Algoritmer och datastrukturer TDA143

2015-02-18

Fredrik Johansson

algorithms

Webben | Bilder | Videor | Nyheter | Böcker | Fler | Sökverktyg

Ungefär 78 400 000 resultat (0,23 sekunder)

Algorithm - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Algorithm** ▾ Översätt den här sidan
In mathematics and computer science, an **algorithm** is a self-contained step-by-step set of operations to be performed. **Algorithms** exist that perform calculation, ...
Muḥammad ibn Mūsā al ... - List of algorithms - Euclidean algorithm - Algorism

Algorithms | Computer science | Khan Academy
https://www.khanacademy.org/.../**algorithms** ▾ Översätt den här sidan
We've partnered with Dartmouth college professors Tom Cormen and Devin Balkcom to teach introductory computer science **algorithms**, including searching, ...

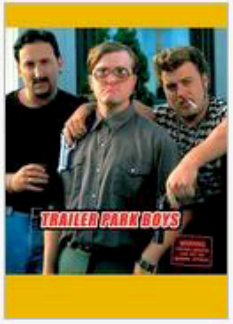
Algorithms, Part I - Coursera
https://www.coursera.org/course/algs4partI ▾ Översätt den här sidan
Algorithms, Part I is a free online class taught by Kevin Wayne & Robert Sedgewick of Princeton University.

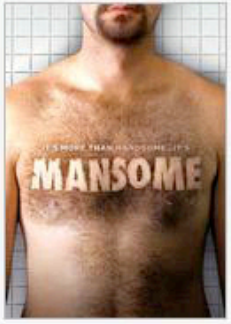
Algorithms, 4th Edition by Robert Sedgewick and Kevin ...
algs4.cs.princeton.edu/ ▾ Översätt den här sidan
The textbook **Algorithms**, 4th Edition by Robert Sedgewick and Kevin Wayne [Amazon · Pearson] surveys the most important **algorithms** and data structures in ...

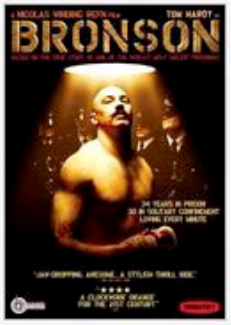
What is algorithm? - Definition from WhatIs.com
whatis.techtarget.com › ... › Mathematics ▾ Översätt den här sidan
An **algorithm** (pronounced AL-go-rith-um) is a procedure or formula for solving a

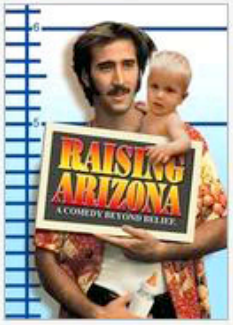
NETFLIX Watch Instantly Just for Kids Your Queue Taste Profile DVDs

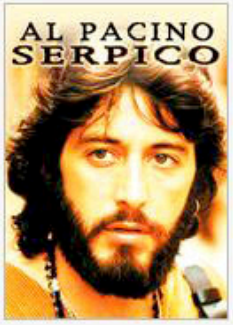
Cool Moustaches

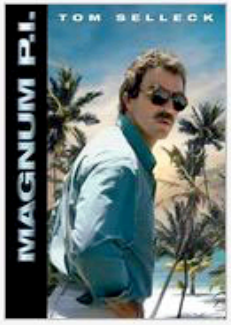


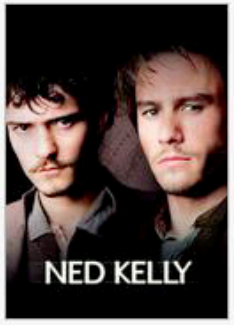


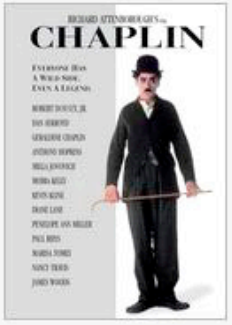


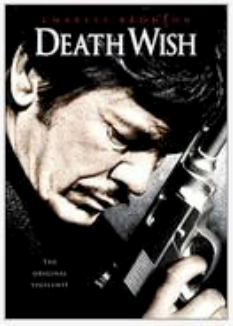


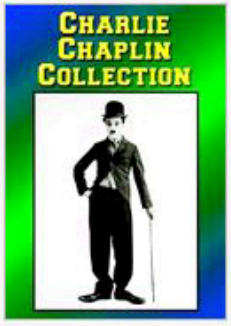


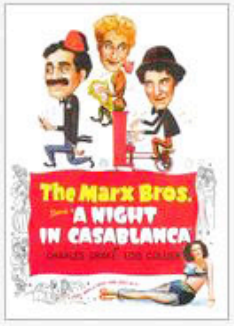


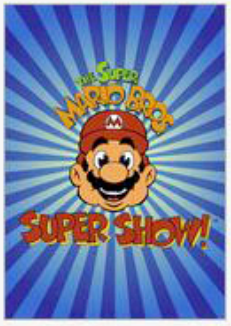


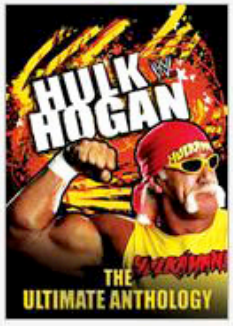













Terminal 1 10:15 Berlin BA982
Terminal 3 10:20 Helsinki BA6433
Terminal 1 10:20 Oslo SK804
Terminal 1 10:25 Nairobi BA065
Terminal 3 10:25 Houston BA195
Terminal 5 10:25 Los Angeles BA283
Terminal 1 10:25 Brussels SN2094
Terminal 5 10:25 Newark UA29
Terminal 5 10:30 Lagos BA075
Terminal 1 10:30 Kuwait KU104
Terminal 4 10:30 Los Angeles

Algoritmer

Informell beskrivning

Ett antal steg som beskriver hur en uppgift görs.

A set of steps that defines how a task is performed.

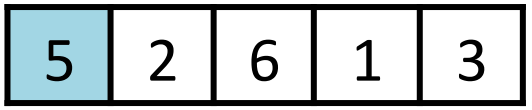
Formell beskrivning:

En algoritm är ett ordnat sätt av entydiga, utförbara steg som beskriver en terminerande process

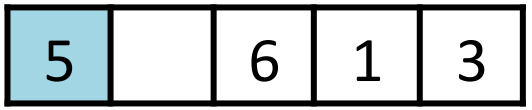
Brookshear: Computer Science, An overview

Exempel: Sortering

1. Antag första siffran “sorterad”

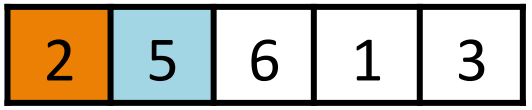


2. Hitta första osorterade siffra

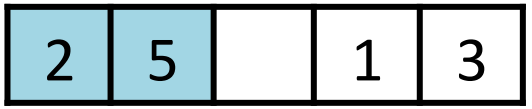


2

3. Hitta rätt plats i sorterad del



4. Upprepa med resten av listan



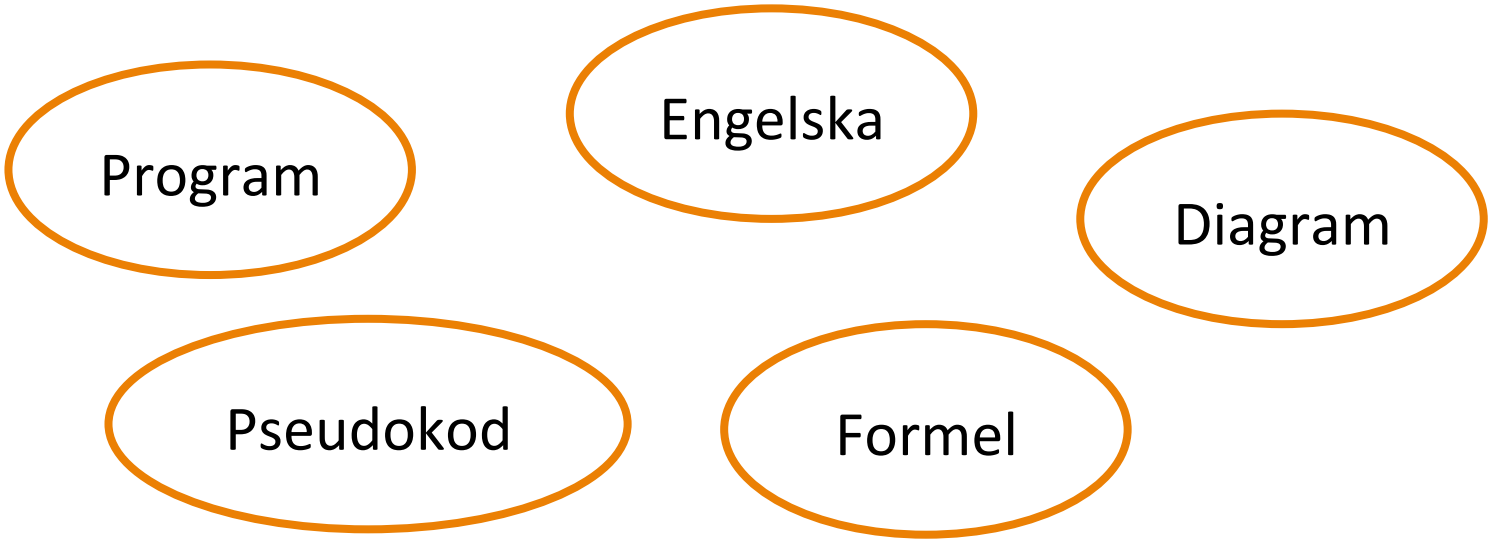
6

...

Algoritmer & representationer

Algoritm: abstrakt idé för att lösa ett problem

Representation: formulering av den abstrakta idén som till exempel:



Representation: Fahrenheit

a) "Multiply the temperature in Celcius by 9/5 and add 32 to the product"

b) $F = 9/5*C + 32$

c)
```
function fahrenheit(c) {
    return 9/5*c + 32;
}
```

Liknelse: bakning

Algoritm	Idé om hur man bakar äpplepaj
Representation	Recept
Implementering	Bageri

Pseudokod

- Vanligaste representationen
- Mellanting mellan naturligt 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?
- *Effektivitet:* hur lång tid tar det att köra algoritmen?

Tidskomplexitet

- Algoritmers körtid mäts i antal *operationer:*
  - *additioner*
  - *tilldelningar*
  - *…*
- Inte i sekunder! *Varför?*
- För en given input-storlek (t ex antal element i lista), hur många operationer utförs?

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"
```

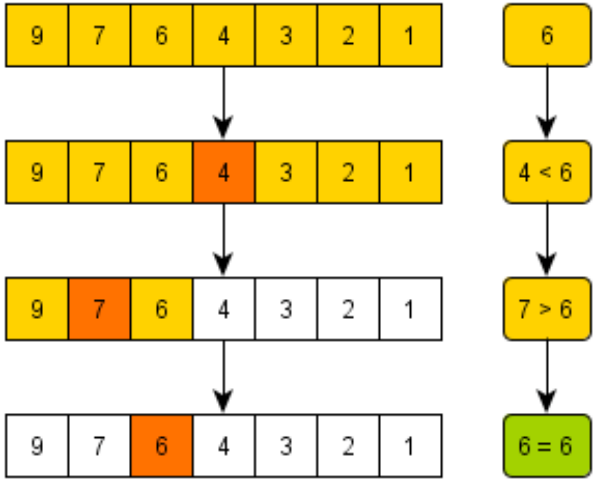
- Om listan har *n* element behövs (i värsta fallet)
  - n uppslag
  - n tilldelningar
  - n jämförelser
  - 1 returnering

  $= 3n + 1$ operationer!

Kan vi göra bättre?

Binärsökning

- Om listan är sorterad



procedure *BinSearch* (List, Value)

if(List empty)

return “failed”

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

Komplexitet och Big-O

- Vanligen räknar man inte exakta antal. Istället beräknar man den *asymptotiska komplexiteten – hur antalet växer med data!*

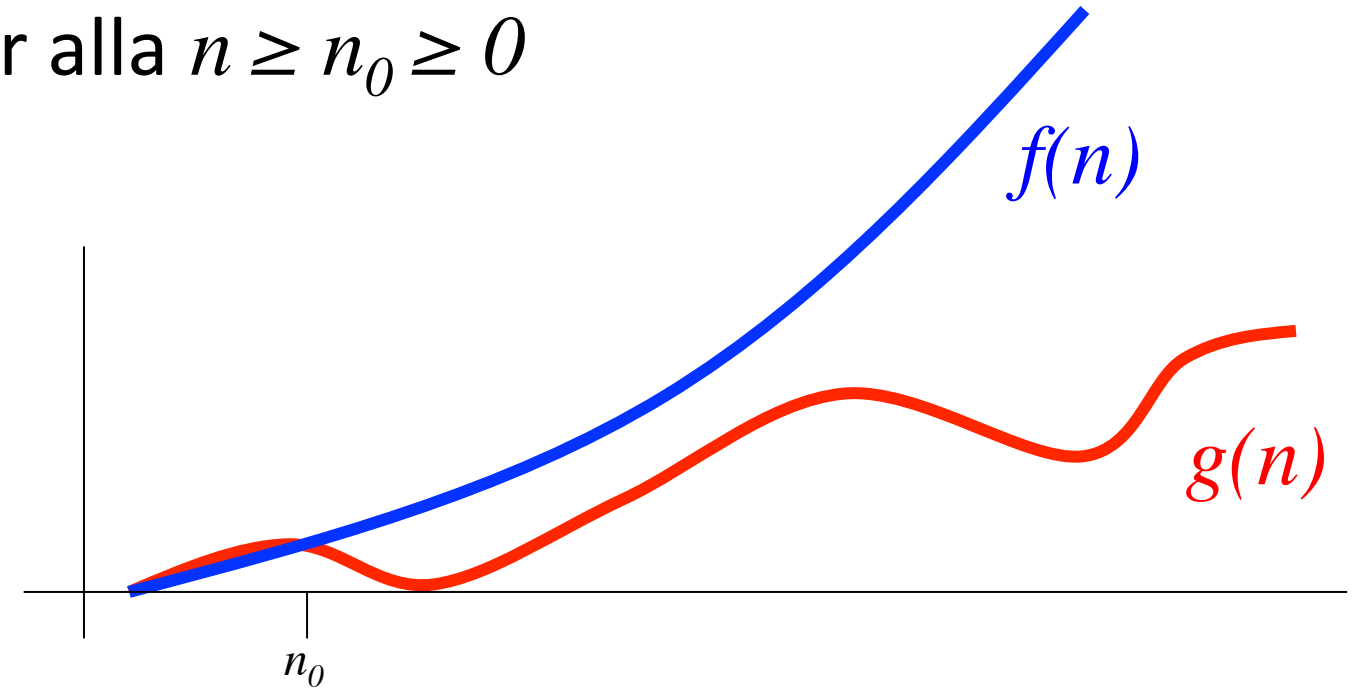
- Exempel: Om vi dubblar antalet element, hur mycket längre tid tar det?

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

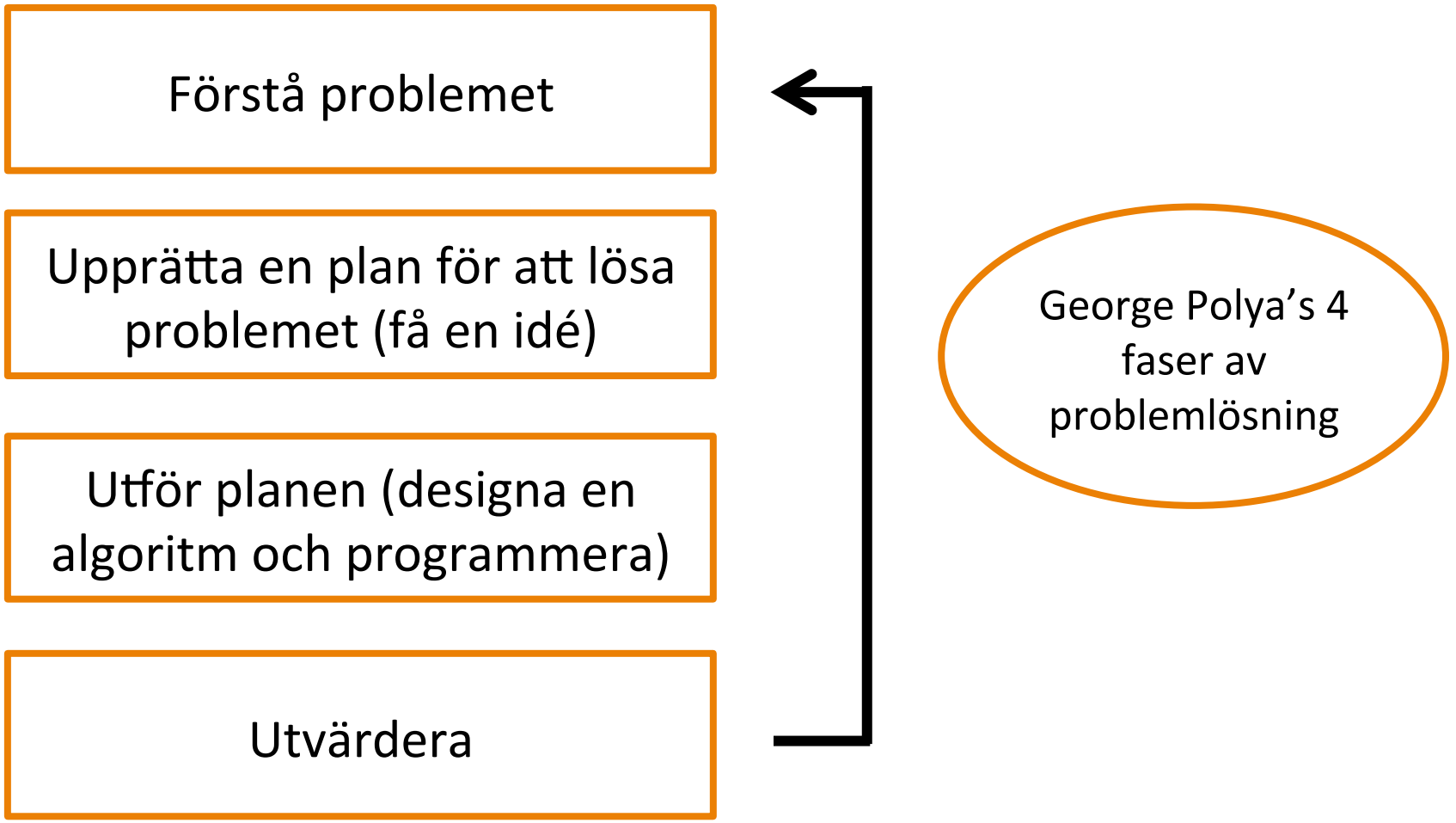
Sortering: tidskomplexitet

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

    => $O(n^2)$ operationer

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

Algorithmer och problemlösning



Algoritmer och problemlösning

- Vissa problem är *olösbara (undecidable)* och vissa problem är *hopplöst svåra* (NP-complete)!

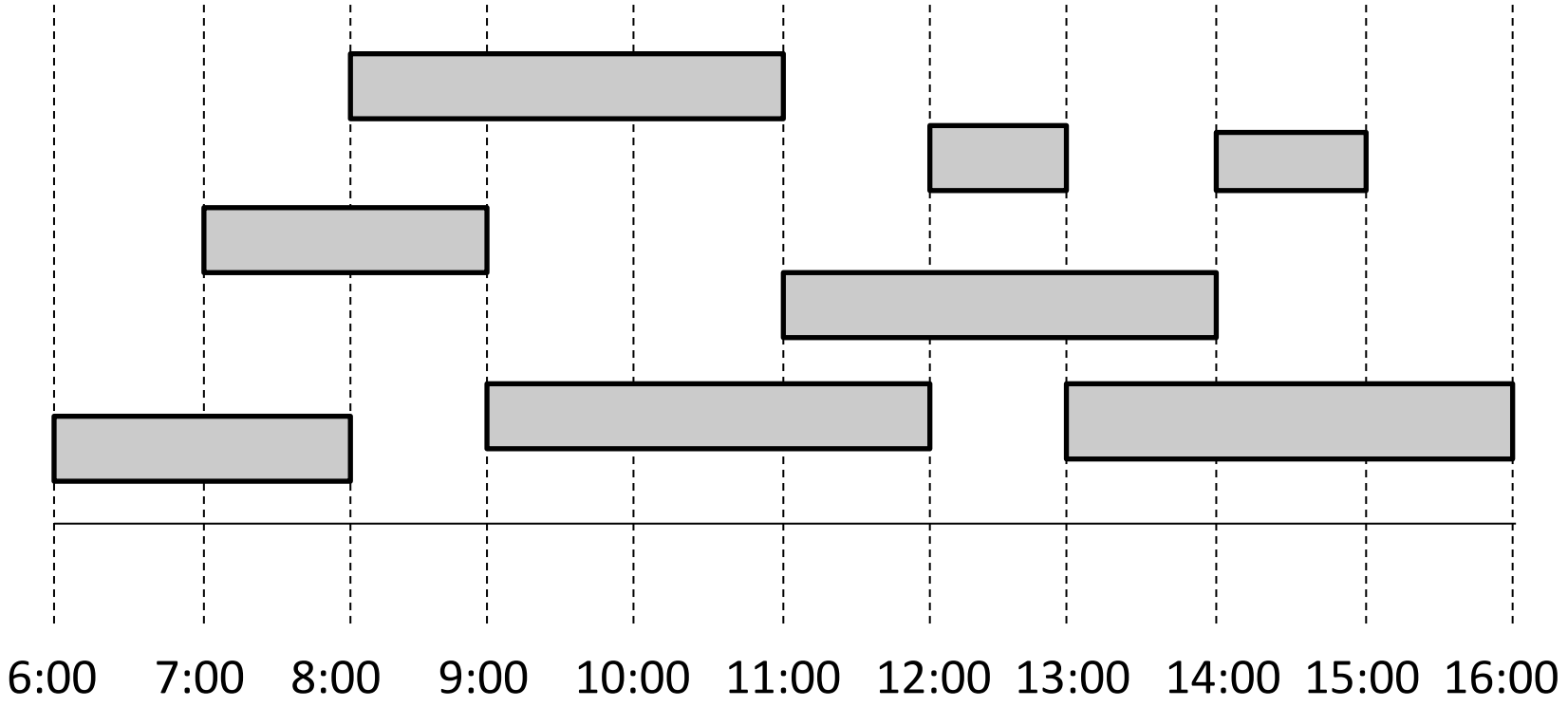
- Andra problem är redan lösta – standardproblem!

Exempel: Schemaläggning

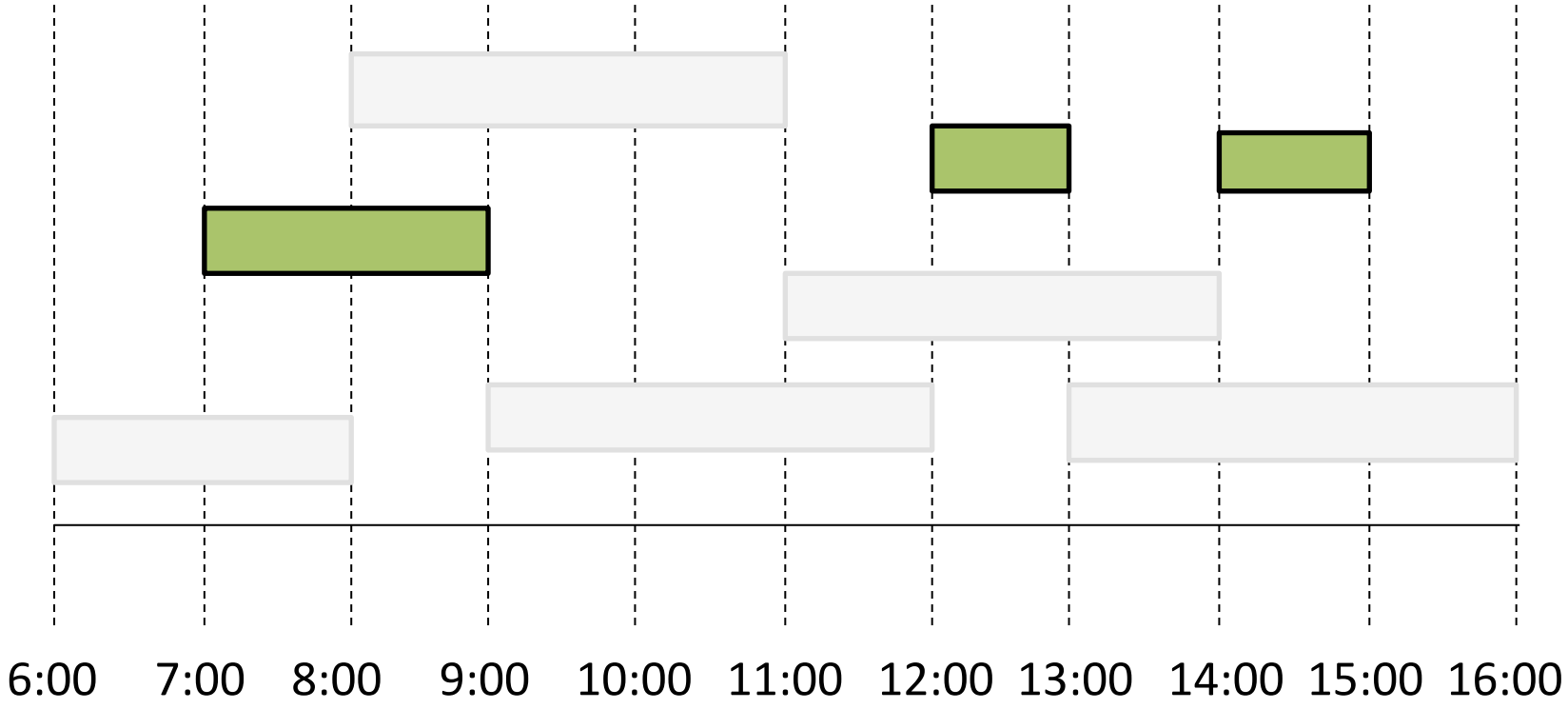
- Du har fått *n* stycken händelser att schemalägga under en dag. Inga händelser får ske samtidigt, så några måste hoppas över.

- Ditt jobb är nu att schemalägga så många händelser som möjligt

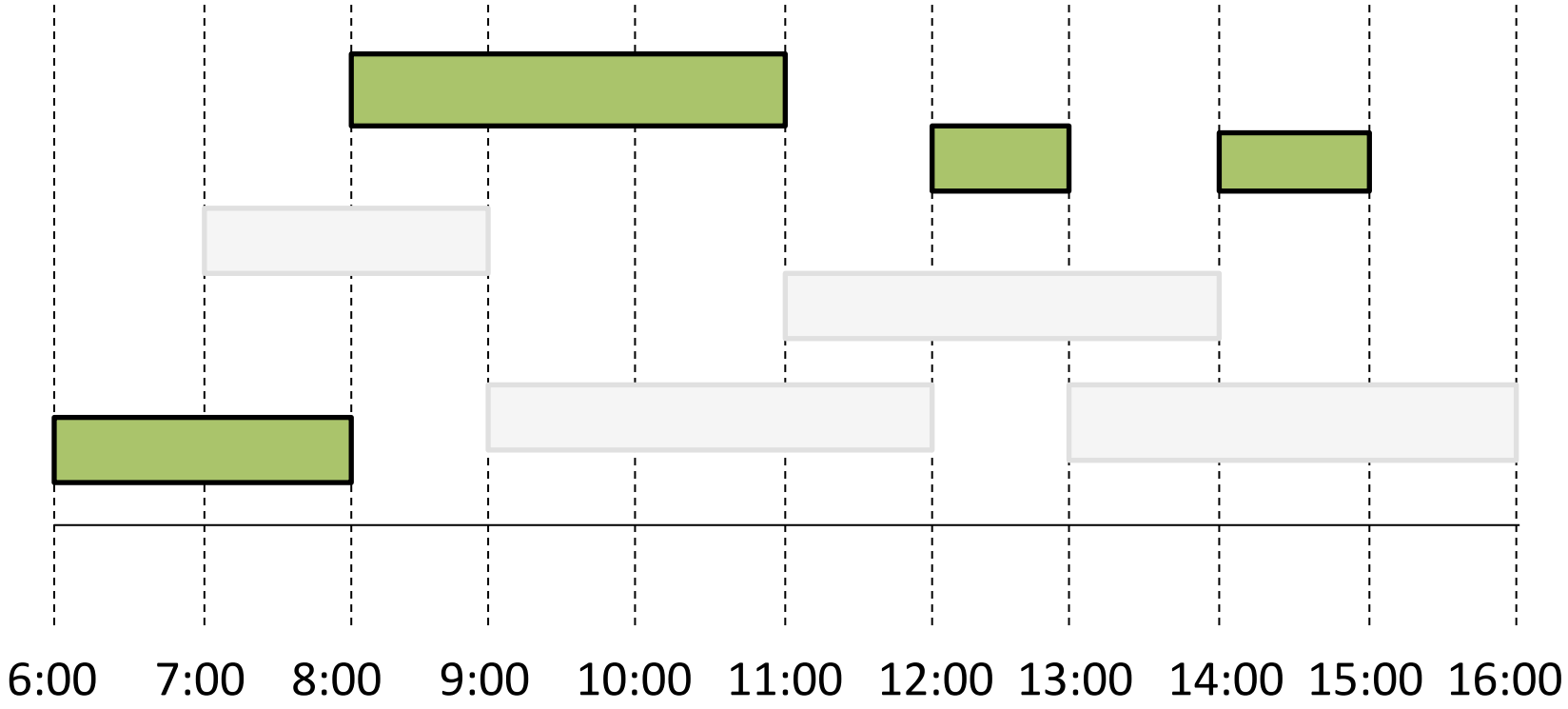
Schemaläggning



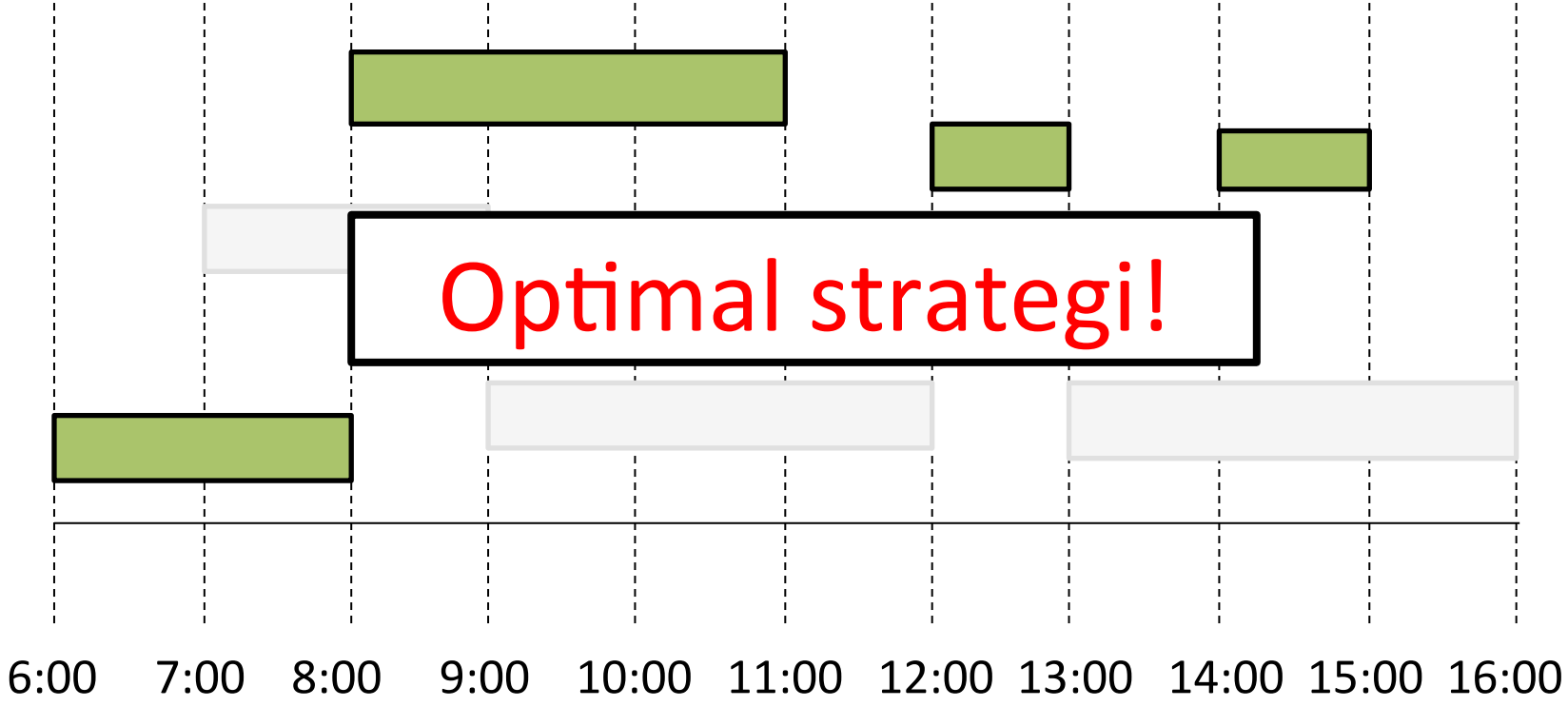
Idé: kortast händelser först: 3!



Idé: tidigast sluttid först: 4!



Idé: tidigast sluttid först: 4!



Att angripa problem

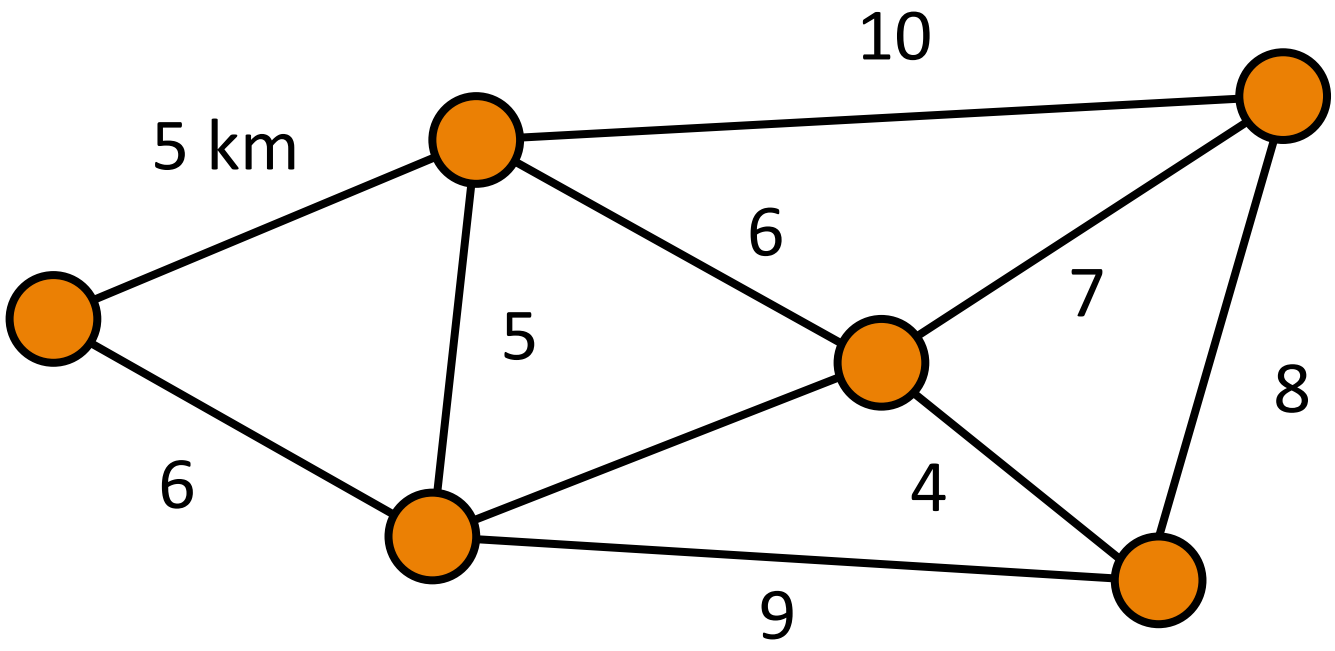
Hur hittar man rätt algoritm?

- Top-down: förfina lösningen gradvis
- Bottom-up: lös delar och kombinera

- Hitta *liknande* problem med kända lösningar

Exempel: Travelling salesman

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



Exempel: Travelling salesman

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

- NP-kompletta problem är lika på flera sätt
  - Inget problem har en effektiv algoritm (än)
  - Om ett problem löses effektivt kan alla göra det

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

Designprinciper för algoritmer

- Uttömmande sökning (brute-force)
- Giriga algoritmer
- Divide-and-conquer
- Dynamic programming
- (Heuristiker)
- …

Datastrukturer

- Mål: Underlätta dataåtkomst för algoritmer för ökad *effektivitet*

- Exempel:
  - Fält
  - Listor
  - Köer
  - …

Datastrukturer & algoritmer

Algoritm + passande ds. => Snabbt program

Algoritm + dålig ds. => Långsamt program

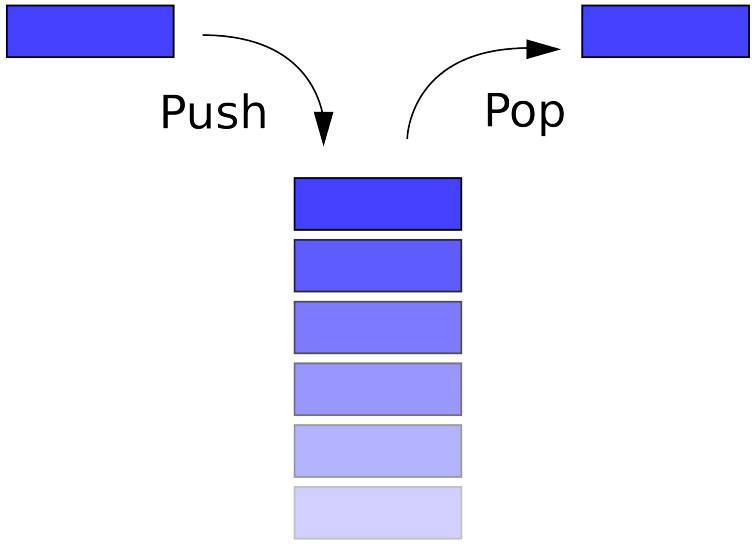
- Ibland designas algoritmer runt en specifik datastruktur
- Ibland uppfinns nya datastrukturer för att hjälpa en algoritm

Datastrukturer

- Fält
- Listor
- Köer
- Träd
- Grafer
- Stack
- Dictionaries
- Mängder
- Kombinationer
- …

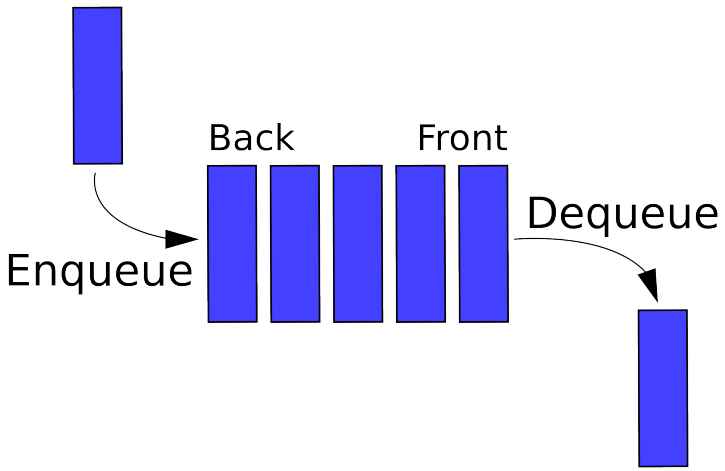
Stack

- Operationer
  - *push* – lägg till överst
  - *pop* – ta bort och returnera översta
- Exempel:
  - Skriva ut listor baklänges



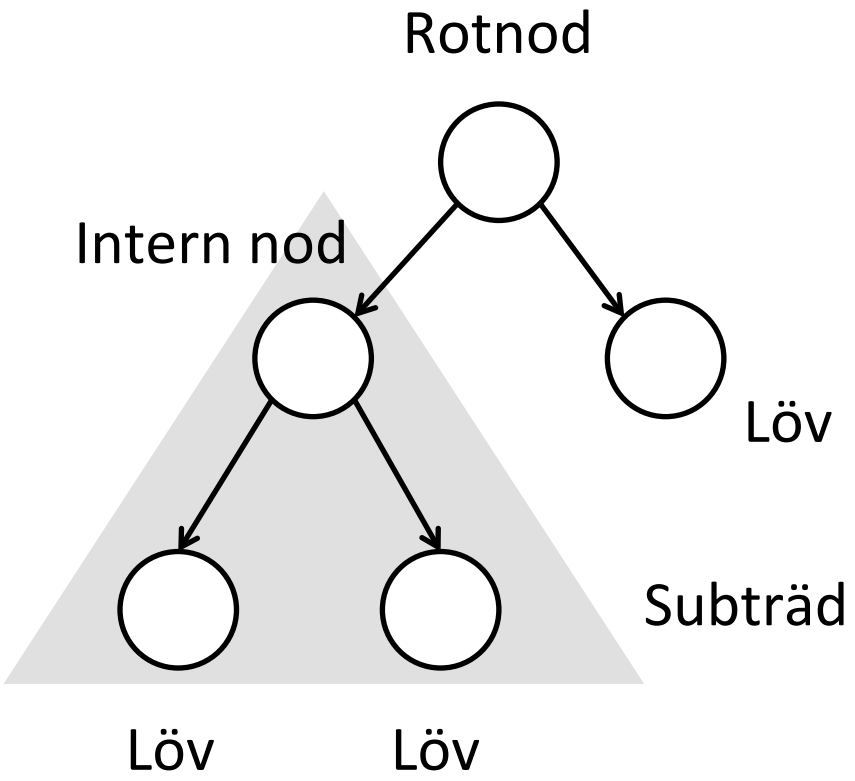
Kö

- Operationer
  - *Enqueue* – lägg till första
  - *dequeue*– ta bort och returnera sista
- Exempel:
  - Verkliga köer…



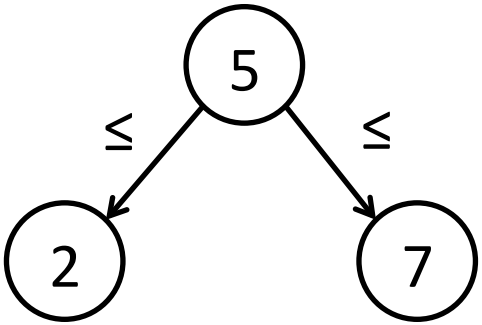
Träd

- Väldigt allmän datastrukturer
- Noder och länkar
- Föräldrar och barn



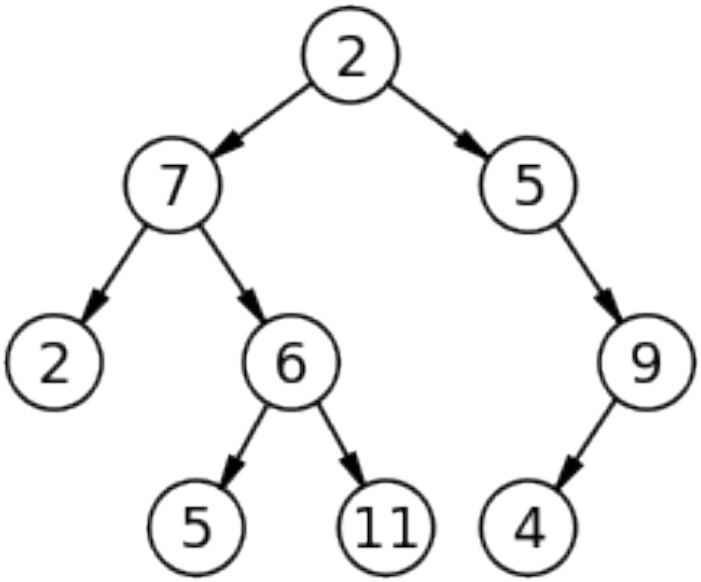
Binära sökträd

- Vanligt form av träd
- Snabb insättning och borttagning i självsorterande strukture
- 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
- Datastructures är nödvändiga verktyg för att implementera effektiva algoritmer
- Många problem har standardiserade lösningar
- Många (andra) problem är väldigt svåra
- Använd designprinciper