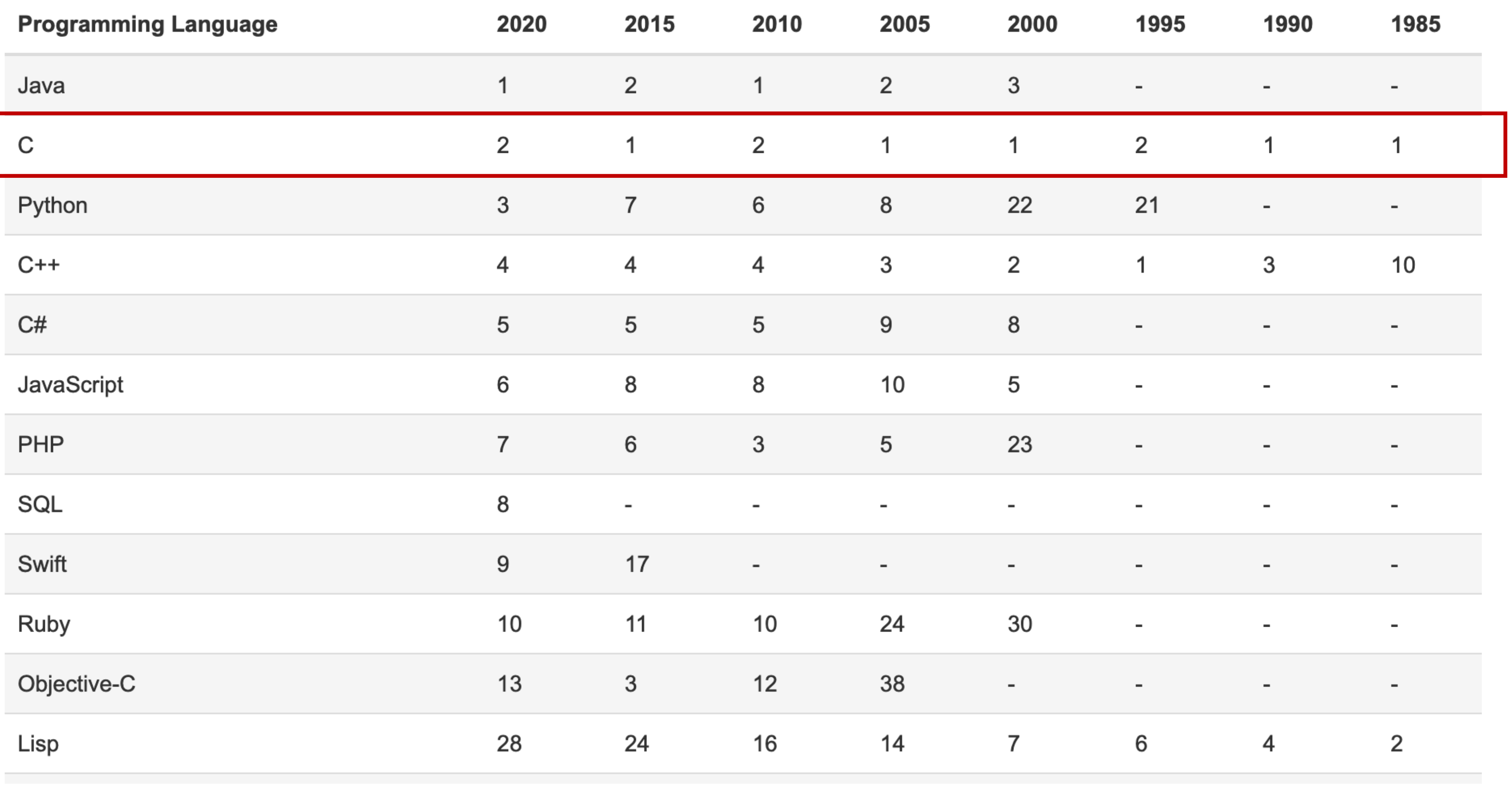
- Almost every algorithm and data structure we'll cover are available in most common languages.
- But using those implementations can cause serious problems if you don't understand the trade-offs.
- You will only learn the intricate details by implementing them yourself.
- You will also learn not to rely on the code completion functionality of IDE's uncritically.

Goal 4: Learn C

I know Java and Python, why learn C?

- Because C doesn't hide overheads the way Java does.
- C isn’t forgiving like Java, and you’ll develop a deeper understanding of how computers work.
- Because C remains one of the most popular programming languages. Given its durability and longevity, if you’re good at C, you’ll remain readily employable.

Programming language popularity



Programming Language	2020	2015	2010	2005	2000	1995	1990	1985
Java	1	2	1	2	3	-	-	-
C	2	1	2	1	1	2	1	1
Python	3	7	6	8	22	21	-	-
C++	4	4	4	3	2	1	3	10
C#	5	5	5	9	8	-	-	-
JavaScript	6	8	8	10	5	-	-	-
PHP	7	6	3	5	23	-	-	-
SQL	8	-	-	-	-	-	-	-
Swift	9	17	-	-	-	-	-	-
Ruby	10	11	10	24	30	-	-	-
Objective-C	13	3	12	38	-	-	-	-
Lisp	28	24	16	14	7	6	4	2

https://www.tiobe.com/tiobe-index/

https://www.youtube.com/watch?v=Og847HVwRSI

Course goals

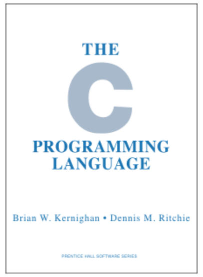
Foster a mindset that focuses on the efficiency of your programs

Introduce more advanced algorithms and data structures than those you met in COSC241

Give you experience in building both data structures and algorithms from scratch

Learn a new programming language (C). Learning C will happen primarily in the labs.

Lectures, Labs, and C


THE
C
PROGRAMMING LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE HALL SOFTWARE SERIES

Sometimes we'll do efficiency content in the lectures, and practical stuff in the labs.

Therefore, it may seem that labs and lectures are mismatched. This is intentional and purposeful.

Attending both lectures and labs is very important.

The assignment will be easy if you have completed the labs.

The exam is based on lectures, specifically, in-class examples.

Today’s outline

1. Course overview
2. Course goals
3. COSC 241 and 242
4. Algorithms and Analysis

COSC 241 and 242

COSC 241 and 242 are tied together by the fact that we revisit some of the same algorithms.

This is to give you a perspective of how Java and C differ. Both languages have their strengths and weaknesses.

We will also pay more attention to calculating efficiencies in 242.

COSC242 isn't a math paper, so we won't overwhelm you with calculating complicated formulas for all the algorithms.

However, we do want you to start getting used to working with big-O calculations.

COSC 241 and 242

We'll keep the math aspect as simple as possible.

The first few lectures are about how to do these calculations in such a way that you can convince your fellow programmers that you've done it correctly.

At the same time you'll be learning the basics of C in the labs, so for a couple of weeks it will seem as if the lectures and labs don't connect.

Today's outline

1. Course overview
2. Course goals
3. COSC 241 and 242
4. Algorithms and Analysis

Algorithm analysis

COSC241 introduced you to the analysis of algorithms, and made the following points:

- To calculate the time efficiency of a program, ignore implementation details (e.g. clock speed). We even ignore the programming language used. So we'll talk of *algorithms* instead of programs.
- How long an algorithm takes depends on how much work the algorithm has to do.
- The amount of work is the number of basic steps the algorithm takes.

Algorithm analysis

We want to relate the amount of work to the number n of data items that need to be processed (the *problem size*).

We want to compare the work this algorithm does with that of other algorithms for producing the same result.

Algorithm analysis

Finally, as professional programmers, we will always be part of a team.

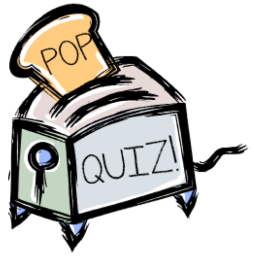
Therefore, we need to know not only how to calculate the amount of work but how to share our calculation with co-workers in a clearly understandable way.

Algorithms

Programs are written in a programming language such as C or Java; all details must be spelled out for the compiler.

Algorithms are written in **pseudocode**: English enriched by symbols so we can clearly understand the essence of what must be done without drowning in detail.

Pop quiz 1



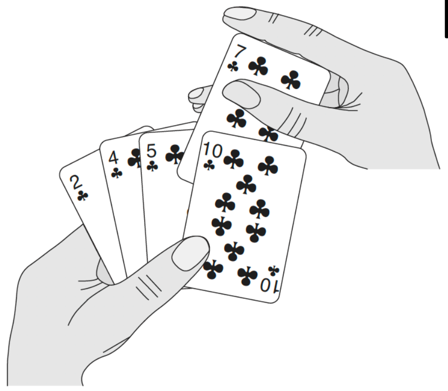
Other than speed, what other measures of efficiency might one use in a real-world setting?

Insertion sort

Lets look at an one such algorithm, **insertion sort**.

Insertion sort is an efficient algorithm for sorting a small number of elements.

Insertion sort works the way many people sort a hand of playing cards.



[1]

Class discussion

Given a deck of cards face down, how would you sort them into your hand?

Sorting problem

Input: A sequence of n numbers $\langle a_1, a_2, a_3, .., an \rangle$

Output: A permutation (reordering) $\langle a'_1, a_2, a'_3, .., a'_n \rangle$ of the input sequence such that: $a'_1 < a'_2 < a'_3 < ... < a'_n$

Key: The numbers we wish to sort

Lets define a solution using pseudocode.

Pseudocode

Pseudocode is similar to programming languages, such as Java, C, or Python.

Pseudocode is designed for expressing algorithms to *humans*. Software engineering issues of data abstraction, modularity, and error handling are often ignored.

We sometimes embed English statements into pseudocode.

Pseudocode solution

Our pseudocode solution will take in an array $\mathrm{A}[1..\mathrm{n}]$, containing a sequence of length n, to be sorted.

The algorithm sorts the input numbers **in place**: it rearranges the numbers within the array A, with at most a constant number of them stored outside the array at any time.

The input array A contains the sorted output sequence when the INSERTION-SORT procedure is finished.

Insertion sort

```
1. for j = 2 to A.length                             j = indicates “current card”
2.     key = A[j]
3.     // Insert A[j] into the sorted sequence A[1..j-1]
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
```

Loop invariant

A[1.. j-1] constitutes the currently sorted hand.

The remaining subarray A[j + 1..n] corresponds to the pile of cards still on the table.

Loop invariant

At the start of each iteration of the “outer” for loop, the loop indexed by j, the subarray A[1 .. j-1] consists of the elements originally in A[1..j-1] but in sorted order.

Insertion sort

Team up with the people next to you. Trace the operation of Insertion Sort on each of the following input arrays:

[1 2 3 4 5]

[5 4 3 2 1]

Questions

1. Which exercise represents a best, and which a worst case?
2. In each case, how many comparisons have to be made between the value key and array entries when i = 1? When i = 2? When i = n - 1?
3. What is the total number of comparisons in the best case? What about in the worst case?

Insertion sort

Questions to think about

1. How much work does insertion sort do?
2. How does this compare with other sorting algorithms?

Answering these two questions will take us several lectures!

Insertion sort

Important questions

1. Is the amount of work done by insertion sort a lot or a little? What benchmark can we use?
2. Is it the amount of work for a specific size of input that we care about, or whether the algorithm scales up?

Landmarks

We saw that the amount of work done by Insertion Sort, in the worst case, is roughly indicated by:

$$f(n) = 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} = \frac{n^2 - n}{2}$$

If you’d like to see how we arrived at this function, see Section 2.1 of the textbook.

Landmarks

Insertion Sort

$$f(n) = 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$$

We'd like to tie this in to some special **landmark functions**, which are given by assigning to input n the outputs:

$$f(n) = 1 \qquad f(n) = \log n \qquad f(n) = n$$

$$f(n) = n \log n \qquad f(n) = n^2 \qquad f(n) = n^3$$

$$\dots$$

$$f(n) = 2^n \qquad f(n) = n!$$

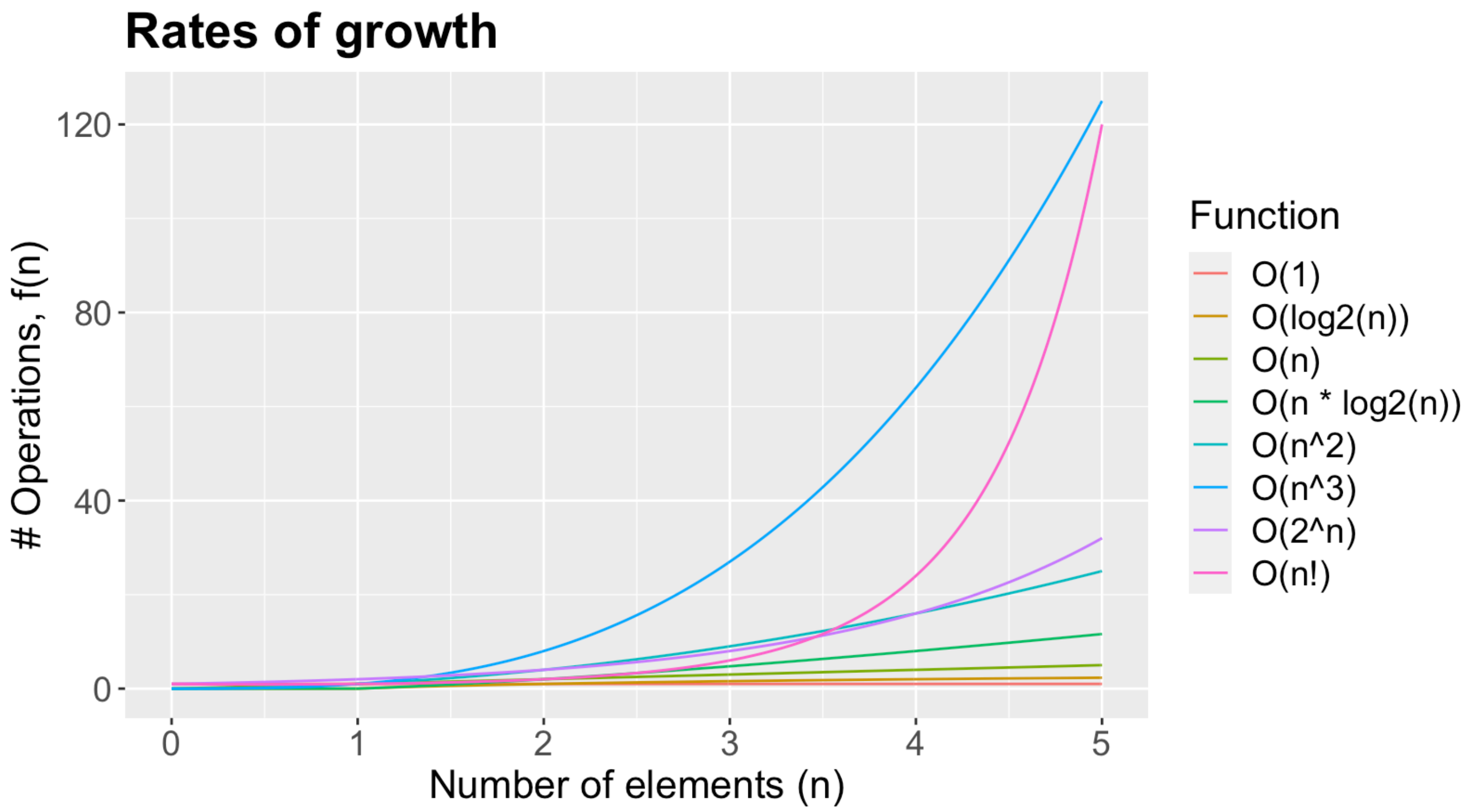
Landmarks

These simple functions are landmarks in the sense that we use their rates of growth to group other functions into classes.

Functions that most closely resemble, say, n^2 form one class, those resembling n^3 another, etc.

Rates of Growth

Growth rate means how fast its output $f(n)$ increases in size as the input n gets bigger. Intuitively, a function with a slow rate of growth scales up better than a function having a high rate of growth.

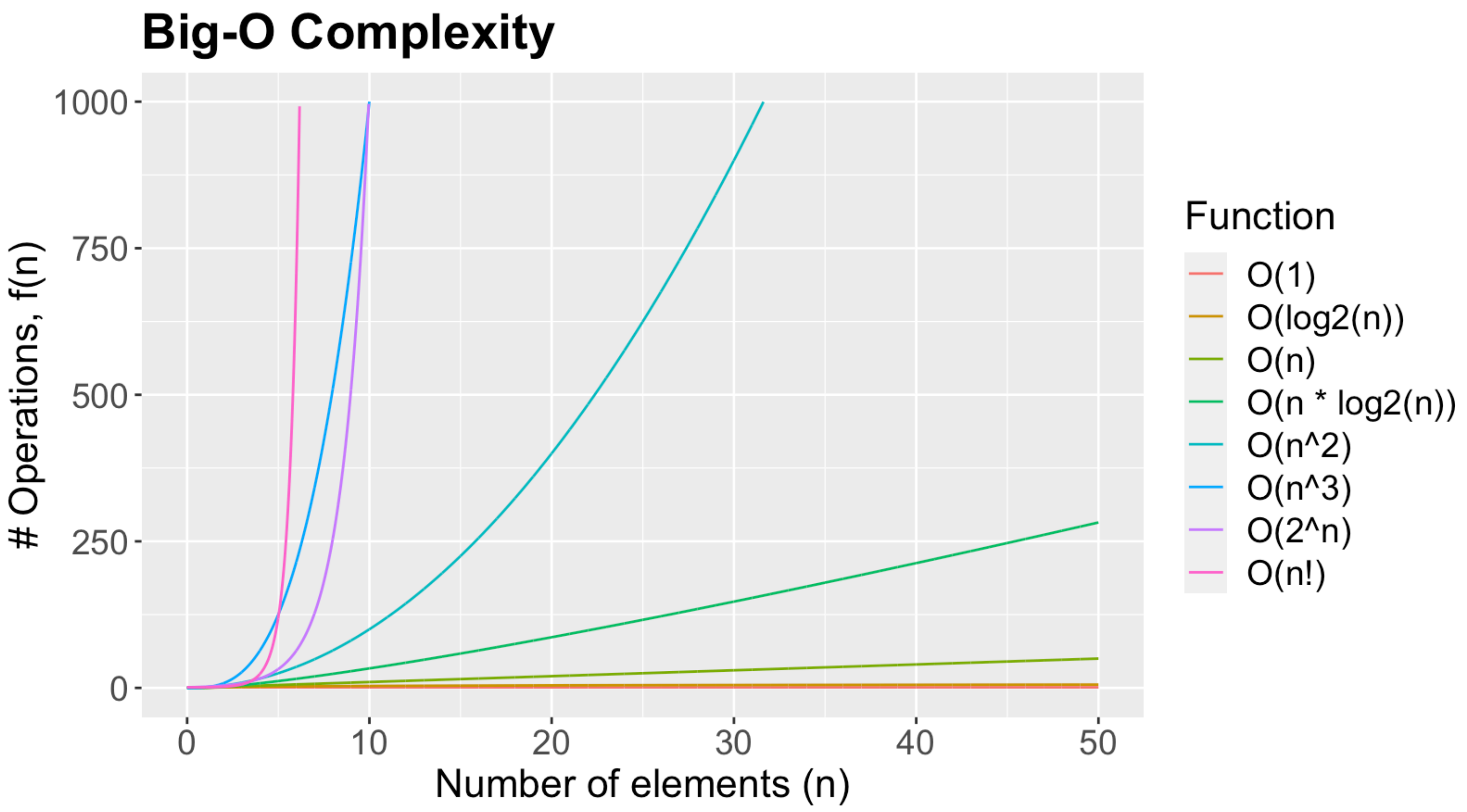


Rates of Growth (Zooming out…)

Order of growth

description	**function**
constant	1
logarithmic	$\log N$
linear	N
linearithmic	$N \log N$
quadratic	N^2
cubic	N^3
exponential	2^N
factorial	$N!$

Commonly encountered order-of-growth functions



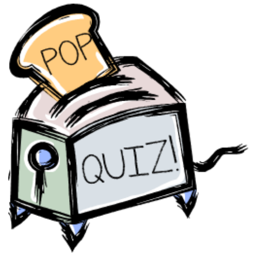
Suggested reading

Insertion sort is discussed in section 2.1*.

*Section numbers refer to those in the 3rd edition.

Solutions

Pop quiz 1



Other than speed, what other measures of efficiency might one use in a real-world setting?

Answers

Memory consumption, bandwidth usage, hardware design

Card sorting solution

1. We start with an empty left hand and the cards face down on the table.
2. We then remove one card at a time from the table and insert it into the correct position in the left hand.
3. To find the correct position for a card, we compare it with each of the cards already in the hand, from right to left.
4. At all times, the cards held in the left hand are sorted, and these cards were originally the top cards of the pile on the table.

Insertion sort

These examples and others give us an understanding of Insertion Sort as being better when input data is already nearly sorted and worse when input data is nearly reverse sorted.

Best case: 1+1+1+...+1 = n-1

Worst case: 1 + 2 + 3 + 4 = 1 + 2 + ... + (n-1) = n(n-1)/2

References

1. Introduction to algorithms, 2009, by Cormen, Leiserson, Rivest, and Stein, any edition, MIT Press (3rd edition).

Image attributions

- This Photo by Unknown Author is licensed under CC BY