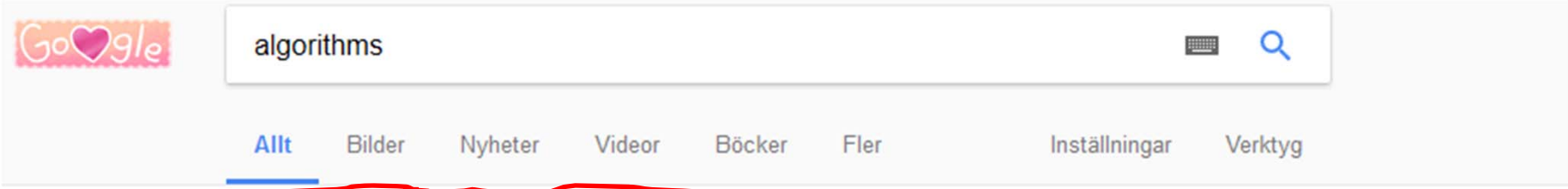
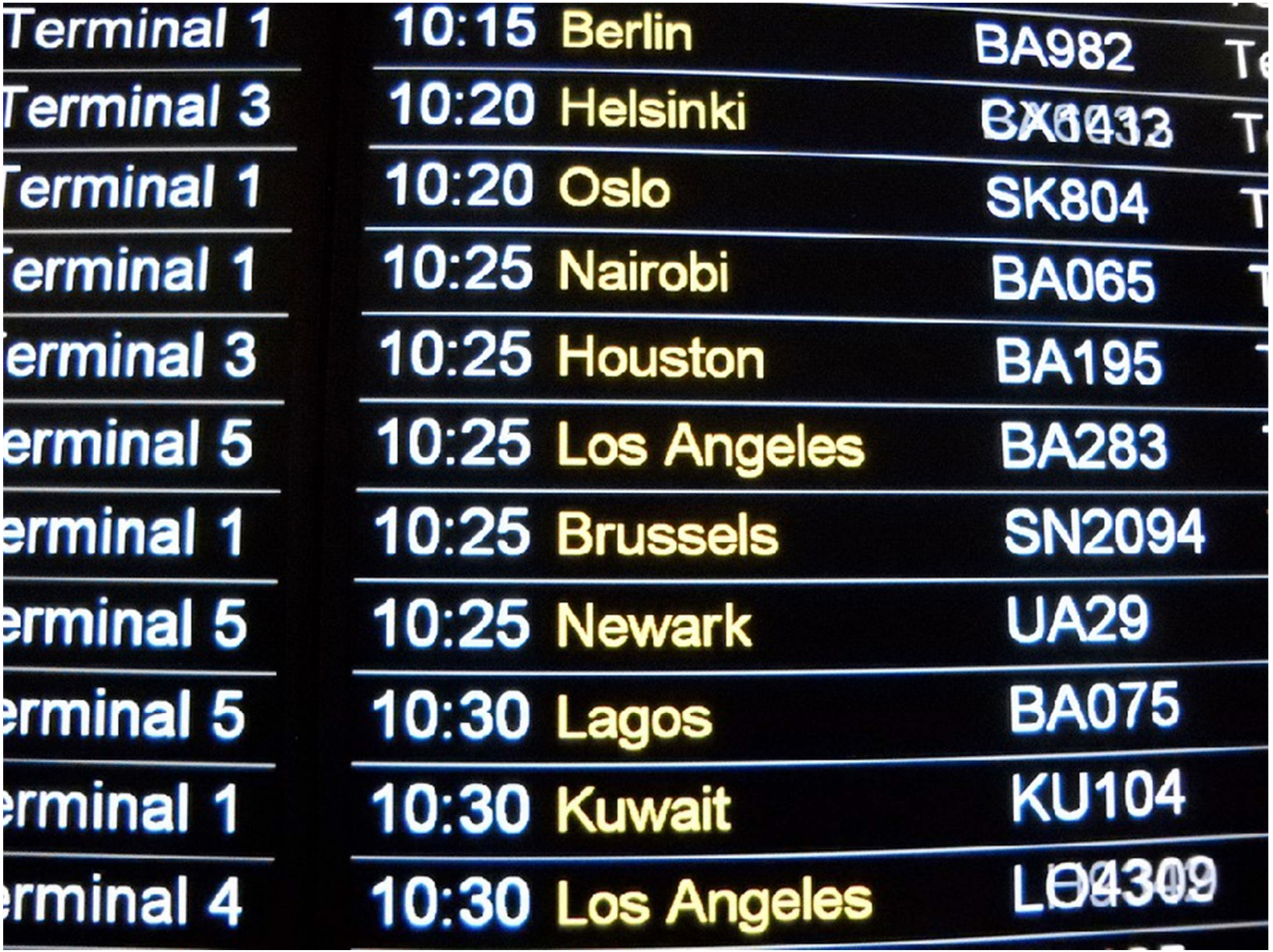
Algoritmer och datastrukturer TDA143

2017-02-15

Uno Holmer


algorithms
Allt
Bilder
Nyheter
Videor
Böcker
Fler
Inställningar
Verktyg
Ungefär 91 500 000 resultat (0,38 sekunder)
Algorithm - Wikipedia
https://en.wikipedia.org/wiki/Algorithm ▾ Översätt den här sidan
In mathematics and computer science, an algorithm is a self-contained sequence of actions to be performed. Algorithms perform calculation, data processing, ...
What is an algorithm and why should you care? (video) | Khan Academy
https://www.khanacademy.org/.../algorithms/...algorithms/.../what... ▾
17 nov. 2014
An algorithm is just a set of steps as explained. You have to learn how to make algorithms using pseudo-code ...
5:27
Algorithms | Computer science | Computing | Khan Academy
https://www.khanacademy.org/computing/computer.../algorith... ▾ Översätt den här sidan
We've partnered with Dartmouth college professors Tom Cormen and Devin Balkcom to teach introductory computer science algorithms, including searching, ...
Intro to algorithms · Running time of binary search · Asymptotic notation · Recursion
What is algorithm? - Definition from WhatIs.com
whatis.techtarget.com › Topics › AppDev › Programming ▾ Översätt den här sidan
This definition explains what algorithms are and how they work. We discuss types of algorithms including search algorithms and encryption algorithms and ...
Algorithms - Stanford University | Coursera
https://www.coursera.org/specializations/algorithms
Algorithms from Stanford University. Algorithms are the heart of computer science, and the subject has countless practical applications as well as ...


Terminal 1 10:15 Berlin BA982
Terminal 3 10:20 Helsinki
Terminal 1 10:20 Oslo SK804
Terminal 1 10:25 Nairobi BA065
Terminal 3 10:25 Houston BA195
Terminal 5 10:25 Los Angeles BA283
Terminal 1 10:25 Brussels SN2094
Terminal 5 10:25 Newark UA29
Terminal 5 10:30 Lagos BA075
Terminal 1 10:30 Kuwait KU104
Terminal 4 10:30 Los Angeles

Algoritm

Informell beskrivning:

Ett antal steg som beskriver hur en uppgift utförs.

Formell beskrivning:

En algoritm är en ordnad samling entydiga, utförbara steg som beskriver en terminerande process.

Brookshear: Computer Science, An overview

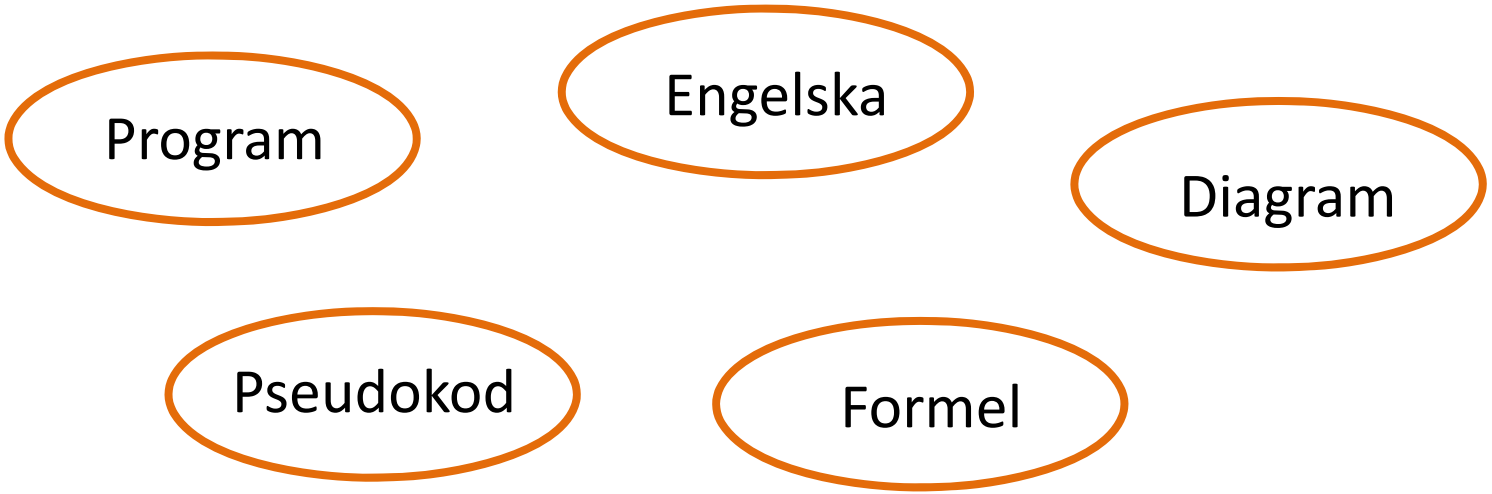
Exempel: Sortering

1. Antag första talet “sorterat” 5 2 6 1 3
2. Hitta första osorterade tal 5 6 1 3
   2
3. Hitta rätt plats i sorterad del 2 5 6 1 3
4. Upprepa med resten av listan 2 5 1 3
   6
   …

Algoritmer & representationer

Algoritm: abstrakt idé för att lösa ett problem

Representation: formulering av den abstrakta idén som till exempel:



Ex. Omvandla Celsius till Fahrenheit

Tre representationer:

- "Multiply the temperature in Celsius by 9/5 and add 32 to the product"
- F = 9/5*C + 32
- ```
  function fahrenheit(c)  {
      return 9/5*c + 32;
  }
  ```
  ```

Liknelse: bakning

Algoritm	Idé om hur man bakar äpplepaj
Representation	Recept
Implementering	Bageri

Pseudokod

- Vanligaste representationen
- Hybrid mellan naturligt språk och programspråk

procedure *SeqSearch* (List, Value)

    while (entries left to be considered)

        do TestEntry ← next entry from List

            if (Value = TestEntry) **then return** “success”

    return “search failed”

Analys av algoritmer

Korrekthet

- Löser algoritmen problemet?
  - Ger den rätt resultat?
  - Terminerar den?

Effektivitet

- Hur lång tid tar den att exekvera?
  - Är den skalbar: hur ökar exekveringstiden med storleken på indata?

Designstrategier för algoritmer

- Top Down/Bottom Up
- Uttömmande sökning (brute-force)
- Giriga algoritmer (greedy)
- Söndra och härska (divide-and-conquer)
- Dynamisk programmering
- Heuristiker (t.ex. inom AI)
- …

Tidskomplexitet

- Algoritmers körtid mäts i *antal operationer:*
  - *additioner*
  - *tilldelningar*
  - *jämförelser*
  - …

  Inte i sekunder! *Varför?*
- För en given indatastorlek (t.ex. antal element i en lista), hur många operationer utförs?

$$\mathrm{T}(n) = \mathrm{f}(n)$$

Exempel: SeqSearch

```
procedure SeqSearch (List, Value)
    while ( entries left to be considered)
        do TestEntry ← next entry from List
            if ( Value = TestEntry ) then return “success”
    return “search failed”
```

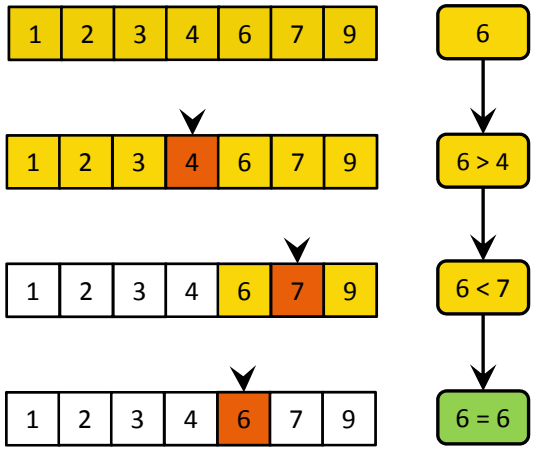
- Om listan har *n* element behövs (i värsta fall)
  - *n* uppslag
  - *n* tilldelningar
  - *n* jämförelser
  - *1* returnering

T(*n*) = 3*n* + 1 operationer!

Kan det göras snabbare?

Binärsökning

- Om listan är sorterad



procedure *BinSearch* (List, Value)

if(List isEmpty)

return “fail”

else

TestEntry ← middle entry of List

if(Value = TestEntry) **return** “success”

if(Value > TestEntry) **return BinSearch**(RightHalfofList, Value)

if(Value < TestEntry) **return BinSearch**(LeftHalfofList, Value)

Binärsökning (forts.)

- Antalet halveringar av n saker = $log_2 n$
- Binärsökning kräver bara $5\ log_2 n + 2$ operationer i värsta fallet

Asymptotisk komplexitet
- Ordo (Big-O)

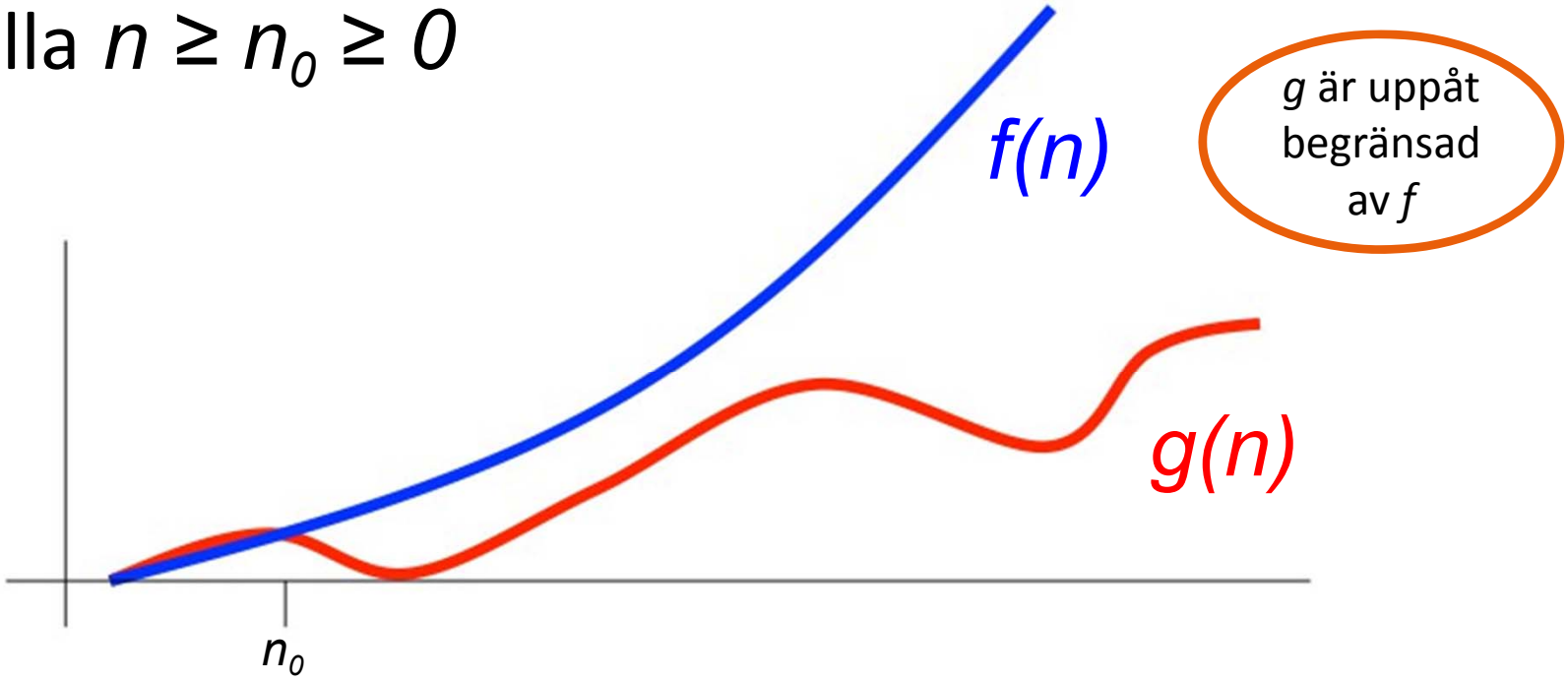
- Vanligen räknar man inte exakta antal. Istället beräknar man den *asymptotiska komplexiteten - hur antalet operationer växer med storleken på indata!*
- Exempel: Om vi dubblar antalet element, hur mycket längre tid tar det att sortera dem?

Ordobegreppet: Big-O

En funktion $g(n)$ är $O(f(n))$ om det finns en konstant $c>0$ så att

$$g(n) \leq c\, f(n)$$

för alla $n \geq n_0 \geq 0$



Insättningssortering

procedure ***InsertionSort***(List)

    N ← 2

while (N ≤ LengthOfList) **do**

    Select the N:th entry in the List as the pivot

    Move pivot to a temporary location leaving a hole

    while((exists entry left of the hole) **and** (entry > pivot) **do**

        Move the entry down into the hole

    Move the pivot into the remaining hole

    N ← N + 1

Sortering: Tidskomplexitet

- Insättningssortering, *grov* analys
  - Två nästlade for-loopar med som mest n iterationer

    => $T(n) = O(n^2)$ operationer

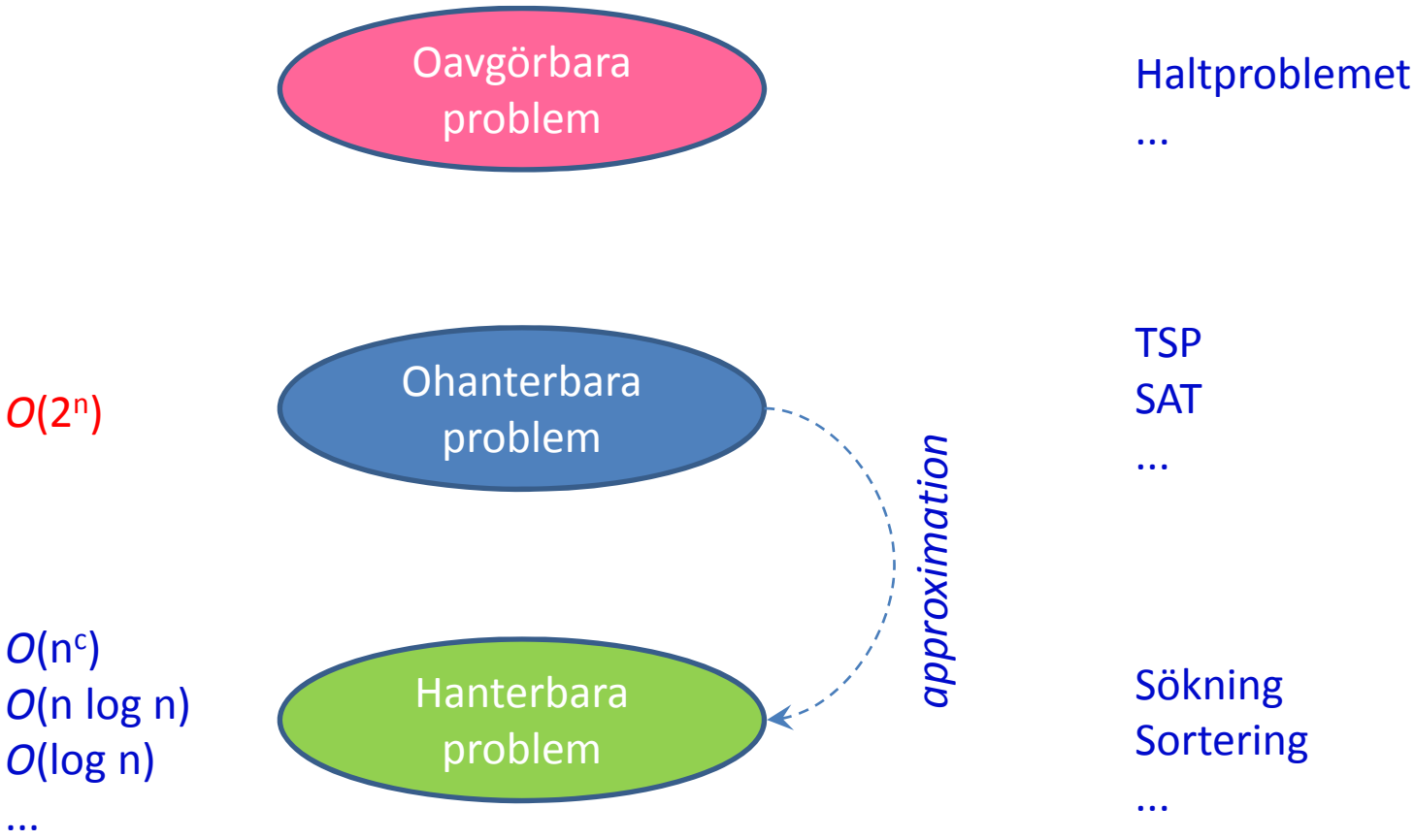
- Merge sort: $T(n) = O(n \log n)$

- *Hur många operationer behövs för att sortera 2n element jämfört med n element med respektive algoritm?*
- *T(2n) = ...*

Lätta och svåra problem

- Vissa problem är *olösbara (oavgörbara = undecidable)*
- och vissa är lösbara - men *hopplöst svåra* (*ohanterbara = intractable*)
  - De enda kända algoritmerna har exponentiell tidskomplexitet: T(n) = $O(2^n)$, eller sämre.

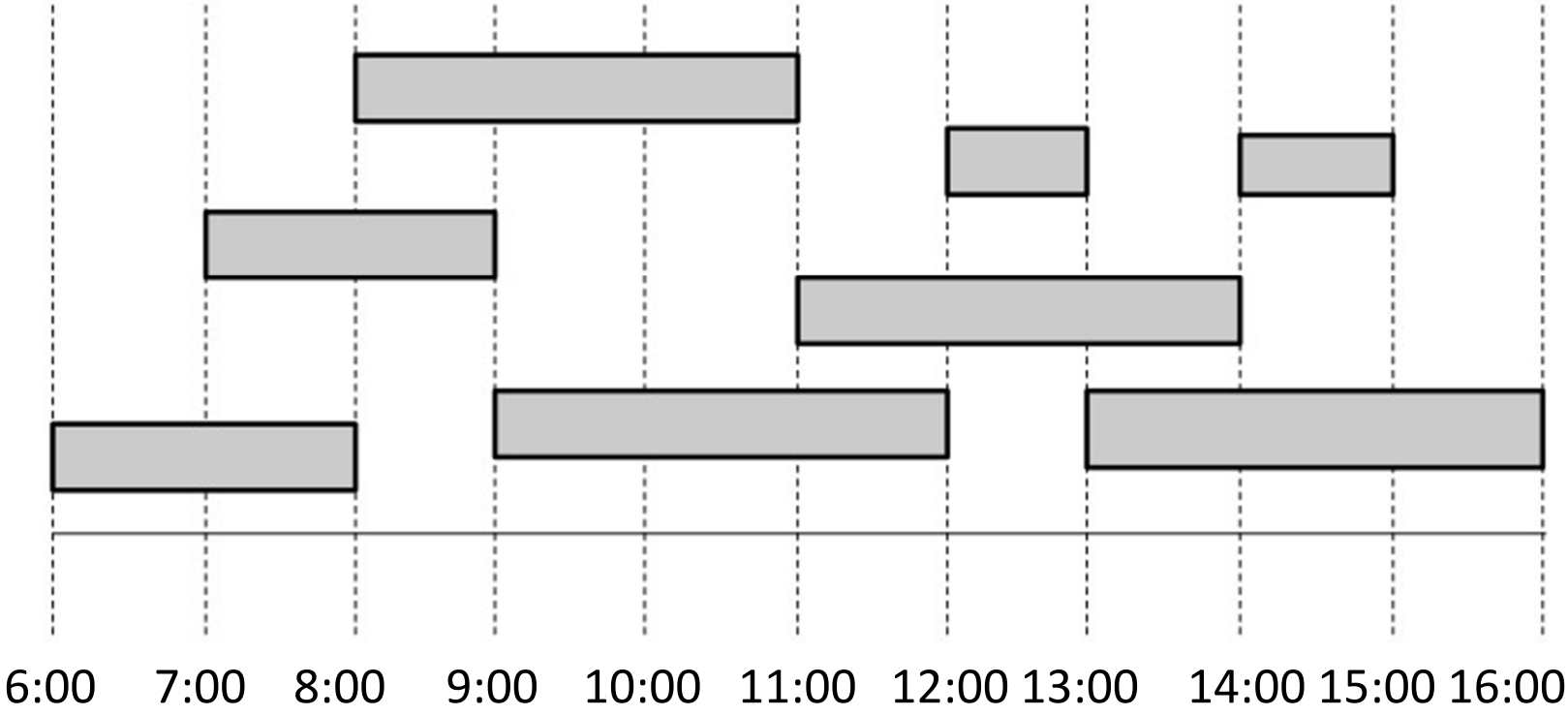
Lätta och svåra problem



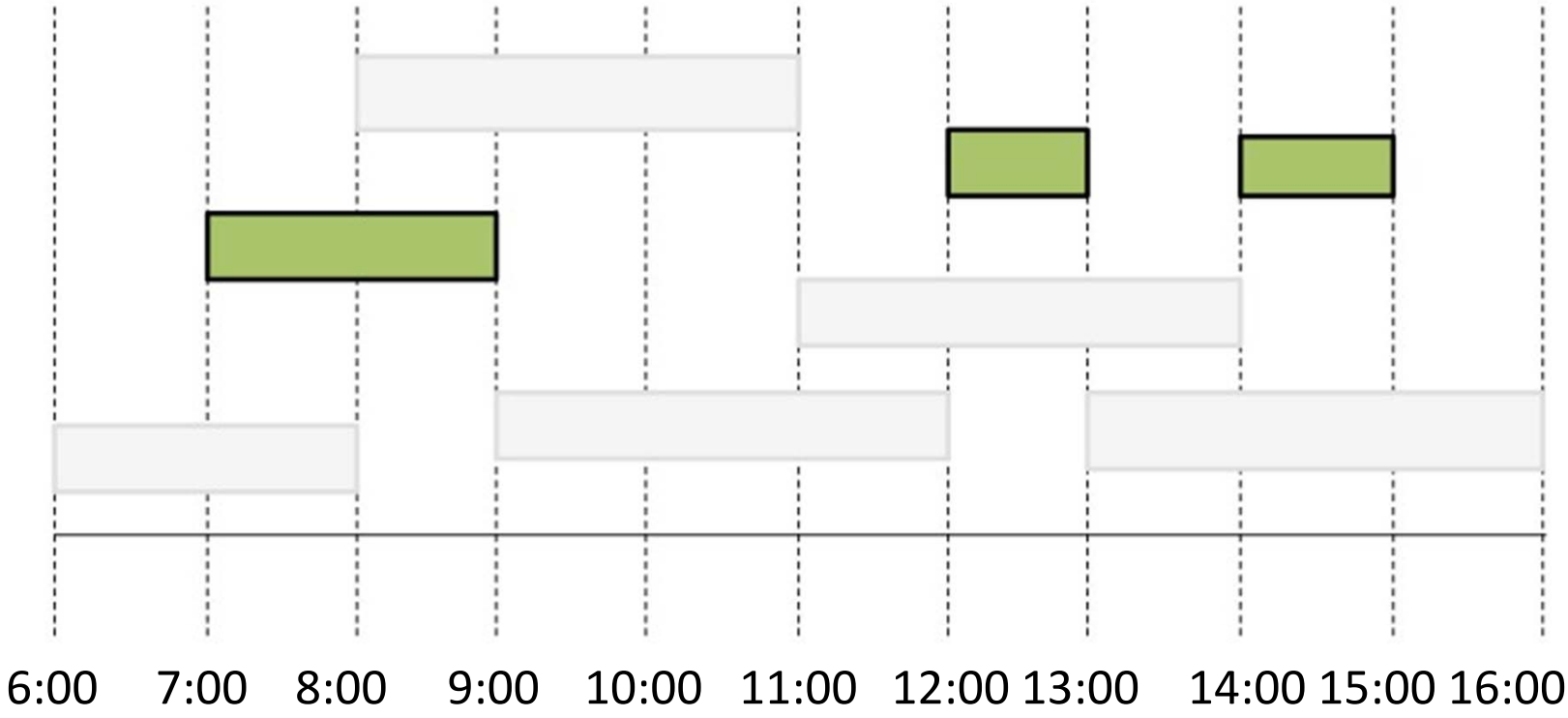
Exempel: Schemaläggning

- Du har fått *n* stycken aktiviteter att schemalägga under en dag. Inga aktiviteter får ske samtidigt, så några måste kanske hoppas över.
- Ditt jobb är nu att schemalägga så många aktiviteter som möjligt.

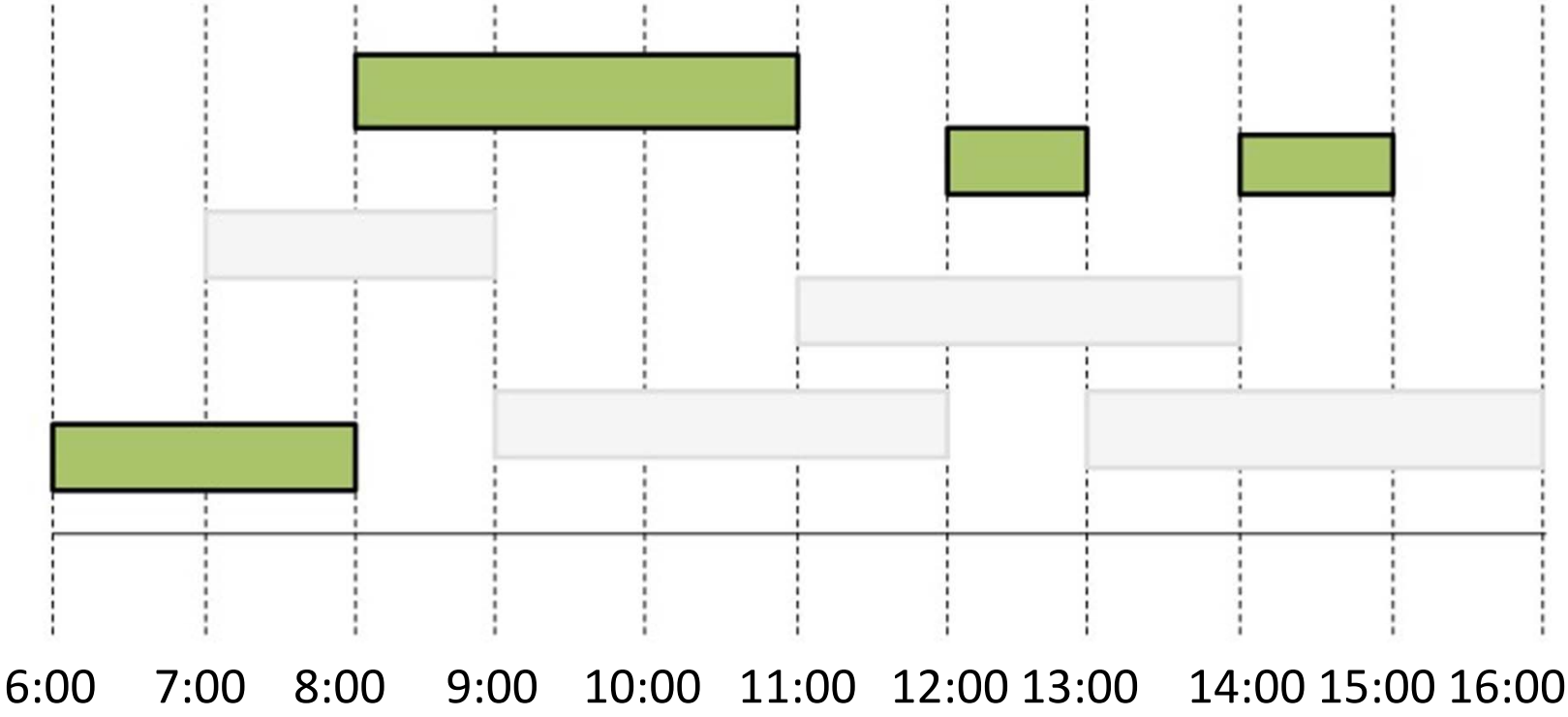
Schemaläggning



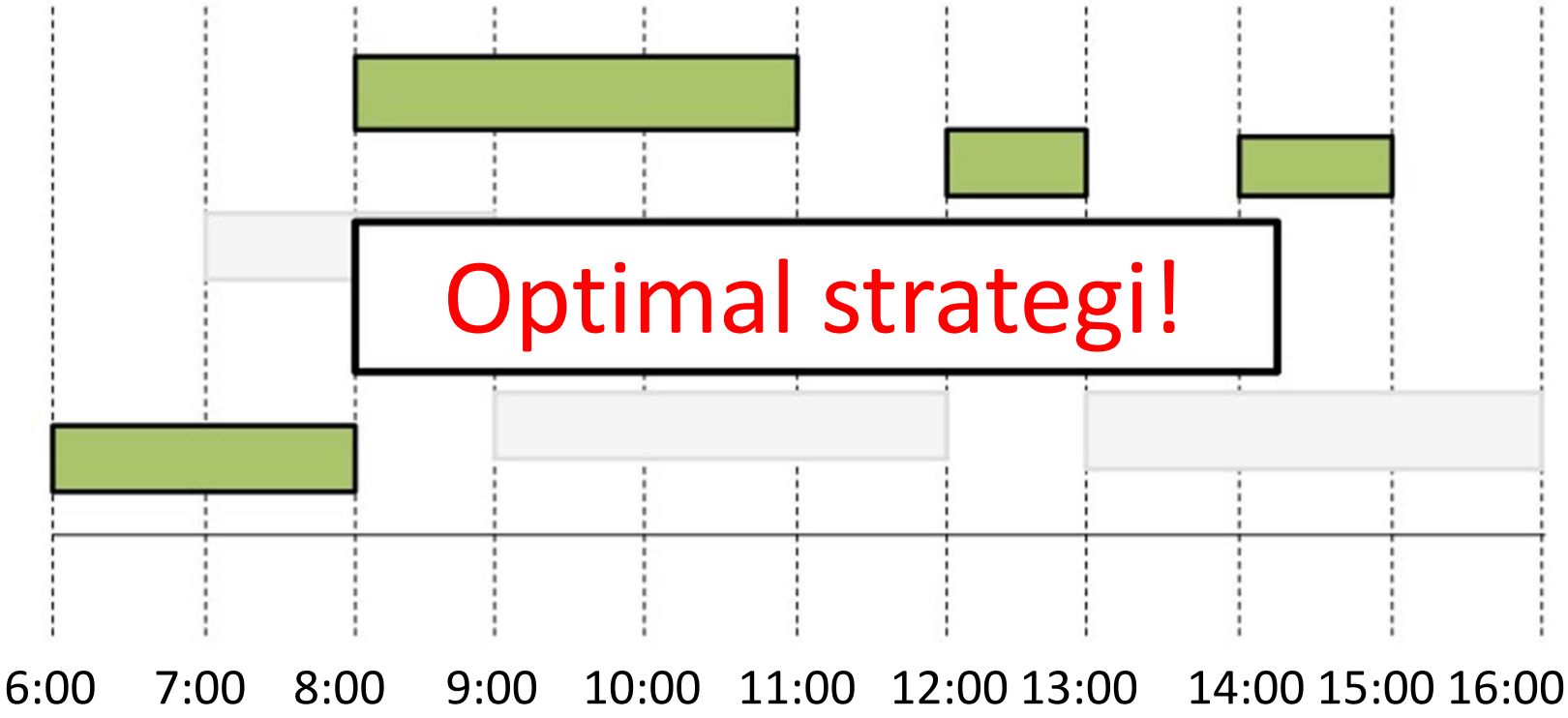
Idé: kortaste aktiviteter först: 3



Idé: Tidigast sluttid först: 4!



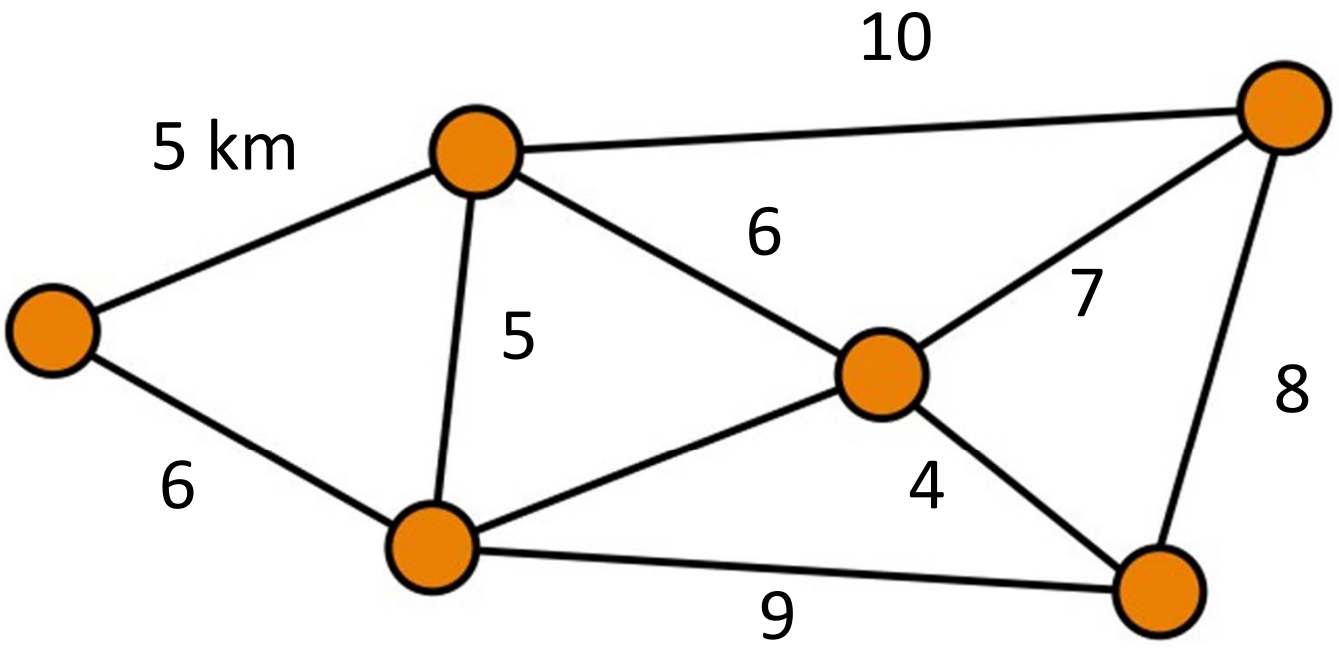
Idé: Tidigast sluttid först: 4!



Exempel: Handelsresandeproblemet

(Travelling Saleseman Problem)

- Hitta den kortaste rutten mellan alla städer så att varje stad endast besöks en gång



Exempel: Handelsresandeproblemet

- TSP är urtypen för ett “svårt” problem
- TSP är ett så kallat NP-komplett problem

- NP-kompletta problem karakteriseras av att
  - Inget problem har en effektiv lösning (än)
  - Om ett problem kan lösas effektivt kan alla

* “Effektivt” betyder att det kan lösas på tid som växer *polynomiellt* med antalet städer, i motsats till *exponentiellt*

Svåra problem “in real life”

Praktiskt viktiga men svåra problem finns t.ex. inom områdena

- Industriell logistikoptimering
- Schemaläggning
- Ruttoptimering (TSP)
- Packningsoptimering
- Kryptologi
- Data Mining
- Genetik

Beslutsproblem

- Beslutsproblem har ja/nej-svar

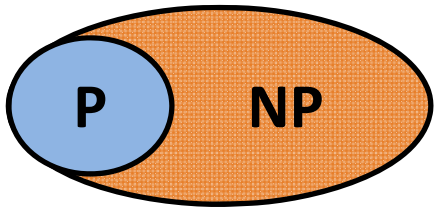
- Ex. SAT: Satisfierbarhet hos logiska uttryck. Givet ett uttryck $A(x_1,\ldots,x_n)$ med sanningsvariablerna $x_1,\ldots,x_n$, finns det en tilldelning av sanningsvärden T/F till $x_1,\ldots,x_n$ så att A blir sann?

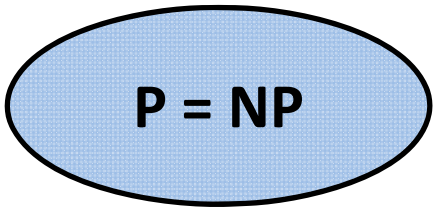
  Ex. $\{x_1=T, x_2=F, x_3=T\}$ satisfierar uttrycket $x_1 \,\&\, (x_2 \mid x_3)$

  Ex. Uttrycket $x \,\&\, !x$ är *inte* satisfierbart.

Svårighetsklasserna P, NP, NPC

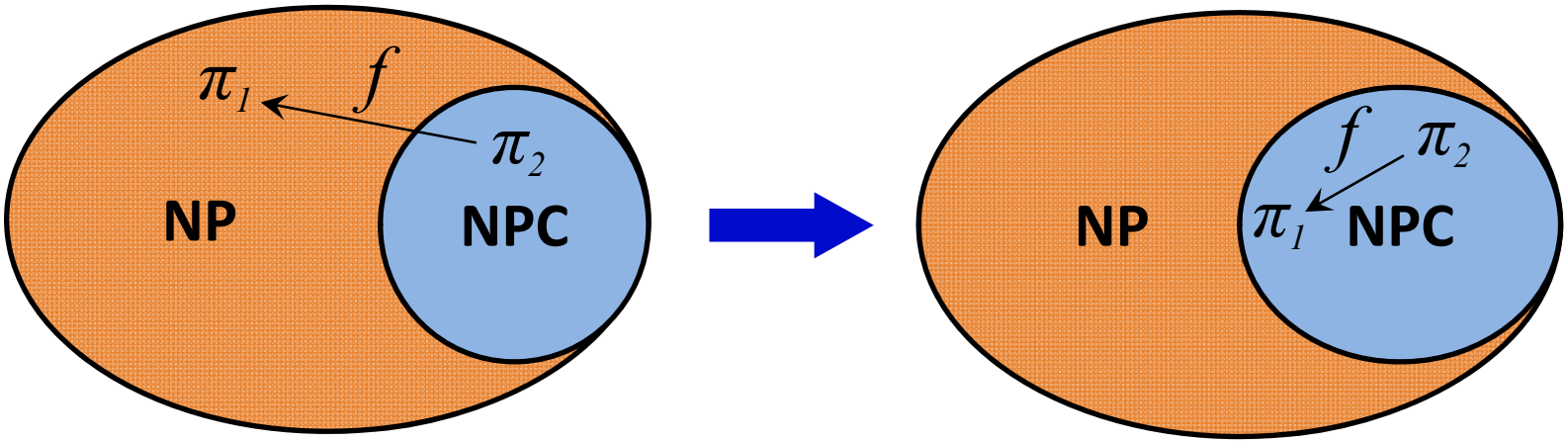
- Beslutsroblem i klassen P kan lösas av en deterministisk maskin i polynomiell tid
- Beslutsproblem i klassen NP:
  - kan lösas av en ickedeterministisk “orakelmaskin” i polynomiell tid (ingen sådan maskin är känd)
  - lösningen kan kontrolleras i polynomiell tid
- *Öppen fråga sedan länge: Är P = NP?*





NP-fullständiga problem: **NPC**

- De svåraste problemen i NP finns i klassen NPC.
- Ett problem π_1 i NP tillhör NPC om vilket annat NPC-problem som helst (π_2) kan översättas till π_1 i polynomiell tid



... men vilket var det första kända NPC-problemet?

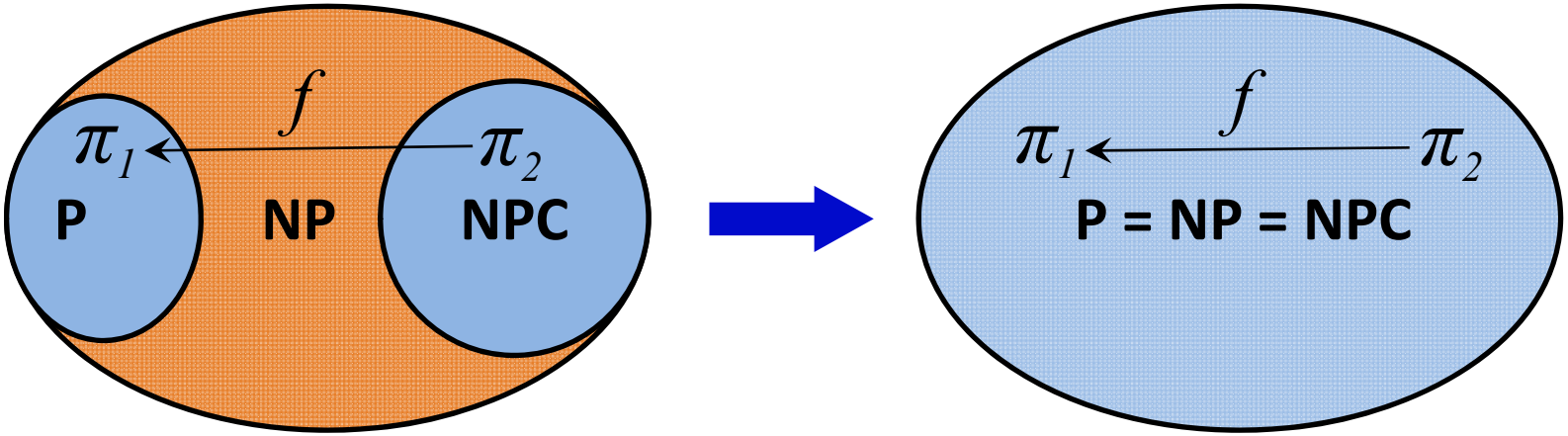
Cook’s teorem

Stephen Cook, Leonid Leven, 1971

Teorem: SAT är NP-fullständigt.
Bevis: Visa att ett godtyckligt problem i NP kan översättas i polynomiell tid av en deterministisk maskin till SAT. □

P = NP = NPC ?

- Det finns inget känt fall där ett NPC-problem kan översättas i polynomiell tid till ett problem i klassen P.
  - *Då kunde ju även NPC-problemet lösas i polynomiell tid och alla andra NPC-problem också!*

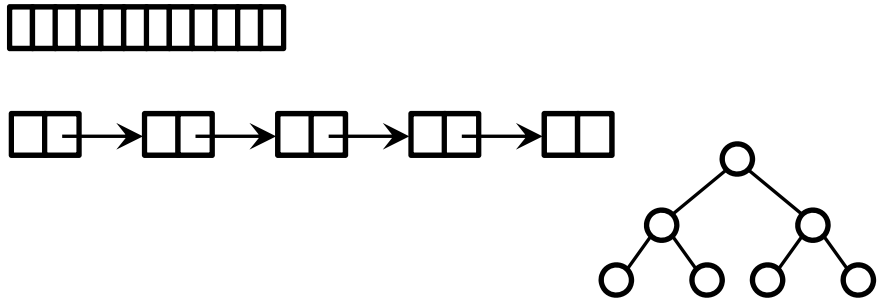


Datastrukturer – Abstrakta datatyper

- En Abstrakt DataTyp (ADT) är en datasamling med specifika åtkomstmönster för elementen.
- Exempel:
  - Kö: FIFO = First In First Out
  - Stack: LIFO = Last In First Out
  - Träd: Binärt sökträd, kodningsträd, abstrakt syntaxträd
- Syfte: Underlätta effektiv dataåtkomst för algoritmer

Datastrukturer – Abstrakta datatyper

- En datastruktur är en lagringsteknik för data som används för att *implementera* en Abstrakt datatyp.
- Exempel:
  - Fält
  - Länkad lista
  - Träd



- *Begreppen används något flytande.*

  Ofta menar vi ADT när vi säger datastruktur,

  det beror på abstraktionsnivån.

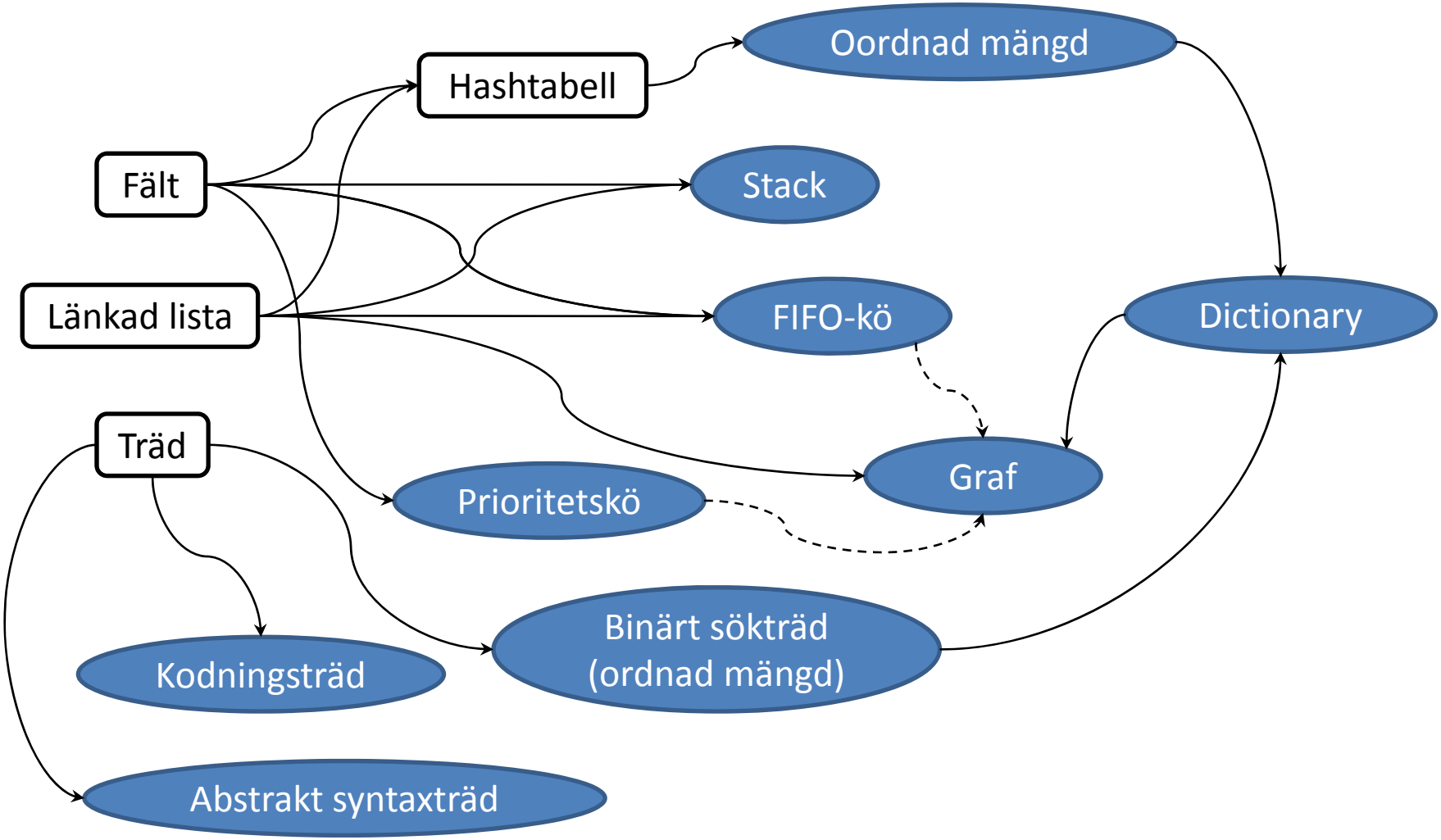
Datastrukturer & algoritmer

Algoritm + passande ds. => Snabbt program

Algoritm + dålig ds. => Långsamt program

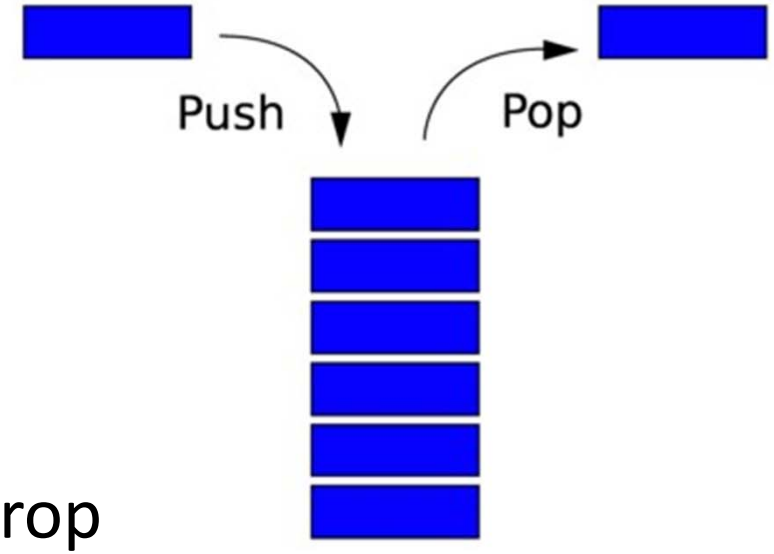
- Ibland designas algoritmer runt en specifik datastruktur.
- Ibland uppfinns nya datastrukturer för att göra en algoritm effektivare.
- Rätt datastruktur ger mer läsbar algoritm!

Datastrukturer och ADT:er



Stack (LIFO)

- Operationer
  - *push* - lägg till överst
  - *pop* - ta bort och returnera översta



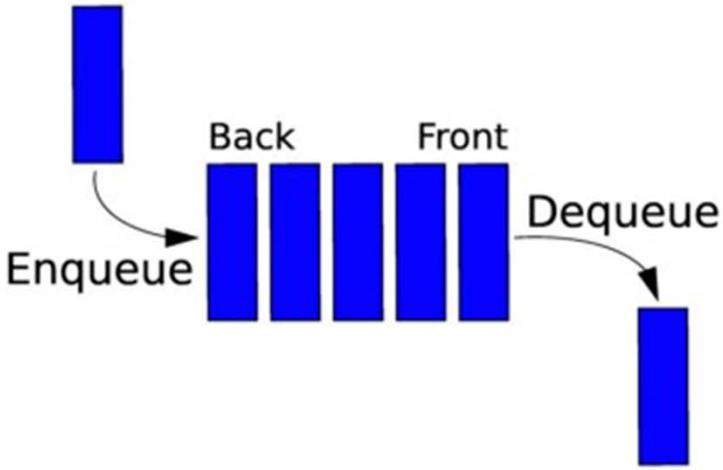
- Exempel:
  - Skriva ut listor baklänges
  - Hålla reda på funktionsanrop i Java

Anrop: f() → g() → h()

Return: f() ← g() ← h()

Kö (FIFO)

- Operationer
  - *enqueue* - lägg till först
  - *dequeue* - ta bort och returnera sista

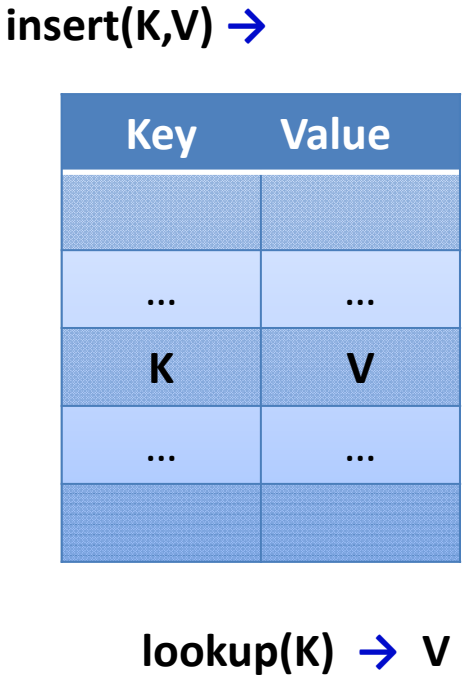


- Exempel:
  - Skrivarkö, grafsökning

Associationslista (*Dictionary*)

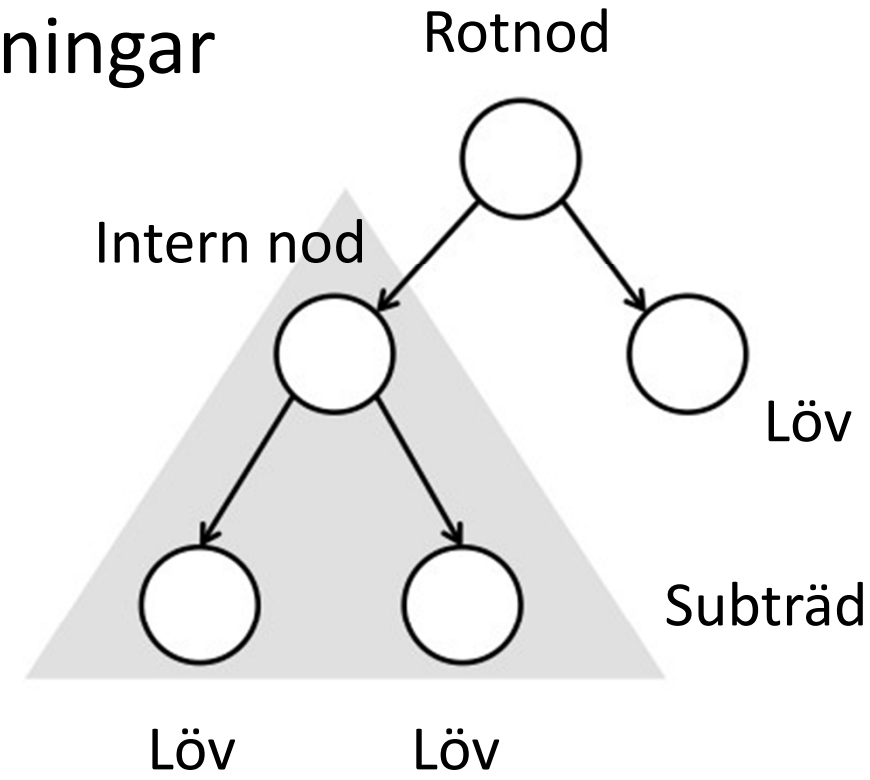
- Operationer
  - *insert* - associera nyckeln *K* med värdet *V*
  - *lookup* - hämta värdet som har nyckeln *K*

- Exempel:
  - Tabeller, avbildningar
  - Koppla ihop personnummer med personuppgifter



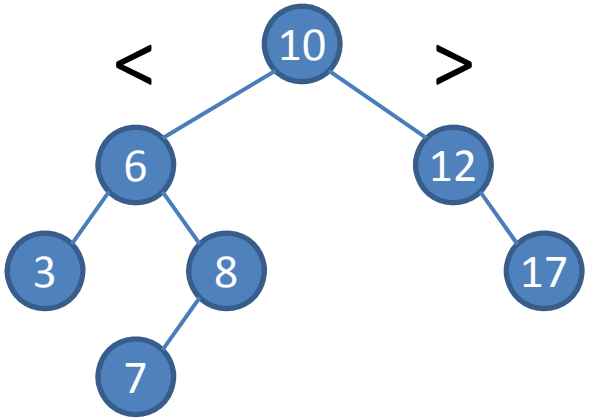
Träd

- Allmäna datastrukturer med många tillämpningar
- Noder och länkar
- Föräldrar och barn



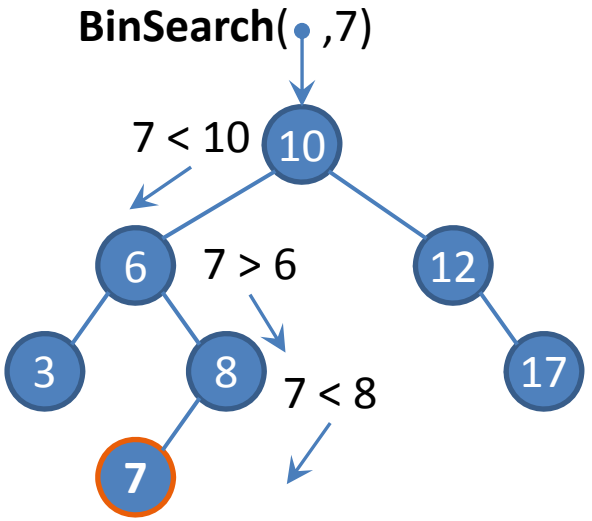
Binärt sökträd (ordnad mängd)

- Vanlig form av träd.
- Snabb insättning och borttagning i självsorterande binära trädstrukturer.
- Max 2 barn per nod.
- Vänster = mindre, Höger = större



Binärsökning igen!

```
procedure BinSearch( Tree, Value )
if( root pointer == null )
    return “Search failed”
else
    TestEntry ← value of root node
    if ( Value = TestEntry )
        return “Search successful”
    if ( Value > TestEntry )
        BinSearch( RightSubtree, Value )
    else
        BinSearch( LeftSubtree, Value )
```



Sammanfattning

- Algoritmer är en viktig del av problemlösning
- Datastrukturer är nödvändiga verktyg för att implementera effektiva algoritmer
- Många problem har standardiserade lösningar
- Många problem är svåra att lösa effektivt