Week 2

Big-O Notation

Thorben Klabunde

www.th-kl.ch

29.09.2025

Agenda

1. Mini-Quiz
2. Assignment
3. Theory Recap
4. Additional Practice
5. Peer Grading

Mini-Quiz

Assignment

Ex. 1.1)

Exercise 1.1 *Sum of Cubes* **(1 point).**

Prove by mathematical induction that for every positive integer n,

$$1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}.$$

Ex. 1.2)

Exercise 1.2 *Sum of reciprocals of roots* **(1 point)**.

Consider the following claim:

$$\frac{1}{\sqrt{1}} + \frac{1}{\sqrt{2}} + \cdots + \frac{1}{\sqrt{n}} \leq \sqrt{n}.$$

A student provides the following induction proof. Is it correct? If not, explain where the mistake is.

Base case: $n = 1$,

$$\frac{1}{\sqrt{1}} \leq 1, \text{ which is true.}$$

Induction hypothesis: Assume the claim holds for $n = k$, i.e.

$$\frac{1}{\sqrt{1}} + \cdots + \frac{1}{\sqrt{k}} \leq \sqrt{k}.$$

Induction step: Then, starting from the claim we need to prove for $n = k + 1$ and using logical equivalences:

$$\begin{aligned}
\frac{1}{\sqrt{1}} + \cdots + \frac{1}{\sqrt{k}} + \frac{1}{\sqrt{k+1}} \leq \sqrt{k+1} &\iff \frac{1}{\sqrt{1}} + \cdots + \frac{1}{\sqrt{k}} \leq \sqrt{k+1} - \frac{1}{\sqrt{k+1}} \\
&\iff \frac{1}{\sqrt{1}} + \cdots + \frac{1}{\sqrt{k}} \leq \frac{k+1}{\sqrt{k+1}} - \frac{1}{\sqrt{k+1}} \\
&\iff \frac{1}{\sqrt{1}} + \cdots + \frac{1}{\sqrt{k}} \leq \frac{k}{\sqrt{k+1}} \leq \frac{k}{\sqrt{k}} \leq \sqrt{k},
\end{aligned}$$

which is true, therefore the claim holds by the principle of mathematical induction.

Ex. 1.2 - Remarks on Proof Technique

Key Elements of an Inductive Proof

When writing the inductive step, you must clearly distinguish between:

- **The statement to be proved**.
  - Here: Show that $\sum_{i=1}^{n+1} \frac{1}{\sqrt{i}} \leq \sqrt{n+1}$ holds (**it doesn't**, don't try to show it ;))
- **The allowed assumptions.**
  - Here, the I.H.: Assume $\sum_{i=1}^{n} \frac{1}{\sqrt{i}} \leq \sqrt{n}$ is true for some $n \in \mathbb{N}$.

Ex. 1.2 - Remarks on Proof Technique

Warning: Correct proofs must build from valid assumptions towards the desired conclusion.

- **Correct Logic:** Start with the I.H. (a true statement, A) and show through a series of valid steps that it implies your goal (statement B). This proves B.
- **Fallacies:**
  - Starting with your goal (B) and showing it implies a known statement A. This only proves the implication $B \implies A$, but not B itself. This would require **equivalences** ($\iff$).
  - Not checking **both implications** for equivalences ($\iff$)
  - **Only proving** the **implication** $A \implies B$ but not the statement A, does not allow us to conclude B.

Always make sure you proved all statements that you cannot assume! Writing out your proofs carefully, step-by-step with explicit explanations ensures this.

Ex. 1.2)

Exercise 1.2 *Sum of reciprocals of roots* **(1 point)**.

Consider the following claim:

$$\frac{1}{\sqrt{1}} + \frac{1}{\sqrt{2}} + \cdots + \frac{1}{\sqrt{n}} \leq \sqrt{n}.$$

A student provides the following induction proof. Is it correct? If not, explain where the mistake is.

Base case: $n = 1$,

$$\frac{1}{\sqrt{1}} \leq 1, \text{ which is true.}$$

Induction hypothesis: Assume the claim holds for $n = k$, i.e.

$$\frac{1}{\sqrt{1}} + \cdots + \frac{1}{\sqrt{k}} \leq \sqrt{k}.$$

Induction step: Then, starting from the claim we need to prove for $n = k + 1$ and using logical equivalences:

$$\begin{aligned}
\frac{1}{\sqrt{1}} + \cdots + \frac{1}{\sqrt{k}} + \frac{1}{\sqrt{k+1}} \leq \sqrt{k+1} &\iff \frac{1}{\sqrt{1}} + \cdots + \frac{1}{\sqrt{k}} \leq \sqrt{k+1} - \frac{1}{\sqrt{k+1}} \\
&\iff \frac{1}{\sqrt{1}} + \cdots + \frac{1}{\sqrt{k}} \leq \frac{k+1}{\sqrt{k+1}} - \frac{1}{\sqrt{k+1}} \\
&\iff \frac{1}{\sqrt{1}} + \cdots + \frac{1}{\sqrt{k}} \leq \frac{k}{\sqrt{k+1}} \leq \frac{k}{\sqrt{k}} \leq \sqrt{k},
\end{aligned}$$

which is true, therefore the claim holds by the principle of mathematical induction.

Ex. 1.4a)

Exercise 1.4 *Proving Inequalities.*

(a) Prove the following inequality by mathematical induction

$$\frac{1}{2} \cdot \frac{3}{4} \cdot \frac{5}{6} \cdot \ldots \cdot \frac{2n-1}{2n} \leq \frac{1}{\sqrt{3n+1}}, \quad n \geq 1.$$

In your solution, you should address the base case, the induction hypothesis and the induction step.

Ex. 1.4b)

(b)* Replace $3n + 1$ by $3n$ on the right side, and try to prove the new inequality by induction. This inequality is even weaker, hence it must be true. However, the induction proof fails. Try to explain to yourself how is this possible?

Ex. 1.3b)

(b) $f(m) = \log(m^3)$ grows asymptotically slower than $g(m) = (\log m)^3$.

Ex. 1.3c)

(c) $f(m) = e^{2m}$ grows asymptotically slower than $g(m) = 2^{3m}$.

Hint: *Recall that for all* $n, m \in \mathbb{N}$, *we have* $n^m = e^{m \ln n}$.

Ex. 1.3d)

(d)* If $f(m)$ grows asymptotically slower than $g(m)$, then $\log(f(m))$ grows asymptotically slower than $\log(g(m))$.

Ex. 1.3e)

(e)* $f(m) = \ln(\sqrt{\ln(m)})$ grows asymptotically slower than $g(m) = \sqrt{\ln(\sqrt{m})}$.

Hint: *You can use L'Hôpital's rule from sheet 0.*

Theory Recap

The Challenge: How to Measure Efficiency?

Why can't we just time our code with a stopwatch?

- The **execution time** of an algorithm **depends on the specific hardware**.
  - CPU speed and microarchitecture
  - Available memory (RAM)
  - ...
- It also depends on the **software environment**.
  - Programming language and compiler
  - Operating System

$\Longrightarrow$ We need a **common frame for comparisons** that is **independent** of these factors.

Solution Part 1: A Universal Model

We abstract away from the specific hardware by creating a simplified model.

The Unit-Cost RAM Model (Random Access Machine)

Instead of measuring seconds, we count the number of **basic operations** an algorithm performs.

- A **basic operation** is an instruction that takes a **constant** amount of time.
- Examples:
  - Arithmetic ($+, -, *, /$)
  - Comparisons ($<, >, ==$)
  - Memory access (assignments)

This gives us a runtime measured in number of operations, making our analysis **machine-independent**.

Solution Part 2: Asymptotic Analysis

Given an input instance I (a bit-string), we measure the number of operations as a function of the length n of I.

- **Problem:** Which input length to analyze?
- Instead of assuming a **specific input length**, analyze the **growth rate** of the runtime.

Asymptotic Analysis

We analyze the growth rate of the runtime as the input size n approaches infinity ($n \to \infty$).

Putting It All Together: Big-O Notation

Big-O notation combines these two ideas. It describes the **asymptotic upper bound** on the number of operations.

Big-O Notation

A function $f(n)$ is in the set $O(g(n))$, written:

$$f(n) \in O(g(n)) \quad \text{resp. in this course using the notation } f(n) \leq O(g(n)),$$

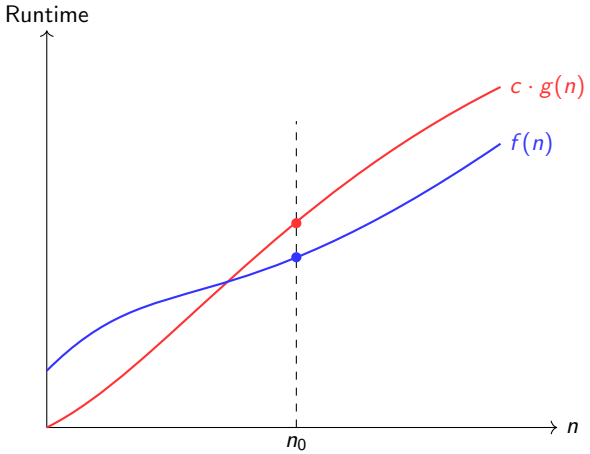
if there exist positive constants c and n_0 such that for all $n \geq n_0$:

$$0 \leq f(n) \leq c \cdot g(n)$$

In plain English: For all large inputs ($n \geq n_0$), $f(n)$ is "less than or equal to" $c \cdot g(n)$ (it is *upper bounded* by g(n)), for some constant factor $c > 0$.

Visualizing Big-O

The function $f(n)$ is our algorithm's runtime. After the point n_0, it is always below the curve of $c \cdot g(n)$.



Caution: Intuition Can Be Misleading

Asymptotic behavior only dominates for **large** values of n.

Example: Which algorithm is "better"?

- Algorithm A has a runtime of $T_A(n) = n^{1000}$.
- Algorithm B has a runtime of $T_B(n) = 1.01^n$.

Asymptotically, Algorithm A is better: $O(n^{1000})$ grows slower than $O(1.01^n)$.

Advice: Trust the formal definitions and analyze the structure (substitute values for variables that contain the critical information), don't try to mentally plot the functions:

- A is a polynomial n^k, $k \in \mathbb{N}$, whereas B is an exponential c^n with $c > 1$. $\implies$ B dominates A!

Limitations of Big-O Notation

While a sensible measure in many cases, Big-O is **not the whole story**.

- **Constant factors do matter** in practice. An algorithm running in $2n$ steps is better than one running in $1000n$ steps, even though both are $O(n)$.
- The **asymptotic view isn't always relevant**. If your application's input size is always small (e.g., $n < 100$), the asymptotically "slower" algorithm might be faster in practice.
- Big-O describes the **worst-case** scenario. Sometimes, the average-case or best-case performance is very different.

Summary: Key Takeaways

1. We need to analyze algorithms in a way that is **independent of hardware** and specific inputs.
2. We do this by **counting basic operations** as a function of input size n.
3. We analyze the **asymptotic growth rate** ($n \to \infty$) to understand how the algorithm scales.
4. **Big-O notation** provides a formal language for the **asymptotic upper bound**, ignoring constant factors.
5. It's a powerful theoretical tool, but always remember its **practical limitations**.

Additional Practice

Exercise 1.2 *Sums of powers of integers.*

(a) Show that, for all $n \in \mathbb{N}_0$, we have $\sum_{i=1}^{n} i^3 \leq n^4$.

(b) Show that for all $n \in \mathbb{N}_0$, we have $\sum_{i=1}^{n} i^3 \geq \frac{1}{2^4} \cdot n^4$.

Hint: *Consider the second half of the sum, i.e., $\sum_{i=\lceil \frac{n}{2} \rceil}^{n} i^3$. How many terms are there in this sum? How small can they be?*

Together, these two inequalities show that $C_1 \cdot n^4 \leq \sum_{i=1}^{n} i^3 \leq C_2 \cdot n^4$, where $C_1 = \frac{1}{2^4}$ and $C_2 = 1$ are two constants independent of n. Hence, when n is large, $\sum_{i=1}^{n} i^3$ behaves "almost like n^4" up to a constant factor.

(c)* Show that parts (a) and (b) generalise to an arbitrary $k \geq 4$, i.e., show that $\sum_{i=1}^{n} i^k \leq n^{k+1}$ and that $\sum_{i=1}^{n} i^k \geq \frac{1}{2^{k+1}} \cdot n^{k+1}$ holds for any $n \in \mathbb{N}_0$.

O-Notation

(a) Prove or disprove the following statements. Justify your answer.

(1) $n^{\frac{2n+3}{n+1}} = O(n^2)$

(2) $e^{1.2n} = O(e^n)$

(3) $\log(n^4 + n^3 + n^2) = O(\log(n^3 + n^2 + n))$

Peer Grading

Peer-Grading Exercise

This week's peer-grading exercise is **Exercise 1.1.**

Each group grades the group below in the table I sent you (resp. the last one grades the first one). Please send the other group your solution. If you don't get their solution, please contact me so I can send it to you.