Algorithms

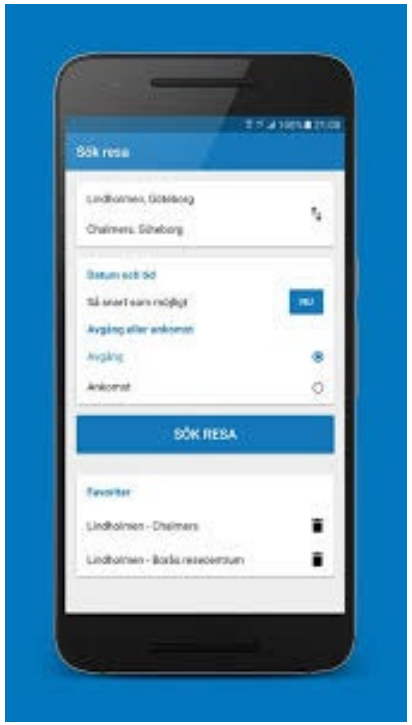
Guest lecture in the course Python for data scientists

Birgit Grohe

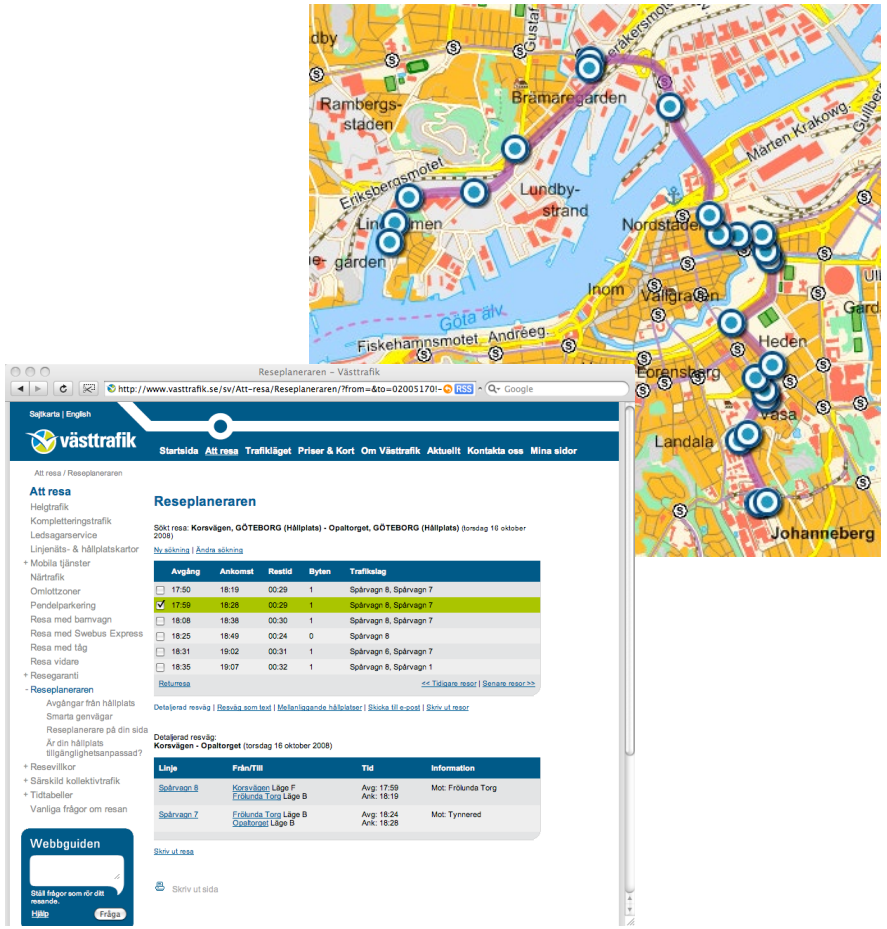
2020-10-21


Algorithm
Game theory
BFS
DFS
Shortest
graph
Matching
Maximum
Searching
Search
Sorting
points
convex
path
ICPC
Hungarian
vertex
Longest
First
Optimal
Exponentiation
Subsequence

Used an Algorithm Today?

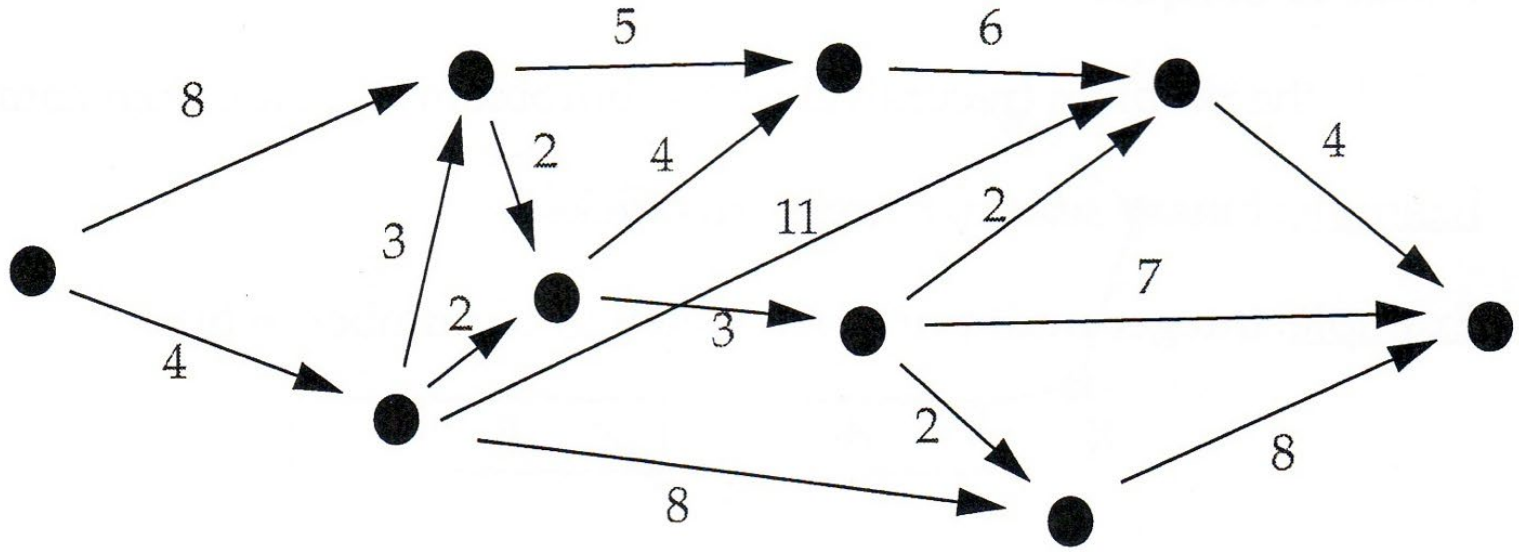


The Shortest Path Problem



How solve?

Solving the directed shortest path problem with dynamic programming



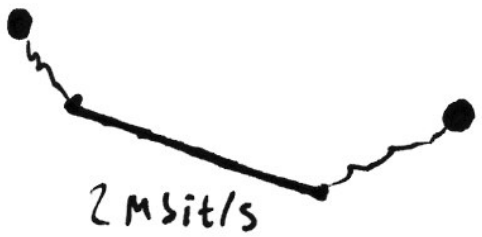
Traverse nodes from "left to right" and mark with distance from origin. Dijkstra´s algorithm 1956 .

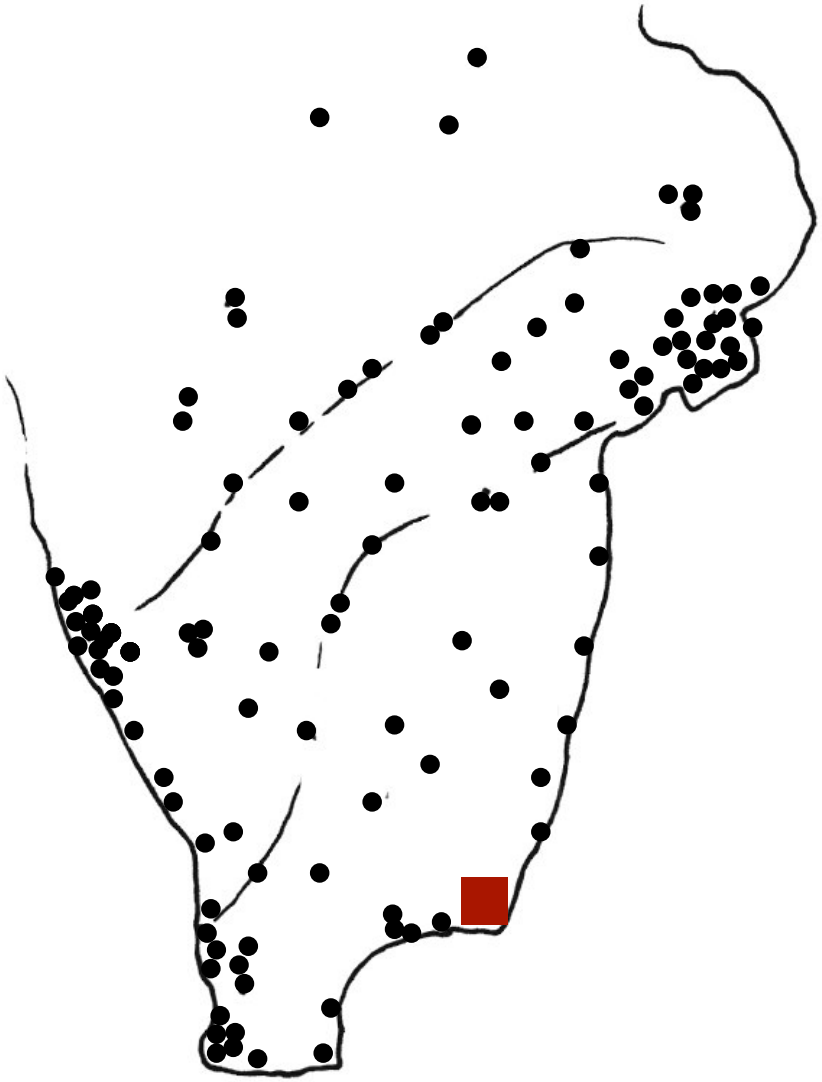
Circumvents the combinatorial explosion!
(not possible for all kinds of problems)

Telephone operator problem (real applied problem)

A Swedish mobile phone operator needs to connect all base station to its main switch.

How can we best rent communication lines from the national fixed network?





What is an Algorithm?

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

An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process

Brookshear

What is an Algorithm?

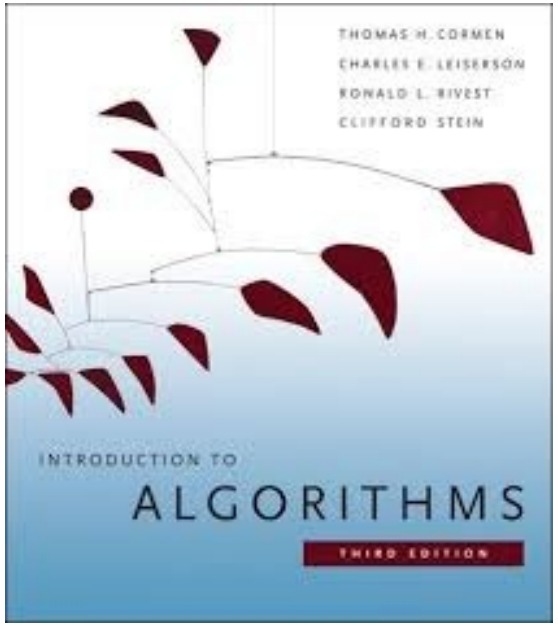
An Algorithm is a well-defined computational procedure that takes some value, or a set of values, as input and

produces some value, or set of values as output.

An algorithm is thus a sequence of computational steps

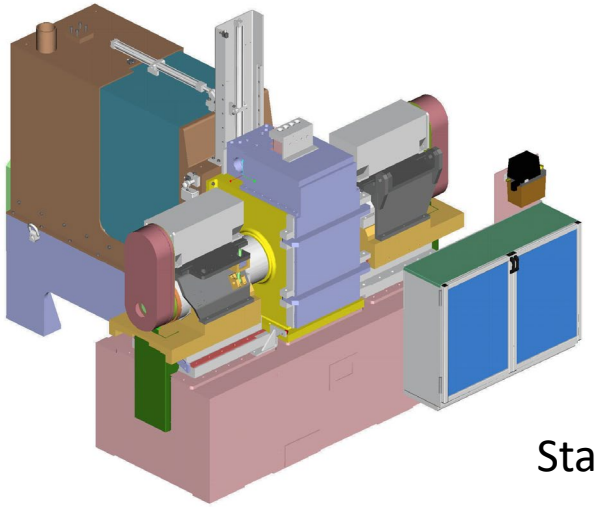
that transform input into output.

- More formal definition uses Turing machines

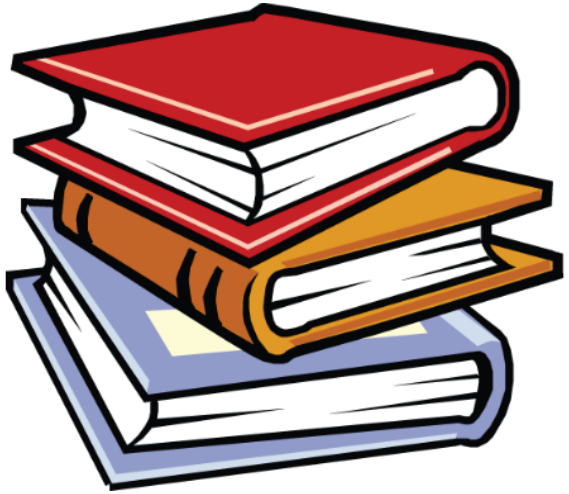


Studying Algorithms?

Toolbox: Design Principles



Standard Algoritms



Standard Problems

Algorithm Analysis

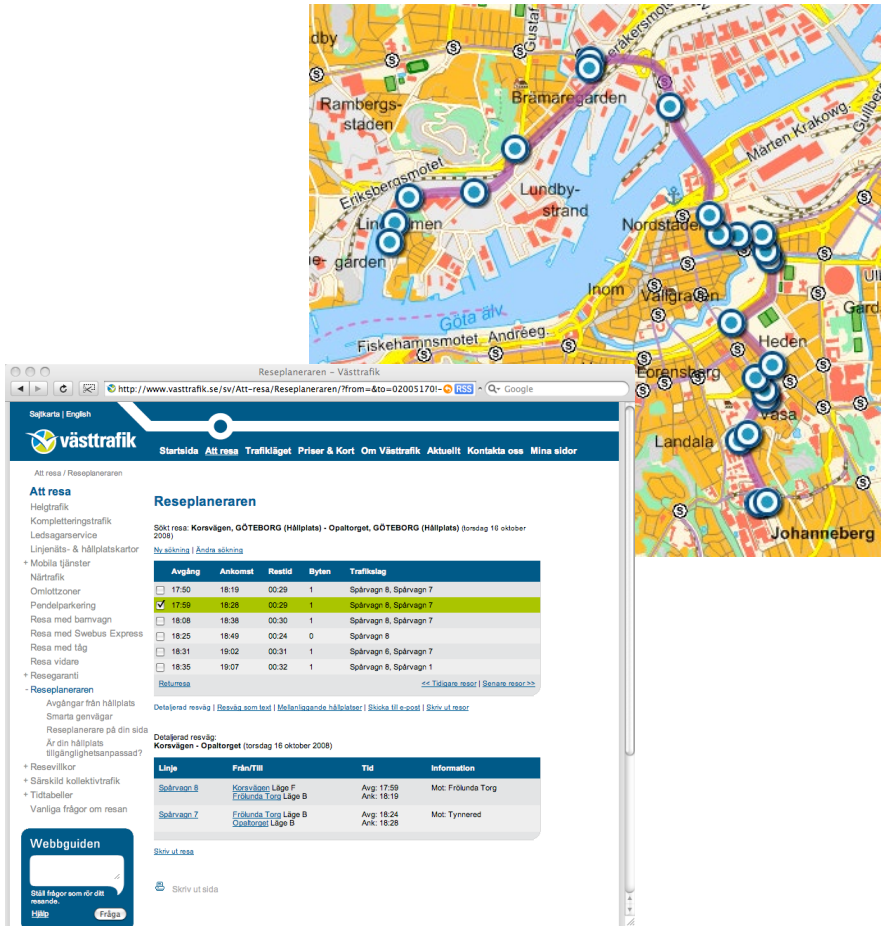
Running time

- How much ´time´ does the algorithm take?
- How do we measure the running time?

Correctness

- Does the algorithm do what it is supposed to do?
- Is it enough to test an algorithm on some instances?

The Shortest Path Problem



polynomial growth

n	$c\ n^2$
10	0,001 s
20	0,004 s
30	0,009 s
40	0,016 s
50	0,025 s
60	0,036 s

Time complexity: Big-O

- O() (pronounced ”Big-O” or ”Order of”)
- Often written a little curly
- Related symbols: Ω (Omega), Θ (Theta)

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

A youtube video on Time complexity and O() incl the formal definition

https://www.youtube.com/watch?v=4UYo8muFwAM

Time complexity: Big O

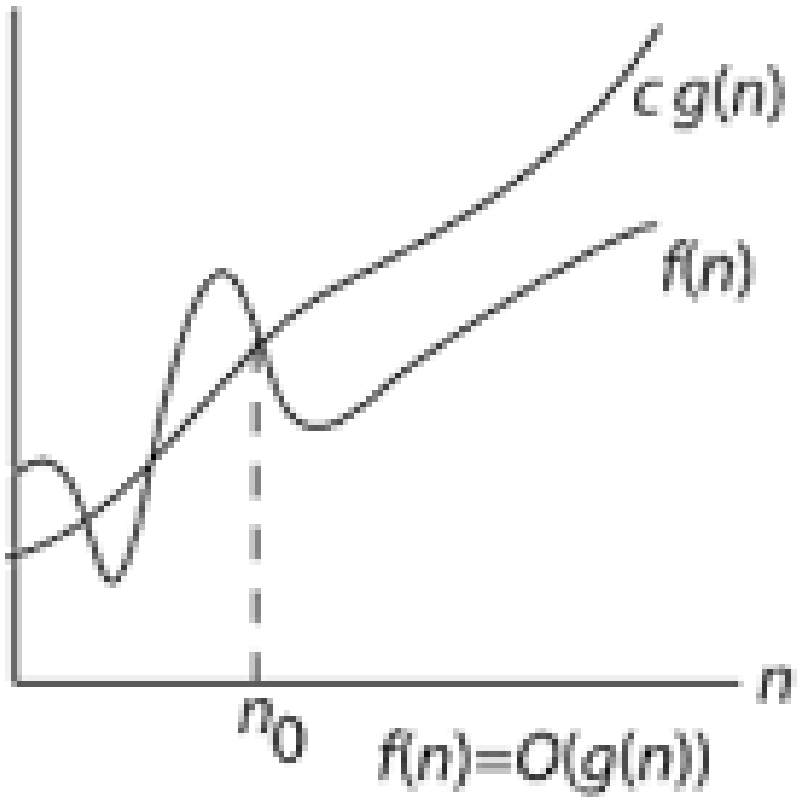
Let n be the input size of a problem.

A function f(n) is O(g(n)) if there exit constants c > 0 and n_0 ≥ 0 such that for all n ≥ n_0 we have

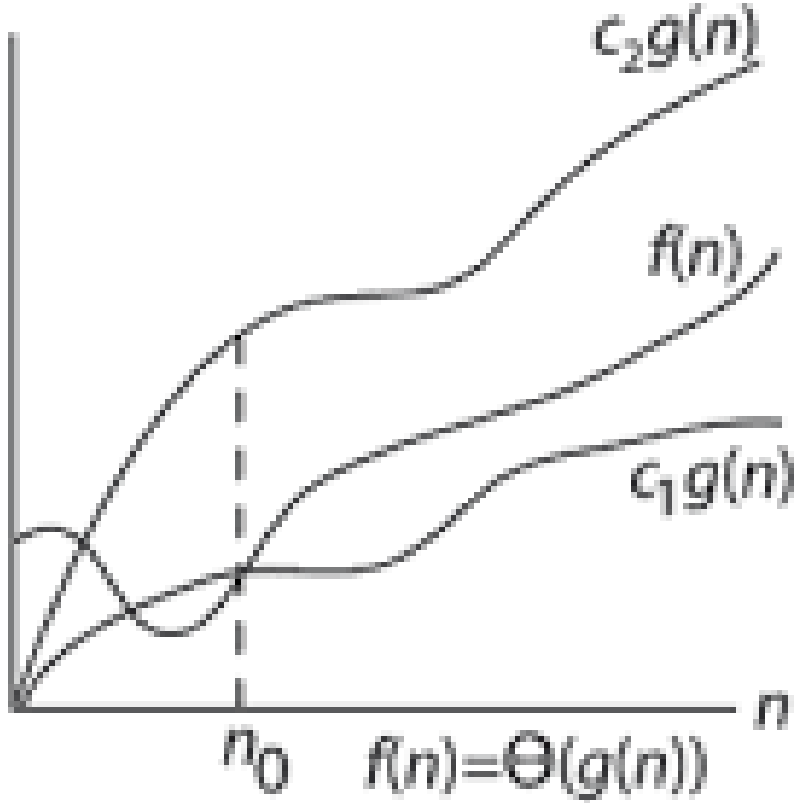
$$f(n) \leq c\, g(n)$$

Function f *is asymptotically upper bounded* by the function g.

Time complexity: Big O



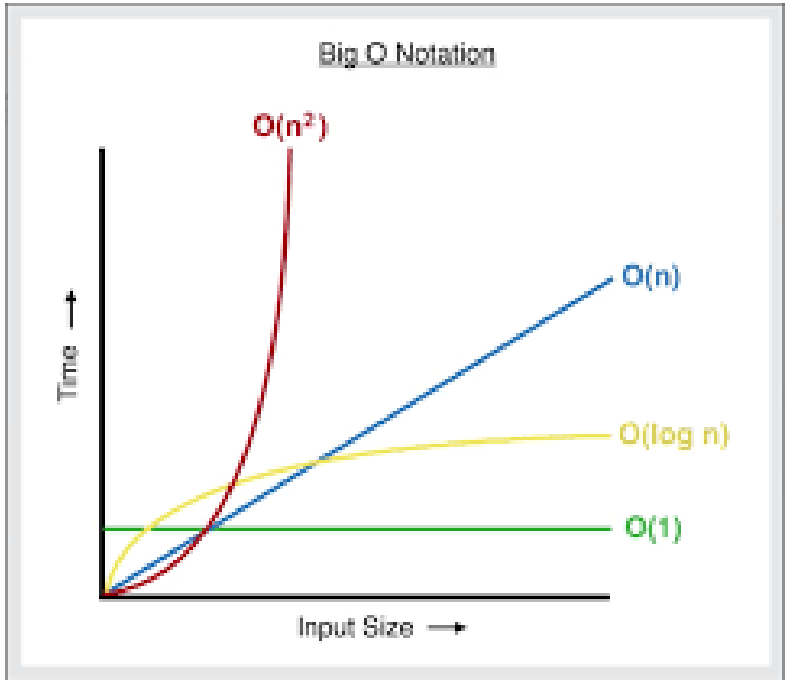
(a)



(b)

Time complexity: Big O

- The running time f(n) can be a complicated function
- The upper bound g(n) is usually a simple function like n, n log n, n^2, n^3, 2^n, n!
- We strive for *tight* upper bounds!



Time complexity: Big O

- Why is it enough for O() to only look at the dominating term in f?
- Why do the constants not matter for O()?
- *Why is all this a good idea?*

The Algorithms Toolbox

Algorithm design principles

- Greedy
- Divide and Conquer
- Dynamic Programming
- Complete Search
- Heuristics
- …
- Reductions

Design principle: Greedy

- In a way the simplest of all design principles
- Take the best alternative available, then repeat until solution found
- Straightforward idea, easy to implement with some loop(s)

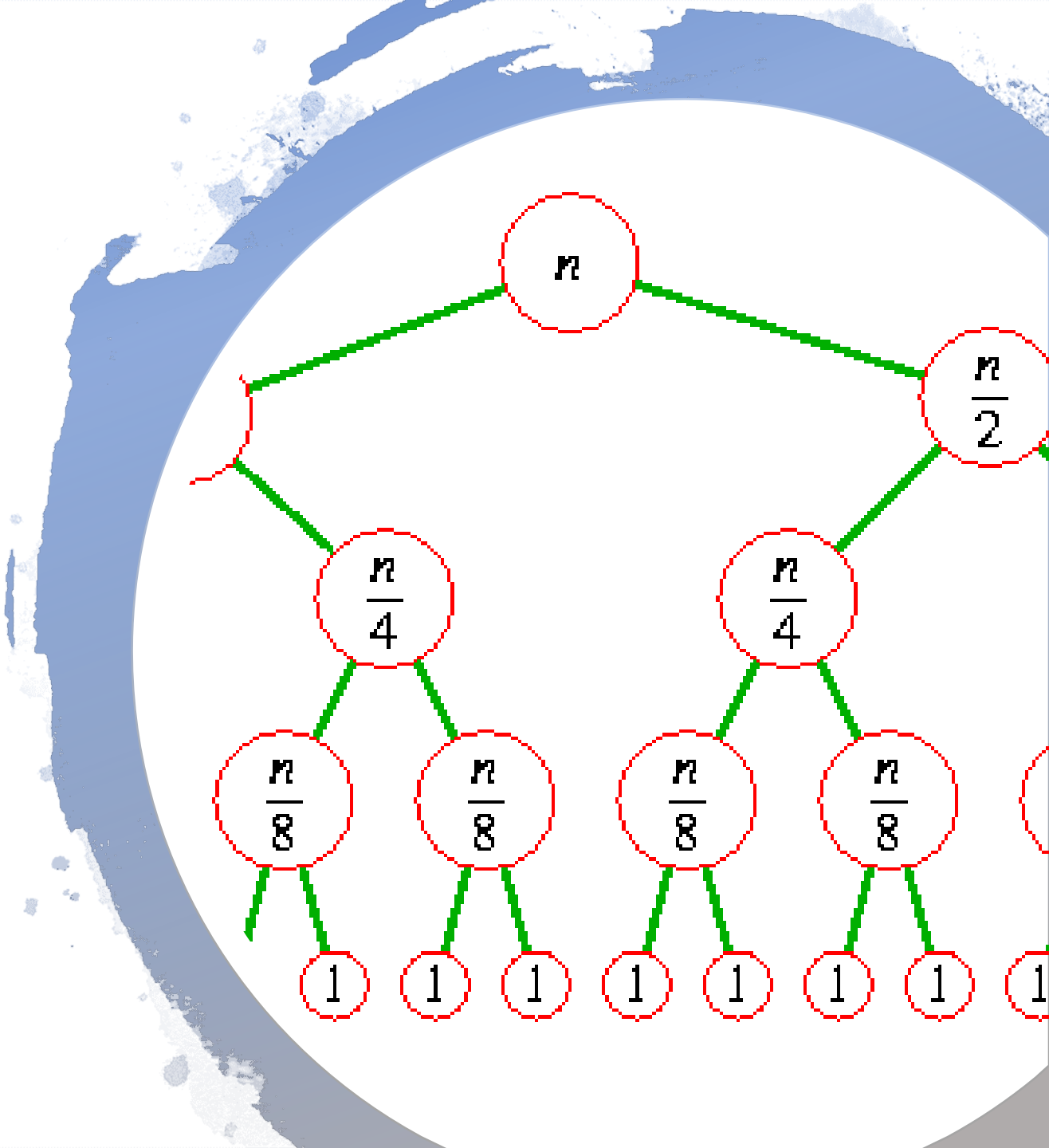
- Works only for few problems with Greedy-friendly structure
- (Can be used as a heuristic for harder problems)

- Examples: MST, Making change, simple scheduling problems

Design principle: Divide & Conquer

- Divide a probelm in (often two) parts, solve the parts independently, combine the solutions of the subproblems
- Often described in a recursive fashion

- Examples: Binary search, Mergesort, Quicksort, fast integer and matrix multiplication, many geonmetric problems (e.g. closest pair of points)



Design principle: Dynamic Programming

- Build up larger subproblems from smaller ones
- Problems need to have a certain structure (optimal subproblem principle)
- Recursive ”thinking” but efficient implementation uses loops
- Often percieved as the most difficult of the design principles

- Examples: Shortest Path, Knapsack, certain scheduling problems

More design principles

- Complete search / complete enumeration
- Branch & Bound, Backtracking (a kind of complete enumeration)
- Reductions
- Heuristics

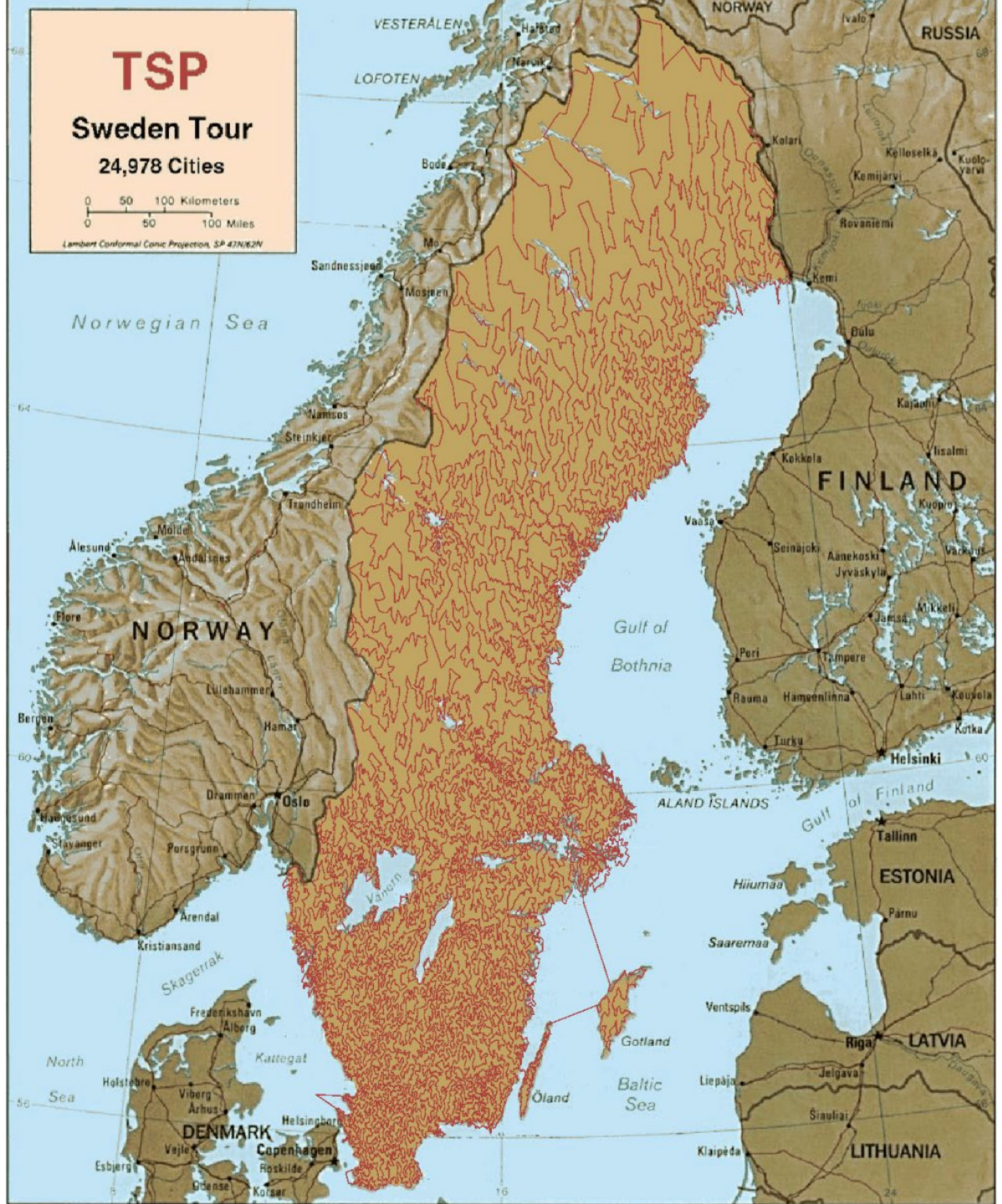
- Problem solvers usually use several design principles to attack a problem. There can be significant differences in both correctness and efficiency.

Did you watch the videos about standard problems?

Reductions and Standardproblems

- Examples for standardproblems: searching, sorting, finding an element in a list, shortest path, minimum spanning tree ..
- More: Travelling salesperson problem, graph coloring, set packing, knapsack, Integer linear programming ..

- Use reductions to re-use known algorithms
- Proof the *relative difficulty* of a problem


TSP
Sweden Tour
24,978 Cities
0 50 100 Kilometers
0 50 100 Miles
Lambert Conformal Conic Projection, SP 47N/62N
NORWAY
RUSSIA
FINLAND
SWEDEN
ESTONIA
LATVIA
LITHUANIA
DENMARK
Norwegian Sea
Gulf of Bothnia
Gulf of Finland
Baltic Sea
North Sea
Skagerrak
Kattegat
ALAND ISLANDS
Gotland
Öland
Hiiumaa
Saaremaa
VESTERÅLEN
LOFOTEN
Bodø
Narvik
Harstad
Sandnessjøen
Mosjøen
Namsos
Steinkjer
Trondheim
Molde
Ålesund
Andalsnes
Florø
Bergen
Haugesund
Stavanger
Lillehammer
Hamar
Drammen
Oslo
Porsgrunn
Arendal
Kristiansand
Frederikshavn
Ålborg
Holstebro
Viborg
Århus
Helsingborg
Vejle
Esbjerg
Copenhagen
Roskilde
Odense
Korsør
Ivalo
Kolari
Kelloselkä
Kuolo-yarvi
Kemijärvi
Rovaniemi
Kemi
Oulu
Kajaani
Iisalmi
Kokkola
Vaasa
Seinäjoki
Äänekoski
Jyväskylä
Mikkeli
Jämsä
Pori
Tampere
Rauma
Hämeenlinna
Lahti
Kouvola
Kotka
Turku
Helsinki
Tallinn
Pärnu
Ventspils
Riga
Jelgava
Liepāja
Šiauliai
Klaipėda

Algorithm design principles

- Greedy
- Divide and Conquer
- Dynamic Programming
- Complete Search
- Heuristics
- …
- Reductions
- Randomization
- LP/ILP

Jon Kleinberg Eva Tardos Algorithm Design