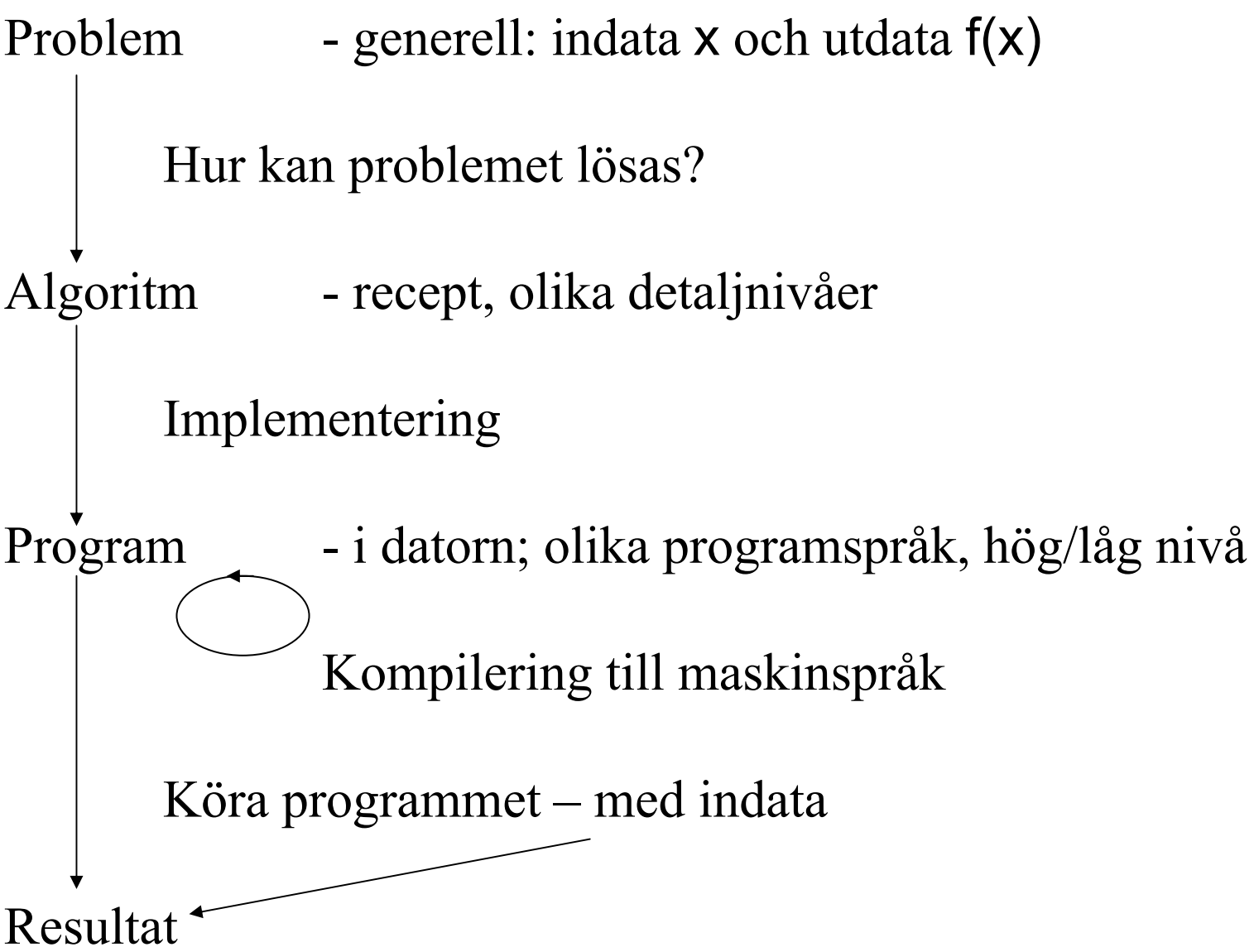
Teori för beräkningar, Grundläggande programmering STS, 6 maj 2003.

Beräkningar = Problemlösning, inte bara räkna med tal!

Men:
all data i datorn representeras som en sekvens av bitar,
och varje sekvens av bitar kan tolkas som ett tal.


Problem
- generell: indata x och utdata f(x)
Hur kan problemet lösas?
Algoritm
- recept, olika detaljnivåer
Implementering
Program
- i datorn; olika programspråk, hög/låg nivå
Kompilering till maskinspråk
Köra programmet – med indata
Resultat

Problemlösning

Dela upp problemet i

- enklare problem
- samma problem, fast mindre

Exempel.

Problem: räkna ut antal dagar mellan datum1 och datum2.

Lösning: räkna ut dagnummer(datum1)
räkna ut dagnummer(datum2)
ta skillnaden.

Enklare!

Exempel: olika algoritmer för ett problem.

Problem: hitta ett namn i en lista med **n** namn.

Lösningar:

1. Linjär sökning: är det första namnet?
   - Ja: hittat!
   - Nej: sök i resten - **mindre problem (n-1)**

   Basfall: tom lista - namnet finns inte.

Om listan är sorterad:

2. Binär sökning: jämför med mellersta namnet
   - =: hittat!
   - < sök i första halvan **- mindre (n/2)**
   - > sök i andra halvan

   Basfall: tom lista - namnet finns inte.

Fråga: hur söker vi i en telefonkatalog?!

Algoritm - Program

Exempel

Lektionsuppgiften:
- läsa indata: tal n och lista L
- skriva ut listan
- ta bort alla n ur L
- skriva ut listan igen

Ett program i Java (lektionen).
Ett program i Prolog (visas).

Olika program
samma algoritm

Ser ganska olika ut.
Gör ”samma sak” - på en abstrakt nivå - samma algoritm

```
main :-
   write('Ge det tal som ska bort'), nl,
   read(Bort),
   in(Tabell),
   write('Talen före'), nl,
   skriv(Tabell),
   remove(Bort, Tabell, NyTabell),
   write('Talen efter'), nl,
   skriv(NyTabell).

in(Tabell) :-
   write('Ge ett antal tal. Avsluta med 0.'), nl,
   read(Tal),
   in2(Tal, Tabell).

in2(Tal, [Tal|Tabell]) :-
   Tal > 0,
   write('Ge nästa tal'), nl,
   read(Nästa),
   in2(Nästa, Tabell).
in2(0, []).

skriv([Tal|Tabell]) :-
   write(Tal),
   write(' '),
   skriv(Tabell).
skriv([]) :-
   nl.

remove(Tal, [Tal|Tabell], NyTabell) :-
   !,
   remove(Tal, Tabell, NyTabell).
remove(Tal, [Annat|Tabell], [Annat|NyTabell]) :-
   remove(Tal, Tabell, NyTabell).
remove(_, [], []).
```

Jämföra program/algoritmer

Kriterier:

- rätt (så klart)
- enkelt
- lätt att ändra, utöka
- snabbt

beror på

• datorn/hårdvara	konstant faktor
• programspråk, kompilator	konstant faktor (oftast)
• **algoritm**	
• problemets storlek	ingångsvärde

Komplexitet

Hur många **steg** kräver algoritmen?

- i bästa fall - händer sällan
- i genomsnitt - vad är ”genomsnittlig indata”?
- **i värsta fall** - liknar oftast genomsnittsfallet.

En funktion av indatas storlek n, t.ex. $2\ n \log(n) + 3\ n - 5$

Vi bortser ofta från konstanta faktorer och mindre viktiga termer:
O(n log(n)) - ordo n log n

Exempel.
Problem: hitta ett namn i en lista med **n** namn.
Lösningar:

1. Linjär sökning: är det första namnet?
   Ja: hittat!
   Nej: sök i resten - **mindre problem (n-1)**

Basfall: tom lista - namnet finns inte.

Komplexitet:
Bästa fall: 1.
Värsta fall: n.
Genomsnitt: n/2 om namnet finns, n om namnet inte finns.

O(n)

Om listan är sorterad:

2. Binär sökning: jämför med mellersta namnet

=: hittat!

< sök i första halvan **- mindre (n/2)**

> sök i andra halvan

Basfall: tom lista - namnet finns inte.

Bästa fall: 1.

Värsta fall: f(n) = 1 + f(n/2) och f(1) = 1.

Lösning: f(n) = 1 + log(n)

Genomsnitt (om namnet finns): ca. log(n) - exempel:

4 3 4 2 4 3 4 1 4 3 4 2 4 3 4 49/15

Sortera en bunt tentor

- ta bort 1 tenta
- sortera n-1 tentor
- stoppa in den borttagna tentan

Basfall: sortera 1 tenta - klart!

Komplexitet:
$f(n) = f(n-1) + 1 + \log(n-1)$ med $f(1) = 0$.
Lösning:
$f(n) = 1 + \log(n-1) + \ldots + 1 + \log(1) < (n-1)(1+\log(n))$

O(n log(n))

Sortera en bunt tentor

- dela bunten i två jämnstora delar
- sortera första halvan
- sortera andra halvan
- sammanfoga

Komplexitet:
$f(n) = f(n/2) + f(n/2) + n$
Lösning: $f(n) = n \log(n)$ eftersom
$n \log(n) = n/2 \log(n/2) + n/2 \log(n/2) + n = n(\log(n)-1) + n$

O(n log(n))

Komplexitet av ett problem

Är komplexiteten av den bästa algoritmen som löser problemet.

Hur vet vi att vi har hittat bästa algoritmen?

Övre gräns - känd algoritm
Undre gräns - analys av problemet.
Exempel: sortering.
Det finns n! permutationer - kräver log(n!) jämförelser.

Undantag:
- om vi har extra information kan vi få ner genomsnittet
- om vi kan göra del av en jämförelse, t.ex. del av namn

Exempel övre/undre gräns

Multiplicera nxn matriser:

standard algoritm	n^3
alternativ	$n^{\log 7}$ - stor konstant faktor
senaste	$n^{2.4\ldots}$
Känd undre gräns:	n^2 (matris x vektor)

Hitta kortaste rundtur som angör n städer:

kända algoritmer	exponentiell	2^n
känd undre gräns	polynom	n^c

P = NP?

Finns det en algoritm för varje problem?

Smuts behöver tvål – inte någon algoritm!

alltså: hur definierar vi *problem*?

Eftersom all data kan tolkas som ett heltal:

en funktion från heltal (indata) till heltal (utdata).

Hur många program finns?

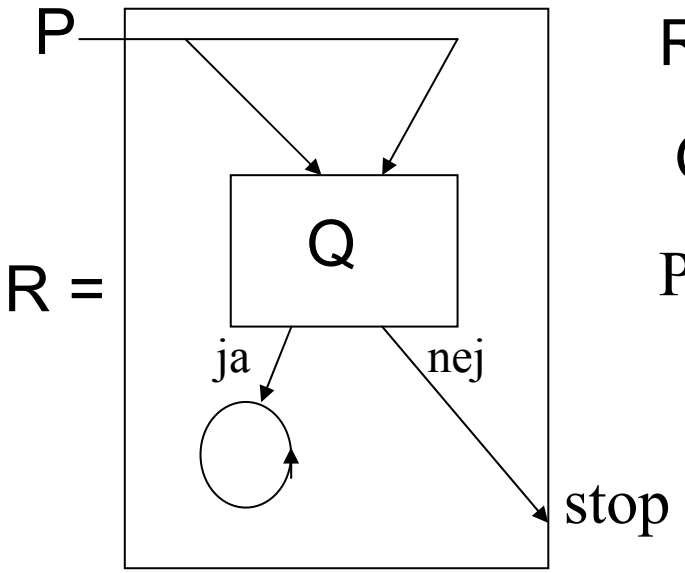
program = data = heltal $\mathbb{N}$

problem = funktion $\mathbb{N} \rightarrow \mathbb{N}$

Ett konkret exempel - stopproblemet

Problem: stannar program P med indata n?

Tänk om det fanns ett program/algoritm Q(P,n) som svarar ja eller nej på frågan.



R(R) stannar ↔

Q(R, R) svarar nej ↔ (vad gör Q?)

Program R stannar inte med indata R

Liknande problem

- Program P stannar för all indata.
- Programmen P och Q gör detsamma för all indata.
- Program P är korrekt (given en specifikation)
- (Rice) Program P har egenskap φ
  (där vissa, inte alla, program har egenskap φ)
- Ta bort kod som aldrig körs.

Det finns ett program som beter sig rätt på all testdata,
men fel på all annan indata.

Litteratur

Goldschlager & Lister

Computer Science – A Modern Introduction

2nd edition, 1988(!)

Avsnitt

1.1, 1.2, 1.4, 2.1: algoritmer och program

2.3, 2.9: problemlösning

3.2.1, 3.2.2, 3.2.4: komplexitet

3.1.2 – 3.1.4: problem som inte kan beräknas