TDDI16 Datastrukturer och algoritmer

Introduktion

LINKÖPINGS UNIVERSITET

Översikt

- Kursupplägg
  - Personal
  - Examination
  - Kursupplägg
- Algoritmer
- Analys
- Komplexitet

Personal

- Examinator
  - Rita Kovordanyi <rita.kovordanyi@liu.se>
- Kursassistenter
  - Dennis Persson <dennis.persson@liu.se>
  - Filip Strömbäck <filip.stromback@liu.se>
  - Robert Kumpulainen <robku937@student.liu.se>
- Administratör
  - Madeleine Häger Dahlqvist <madeleine.hager.dahlqvist@liu.se>

Examination

- Uppgift (UPG1)
  - Läs och arbeta igenom OpenDSA
- Laborationer (LAB1)
  - 4 labbar
- Tentamen (DAT1)
  - Dator + papper

Uppgift (OpenDSA)

- Open Source Interactive Data Structures and Algorithms
  - Läs igenom respektive kapitel inför varje föreläsning
  - Lättare att ställa kluriga frågor
- Ger en första steg in i labbarna

OpenDSA

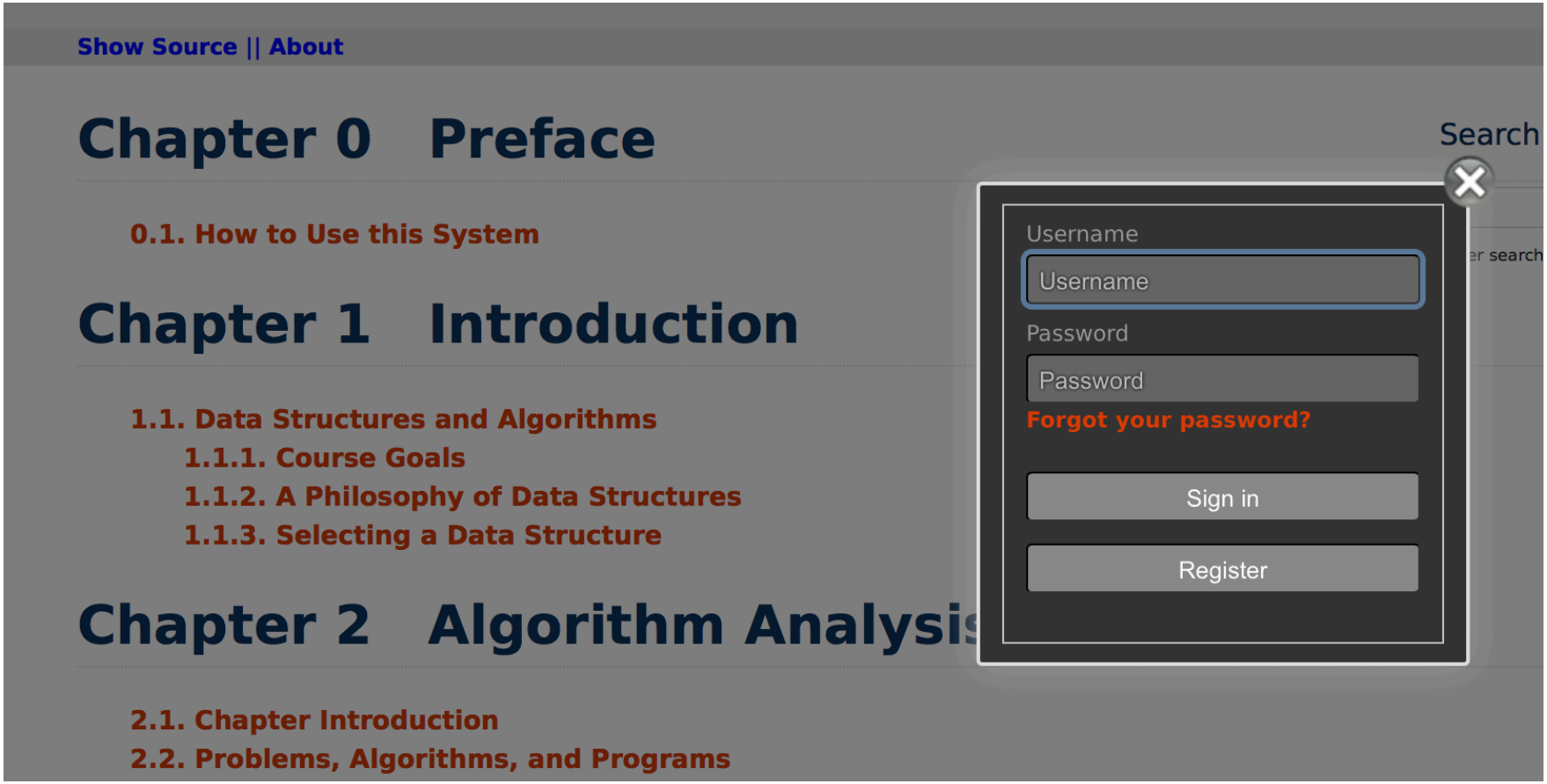
- Text och interaktiva uppgifter
- Alla uppgifter måste lösas, dvs. 100% korrekta svar för G på denna del av kursen

OpenDSA

- Årets omgång öppnar idag
- Alla resultat läses i samband med tentamen

- Slutdatum för att ha klarat OpenDSA
  - 100% av alla uppgifter korrekt besvarade
  - Deadline 30 oktober

OpenDSA


Show Source || About
Chapter 0 Preface
0.1. How to Use this System
Chapter 1 Introduction
1.1. Data Structures and Algorithms
1.1.1. Course Goals
1.1.2. A Philosophy of Data Structures
1.1.3. Selecting a Data Structure
Chapter 2 Algorithm Analysis
2.1. Chapter Introduction
2.2. Problems, Algorithms, and Programs
Search
Username
Username
Password
Password
Forgot your password?
Sign in
Register

OpenDSA

- VIKTIGT! Använd din LiU-ID som inloggning!
  - Så att dina resultat kan bli registrerade i LADOK

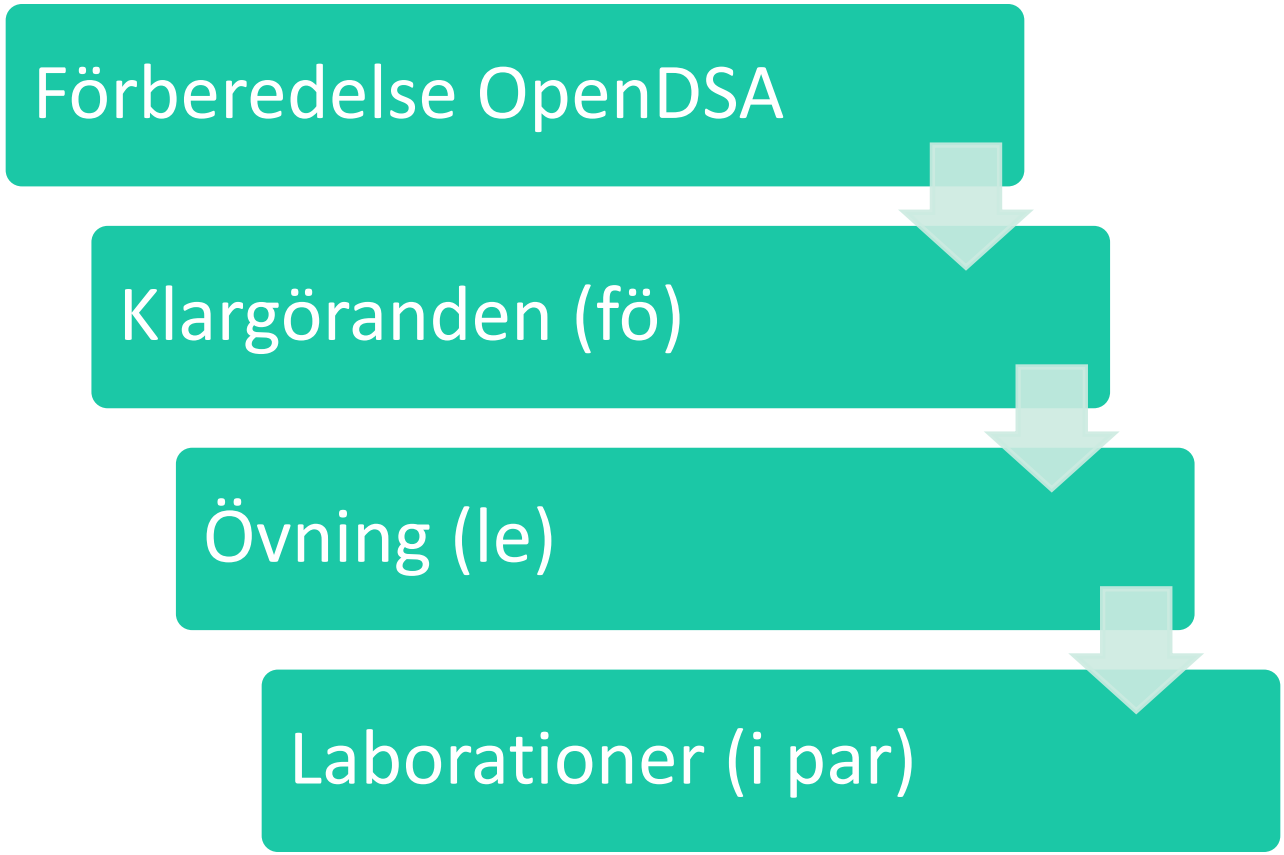
Laborationer

- Lab1
  - AVL-träd
- Lab2
  - Hashning
- Lab3
  - Heapar
- Lab4
  - Sortering

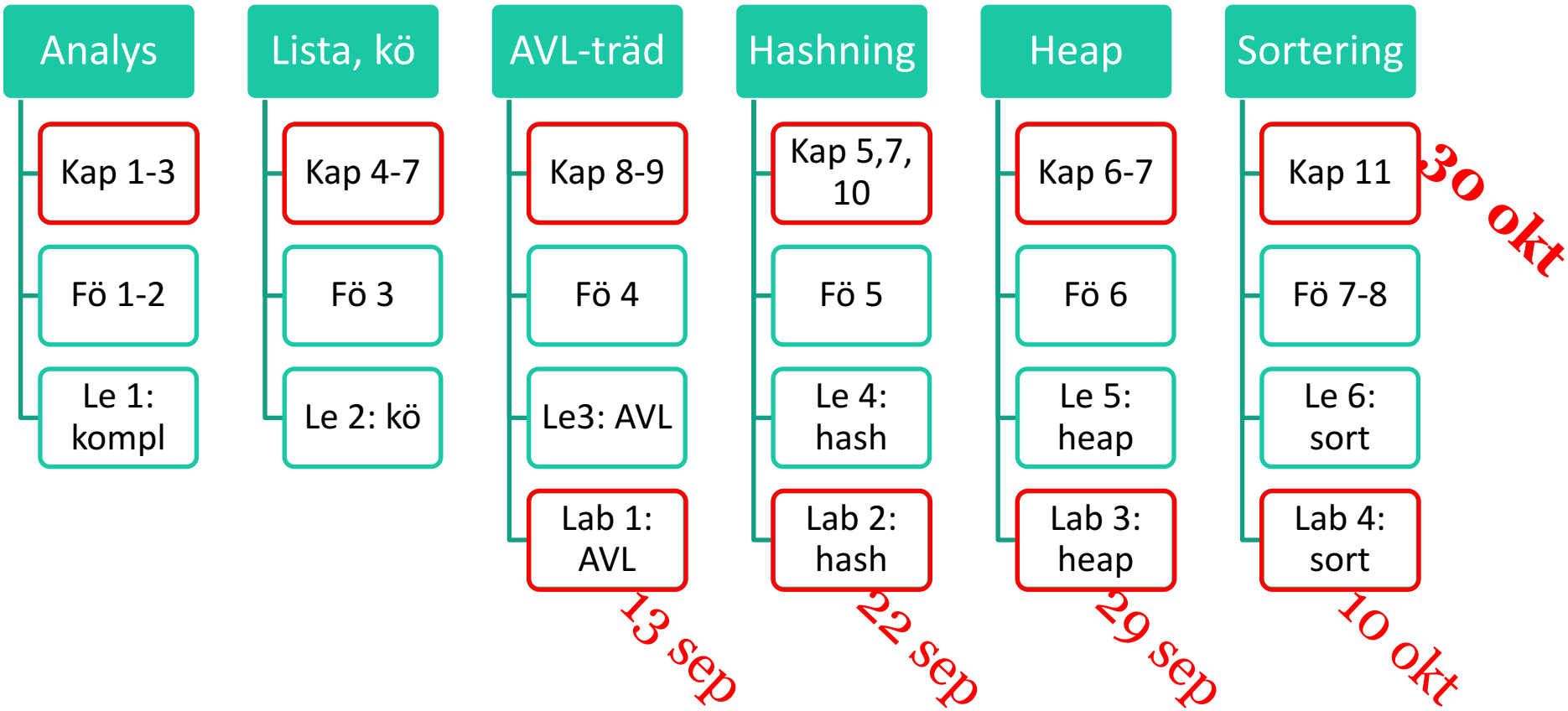
Laborationer

- Betyg U/G
- Genomförs i par (eller på egen hand)
  - Viktigt att båda är aktiva och är insatta
  - Muntlig demo + besvara frågor från assistenten
  - Automatisk testning av kod
  - Slutlig rättning av kursassistent
- Registrering av grupper i webreg (se kurshemsidan för länk)

Arbetssteg


Förberedelse OpenDSA
Klargöranden (fö)
Övning (le)
Laborationer (i par)

Kursupplägg



Tentamen

- För G: datortenta
  - Säkerställa att OpenDSA-resultaten är dina
  - Tar ca. 30-60 min
- För 4, 5: Efterföljande papperstenta
  - Sitt kvar i datorsalen, lös mer kluriga uppgifter
  - Lämnas ut som pdf-fil
  - Lämnas in på papper

Datortenta

- Målet är att säkerställa att just *du* har rätta kunskapen (från OpenDSA)
  - Urval av de övningsfrågor du mötte i OpenDSA
  - Inte tillgång till texten i OpenDSA, bara frågorna

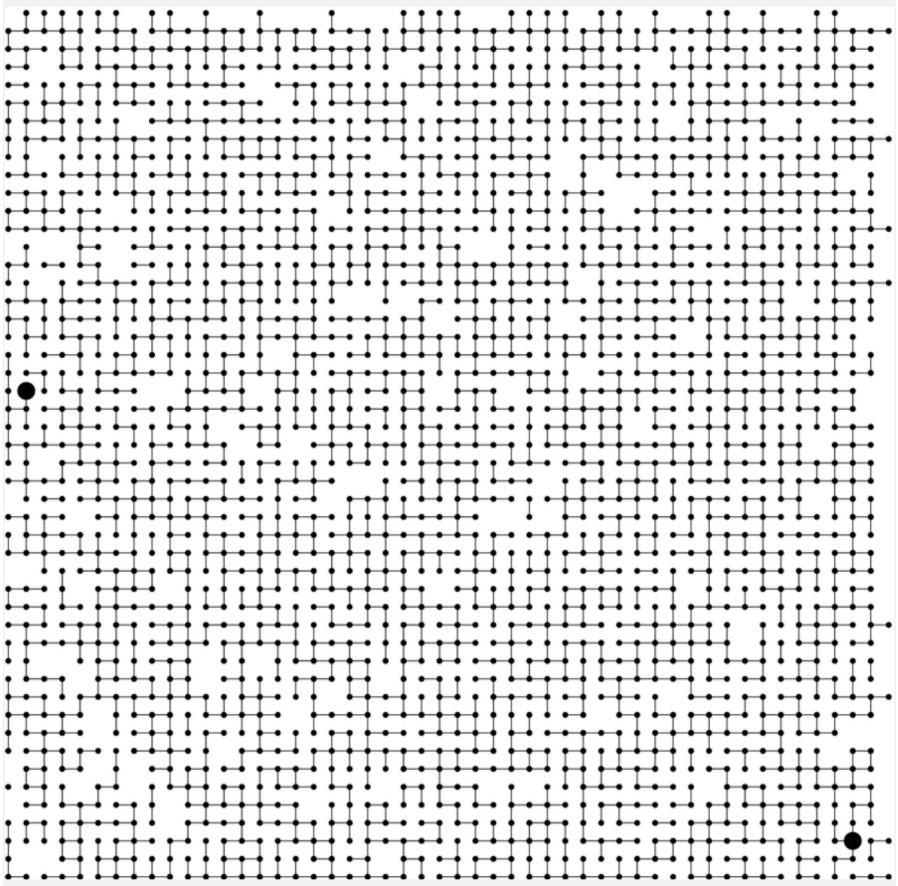
Papperstentamen

- Samma tillfälle som datortenta
  - Men, för högre betyg
  - Svårare uppgifter (levereras i pdf dokument i din tentamensdator)
- Handskrivna svar på papper
- Lämnas in till tentavakter

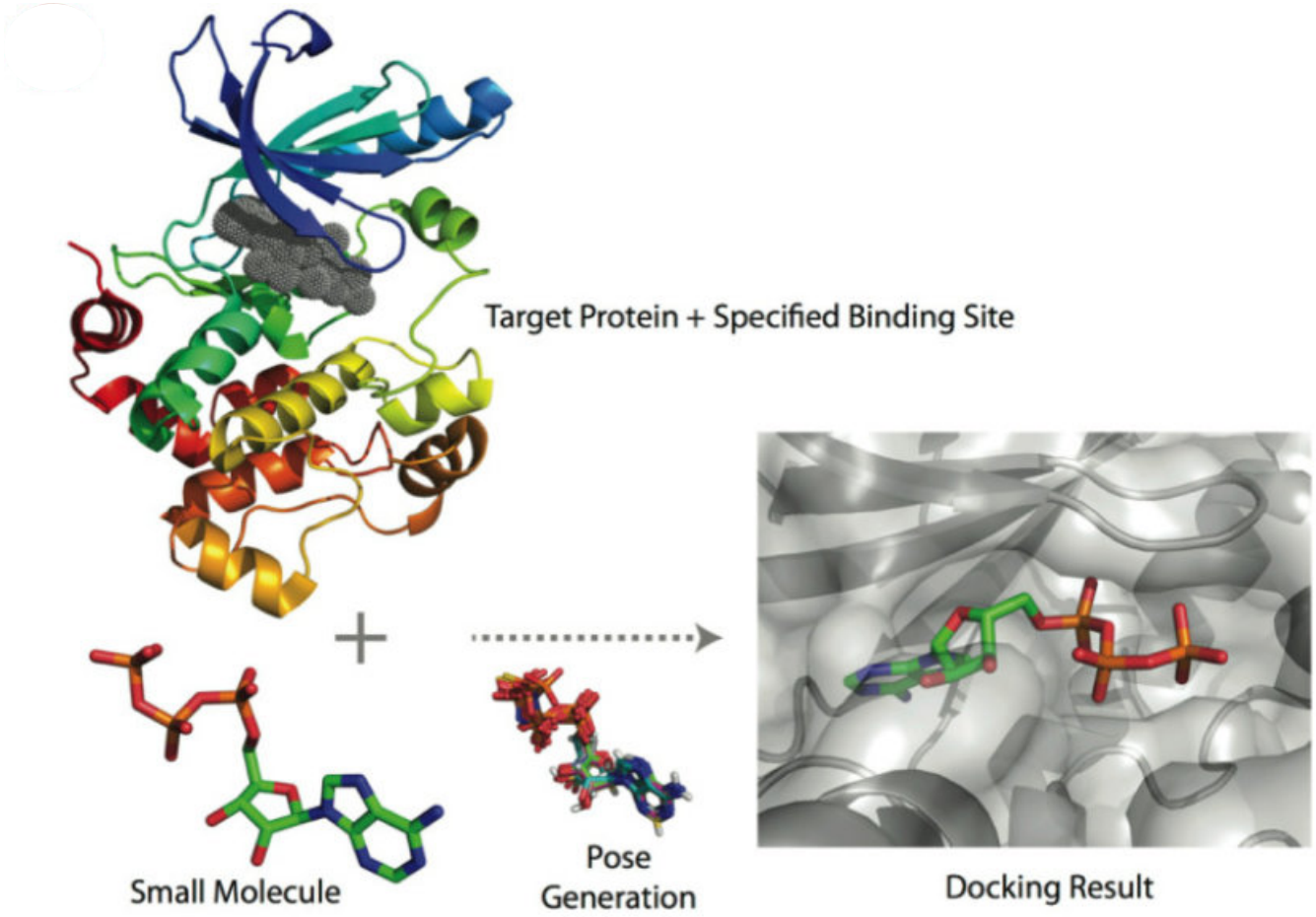
Varför ska man plugga DALG?

Lösa teoretiskt svåra problem

- Path finding



Lösa dataintensiva problem



Få intellektuell stimulans

THE THEME OF THIS FIRST-OF-THE-CENTURY ISSUE OF *COMPUTING IN SCIENCE & ENGINEERING* IS ALGORITHMS. IN FACT, WE WERE BOLD ENOUGH—AND PERHAPS FOOLISH ENOUGH—TO CALL THE 10 EXAMPLES WE'VE SELECTED "THE TOP 10 ALGORITHMS OF THE CENTURY."

"For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing."

- Francis Sullivan, Maryland Cybersecurity Center

Bli en kunnig programmerare

"…Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

– Linus Torvalds (skaparen av Linux)

Kort om datastrukturer

Abstrakta datatyper (ADT)

- Spec av datastruktur + tillhörande operationer
  - T.ex. Mängd: läggTill(E), antalElement(), …
  - **Datastruktur och operationer dolda bakom en fasad**, dvs. anroparen vet ej hur dessa gar implementerats
- Ger utrymme för olika implementationer
  - Kan byta fritt, utan att anroparens kod påverkas

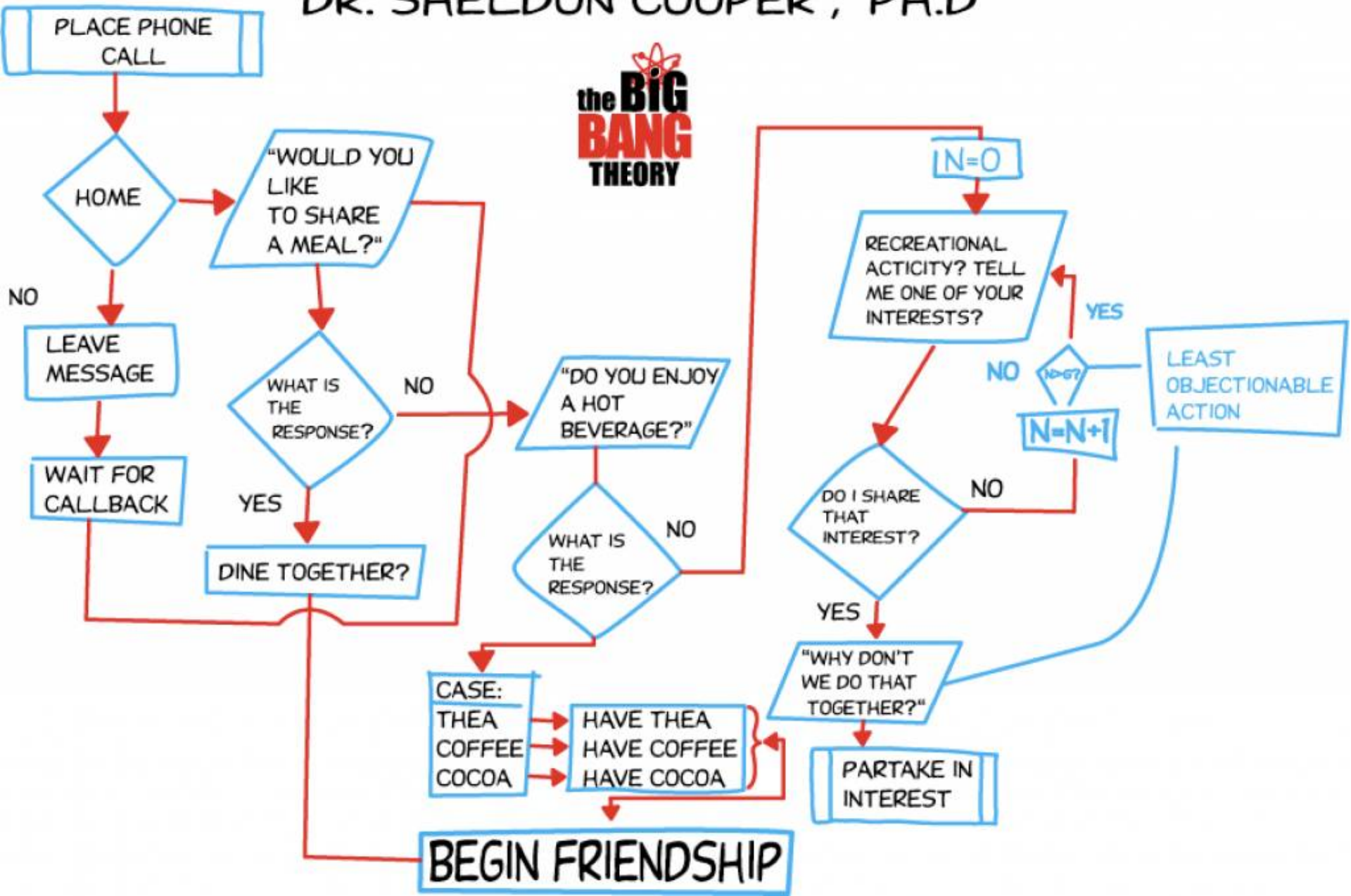
Datastruktur

- *Logisk* organisation av datorns minne för att lagra data för att underlätta åtkomst och ändring
  - T.ex. organisera i en lista
    - Logisk, dvs. behöver inte ligga direkt efter varandra i minnet
  - Lämpliga operationer: läggTill(), längd(), …

Algoritmer

THE FRIENDSHIP ALGORITHM

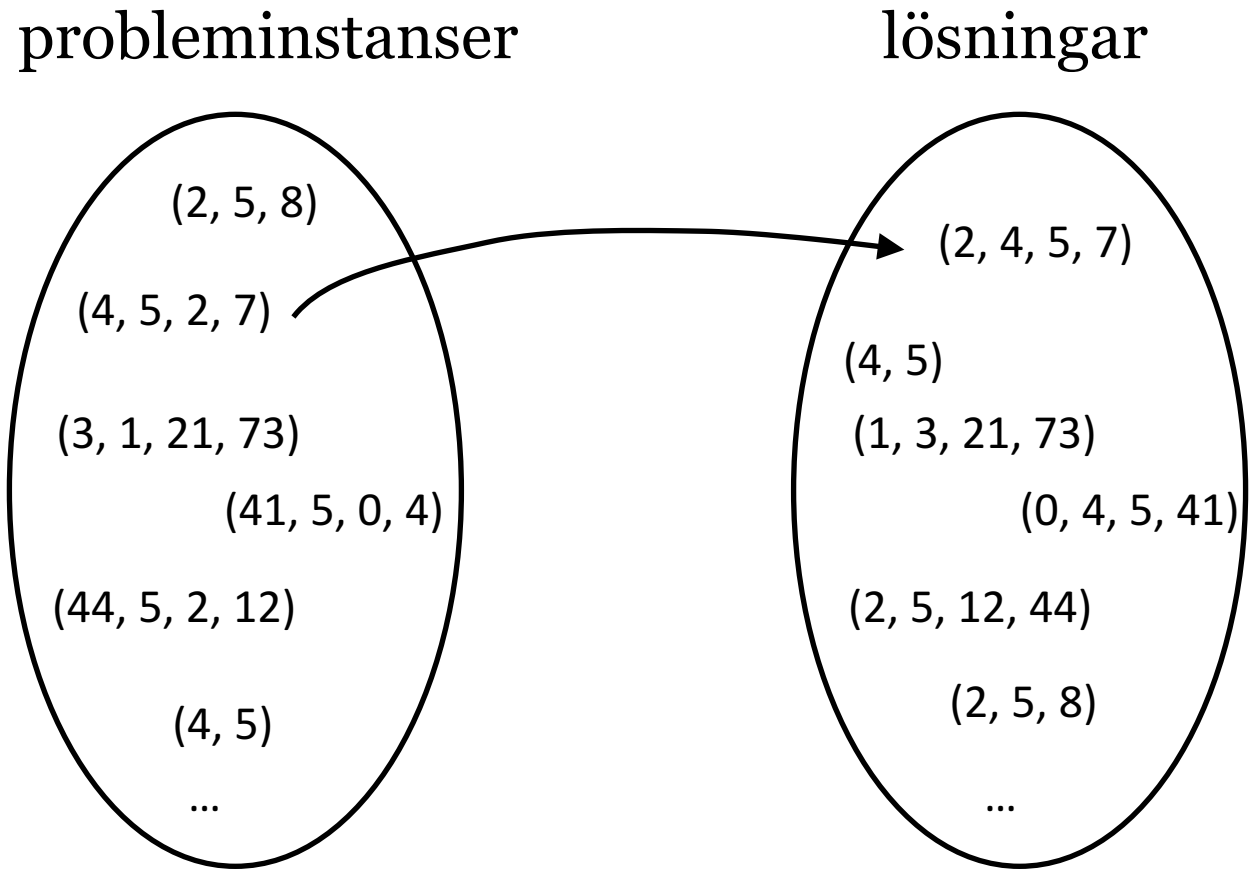
DR. SHELDON COOPER , PH.D



Algoritm

- Sekvens av väldefinierade steg som löser ett *beräkningsproblem*
  - Problem: Sortera en lista av tal
  - En *instans* av detta problem: Sortera talen (4, 5, 2, 7)

Algoritm: probleminstans → lösning



Gammalt ursprung, nya möjligheter

- Studiet av algoritmer har pågått sedan Euklides
  - Kom på algoritm för att hitta största gemensamma nämnare

- Formaliserades av Church och Turing på 1930-talet
- Vissa viktiga algoritmer upptäcktes av studenter i liknande kurser som denna!

Algoritmanalys

- Korrekthet (inte i denna kurs)
  - Ger förväntat resultat
- Terminering (inte i denna kurs)
  - Ger alltid ett svar
- *Algoritmkomplexitet*
  - Tidskomplexitet — hur lång tid tar algoritmen?
  - Minneskomplexitet — hur mycket minne använder algoritmen?

Algoritmkomplexitet

Tidskomplexitet

- Mäta tiden det tar att köra ett program som implementerar algoritmen
  - Men, tiden kan påverkas av datorns belastning, etc.
  - Skulle bli olika tider från gång till gång
- Bättre att använda ett mer robust mått, t.ex. antal beräkningssteg som utförs i algoritmen

Algoritmanalys

- Uppskattning av antal beräkningssteg
  - Approximation av faktisk tidsåtgång
    - T.ex. räknar antal varv som görs i en loop, men ej kostnad för enklare operationer i varje varv
  - Ger fel i uppskattningen med en konstant faktor, c
    - T.ex. algoritmen kör n varv för data av storlek n
      - Egentliga tiden är $n \cdot c$, där c är tiden för den addition som görs i varje varv

Stora skillnader i algoritmers effektivitet

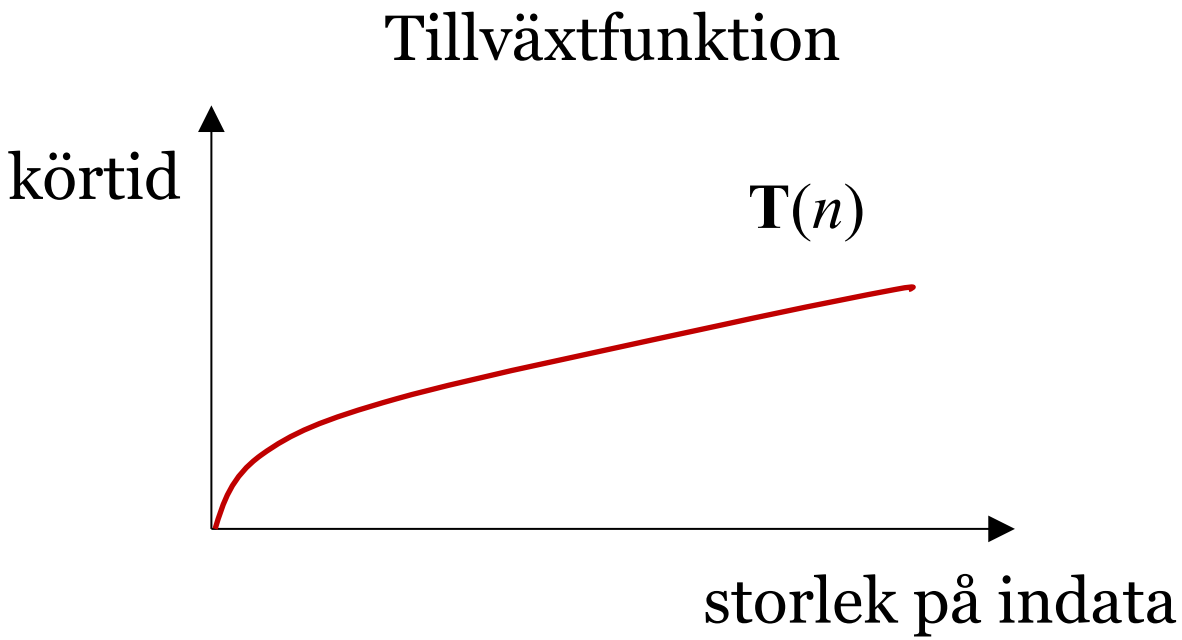
- Vissa algoritmer urartar för stora indata

n	$\log n$	n	$n \log n$	n^2	2^n
2	1	2	2	4	4
16	4	16	64	256	$6.5 \cdot 10^4$
64	6	64	384	4096	$1.84 \cdot 10^{19}$

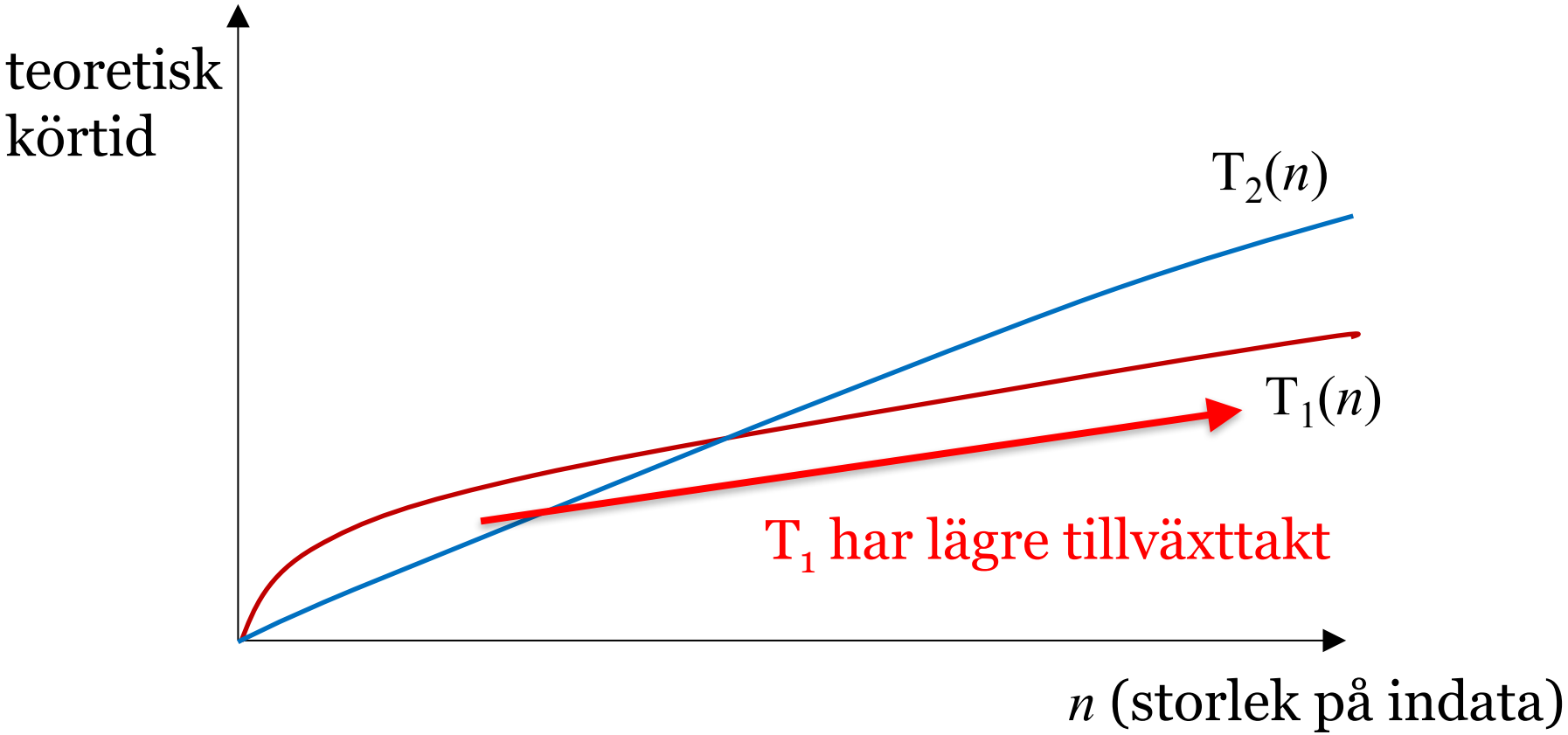
- $1.84 \cdot 10^{19}$ µs = $2.14 \cdot 10^8$ dagar = 583.5 årtusenden

Resurssåtgång beror på n (indata-storlek)

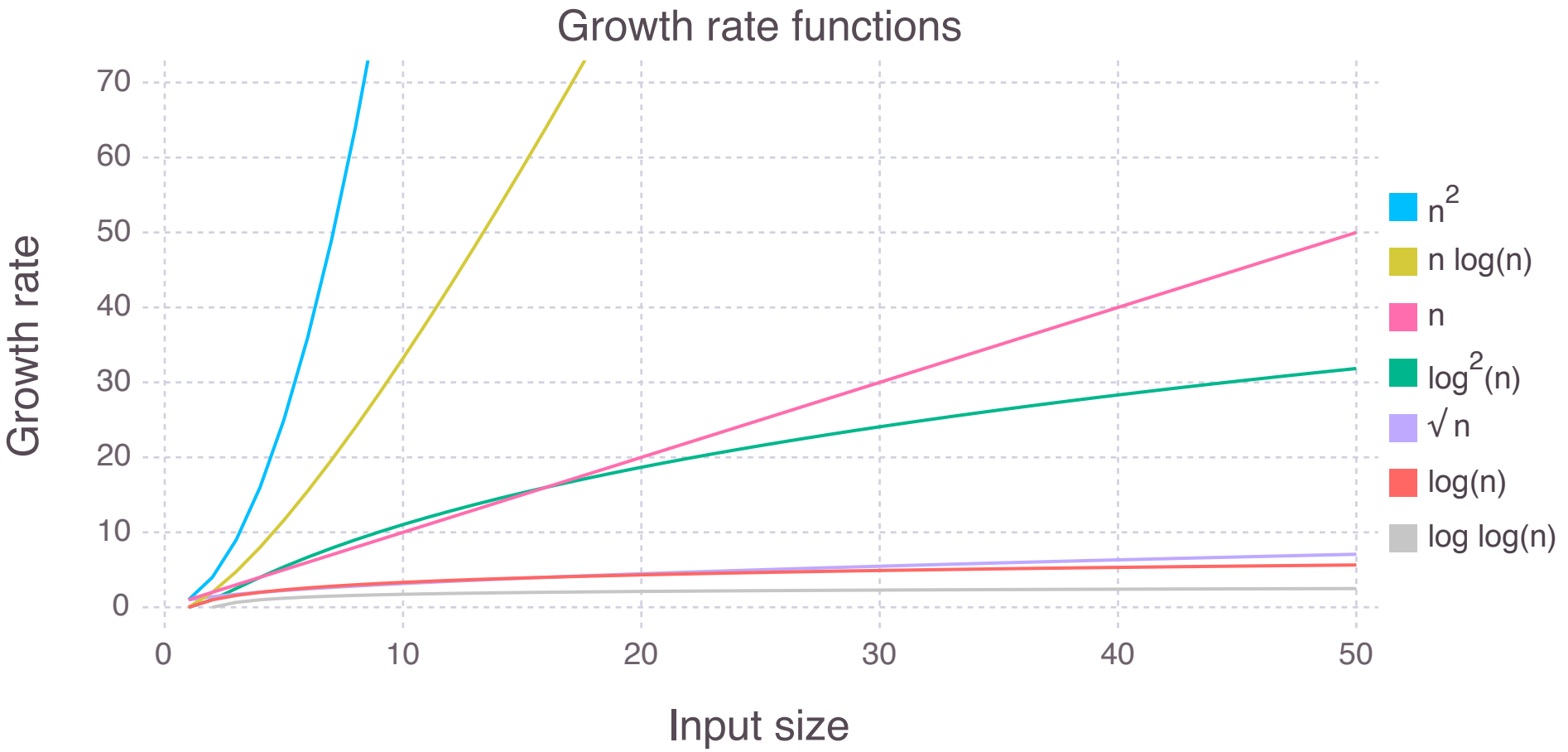
- Tillväxtfunktion $\mathbf{T}(n)$ beskriver sambandet: storlek på input → tids- eller minnesförbrukning



Två tillväxtfunktioner som beskriver två algoritmer



Vanliga tillväxtfunktioner



Ordo-notation

Ordo-notation

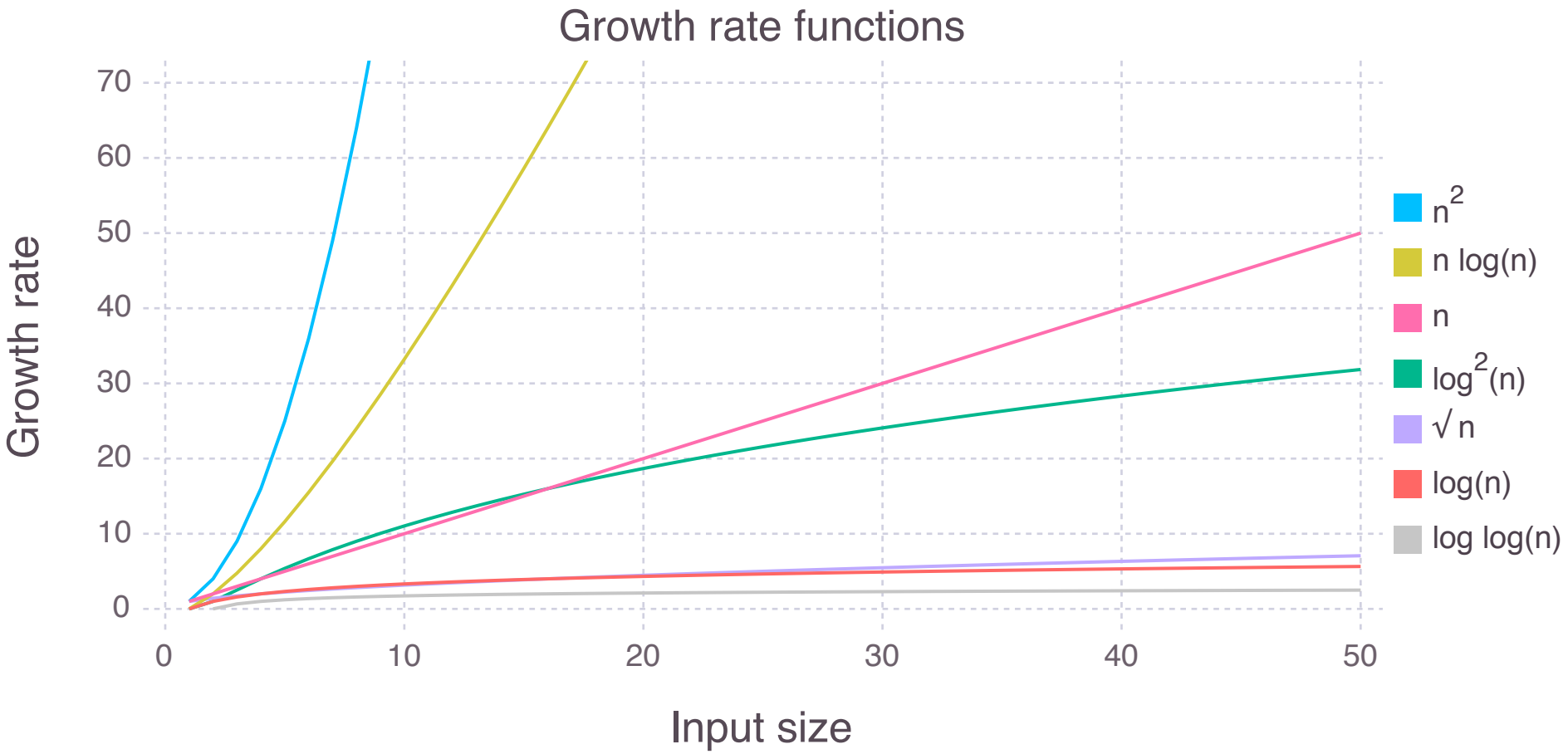
- Kan oftast inte ange *exakt* hur resursförbrukning växer för en algoritm
  - Bl.a. brukar analyser fela med en faktor c
- Vill dock kunna garantera att tillväxttakt inte är snabbare än …
  - Bortsett från en faktor c

- Ordo, $\mathcal{O}$, för att ange övre tak för tillväxttakt

Ordo: kategori av funktioner

- Eftersom vi bortser från c, kan flera tillväxtfunktioner tillhöra samma ordo-kategori
  - T.ex. har $\mathbf{T}(n)$ och $c \cdot \mathbf{T}(n)$ samma övre tak för tillväxttakt

Vanliga tillväxtfunktioner



Vanliga ordo-kategorier

- $\log \log n$
- $\log n$
- $\sqrt{n}$
- $\log^2 n$
- n
- $n \log n$
- n^2
- 2^n
- $n!$

$f(n)$

$\Longleftrightarrow$

$\mathcal{O}(\log \log n)$

$\mathcal{O}(\log n)$

...

...

$\mathcal{O}(f(n))$

Ordo-notation: definition

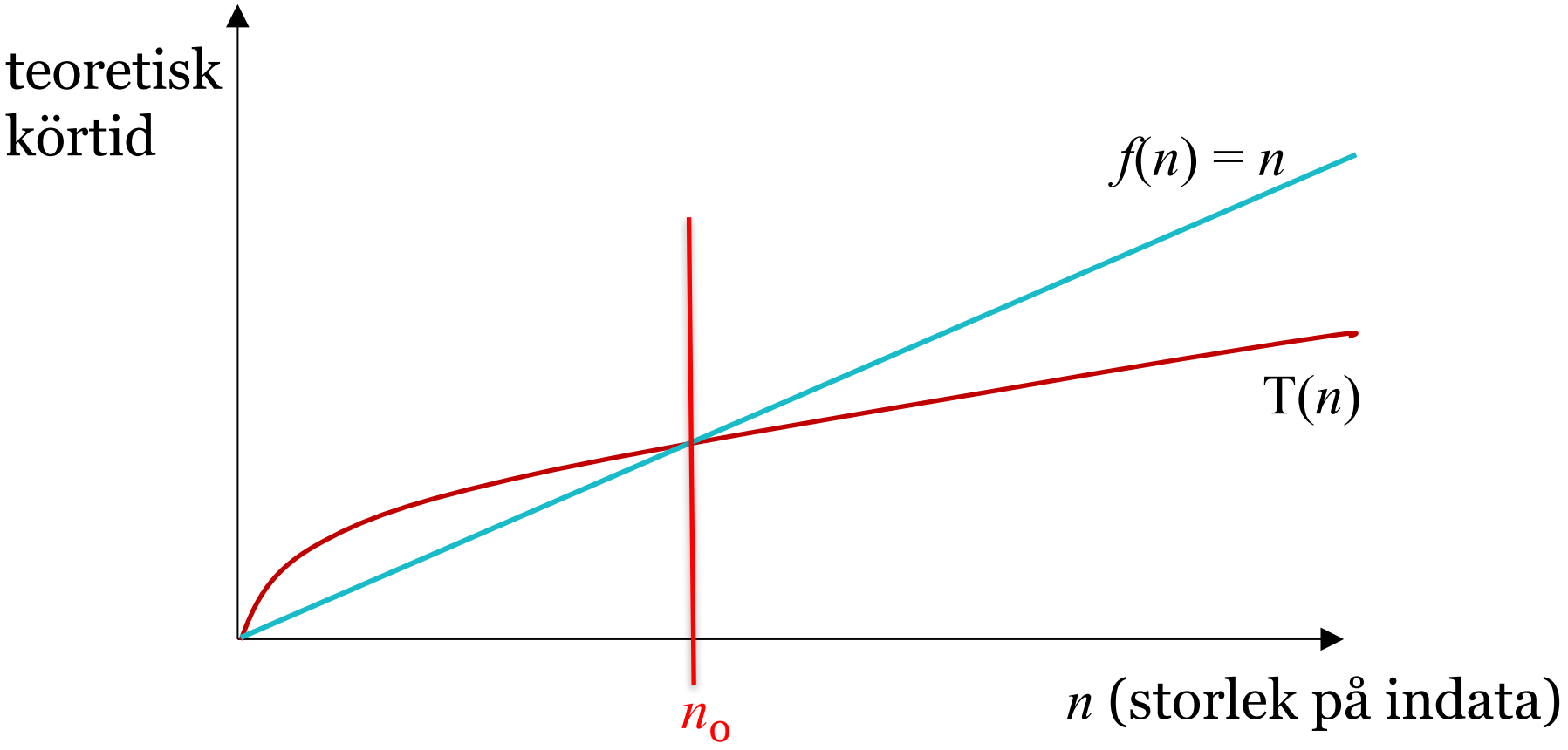
- Algoritmen med tillväxtfunktion $\mathbf{T}(n) \in \mathcal{O}(f(n))$

  omm

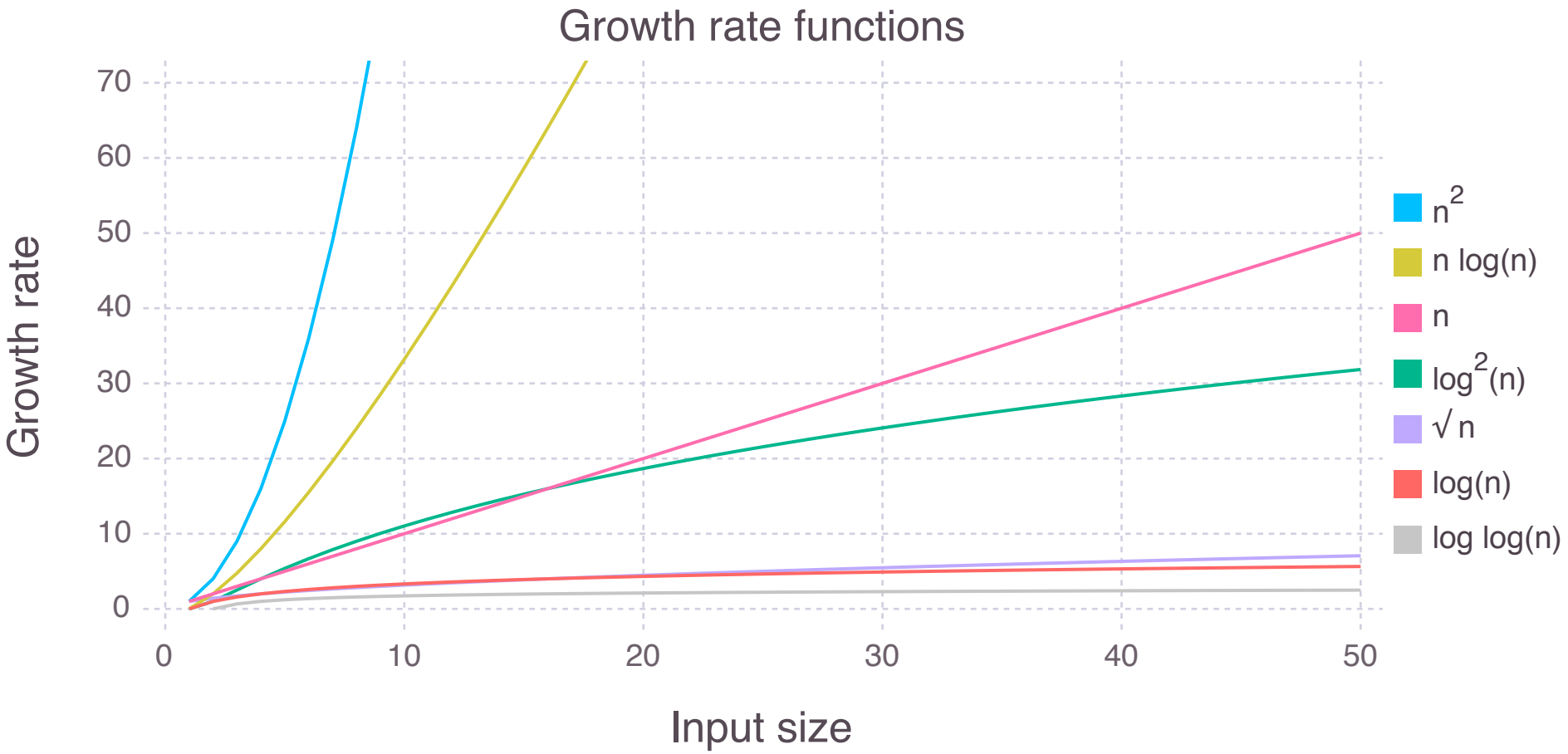
  $\mathbf{T}(n) \leq c \cdot f(n)$, för alla $n > n_0$, där c och n_0 är positiva konstanter

- Dvs. efter ett visst n_0 ska $\mathbf{T}(n)$ ligga under $f(n)$, bortsett från faktor c

Exempel: $\mathrm{T}(n)$ tillhör $\mathcal{O}(f(n))$



Vanliga tillväxtfunktioner



Ordo är transitiv

- Om $\mathbf{T}(n) \in \mathcal{O}(f(n))$ och $f(n) \in \mathcal{O}(g(n))$, så $\mathbf{T}(n) \in \mathcal{O}(g(n))$
- Dock, eftersträvar man att lägga $\mathcal{O}$-taket så lågt som möjligt

Ω golv för tillväxttakt: definition

- Algoritmen med tillväxtfunktion $\mathbf{T}(n) \in \Omega(f(n))$

  omm

  $\mathbf{T}(n) \geq c \cdot f(n)$, för alla $n > n_0$, där c och n_0 är positiva konstanter

- Dvs. efter ett visst n_0 ska $\mathbf{T}(n)$ ligga över $f(n)$, bortsett från faktor c

Θ "teta": definition

- Algoritmen med tillväxtfunktion $\mathbf{T}(n) \in \Theta(f(n))$

  omm

  $\mathbf{T}(n) \in \mathcal{O}(f(n))$ och $\mathbf{T}(n) \in \Omega(f(n))$

- Dvs. $\mathbf{T}(n)$ och $f(n)$ ligger inom samma Ordo-kategori, bortsett från faktor c

Nästa gång: Mer ingående algoritmanalys

www.liu.se

LINKÖPINGS UNIVERSITET