



Problem



Krav



Virkemåte



Styrke



Svakhet



Kjøretid



Sortering



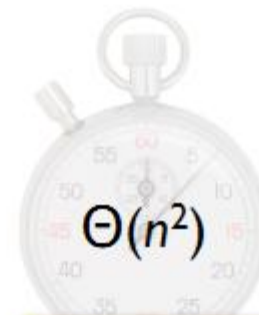
For hver i  
Sett inn blant i-1



Lavt konstantledd



Skalerer dårlig



$\Theta(n^2)$

Merk at vi kan få  $O(n)$  i best-case her, hvis vi implementerer riktig.

Hvordan kan vi få bra best-case på nesten alle algoritmer?

Sortering ved innsetting

Besøk noder

Evt. på leting



Ingen sykler

Føreløpig



Besøk deltrær

Rekursivt



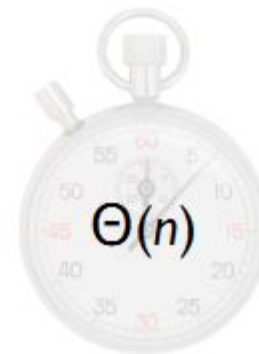
Siden vi ser bort fra  
sykler gjelder dette  
altså bare for trær.

Merk: Metodene  
her fungerer også  
for DAGs. Det eneste  
som skjer er at  
enkelte noder blir  
besøkt flere ganger.

Grundig

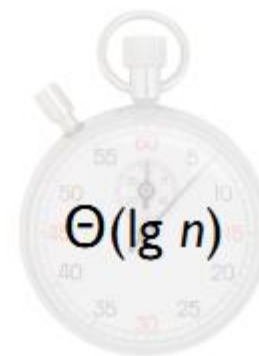


Retningsløs

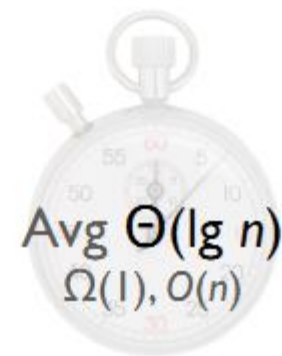


$\Theta(n)$

Traversering



Binærsøk



Søk i søketre uten balansering



Direkte oppslag

God plass



Statiske objekter



Beregn indeks

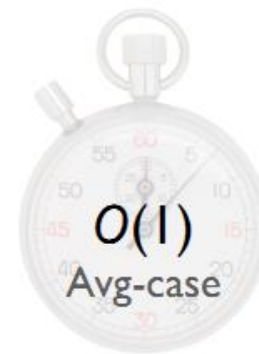
Håndter kollisjoner



Veldig kjapt



Kollisjonsfare



$O(1)$

Avg-case

Hashing



Besøk noder  
BFS: Korteste vei



Uvektede grafer



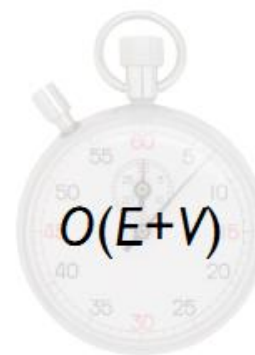
«Huskeliste»  
Oppdateres



Svært anvendelige



Ganske naive



$O(E+V)$

BFS og DFS

Kant-ensretting

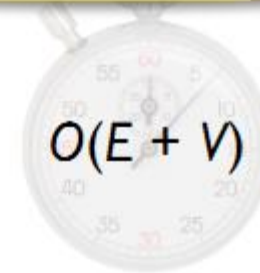


Ingen sykler



Sorter etter  $-f$   
Underveis

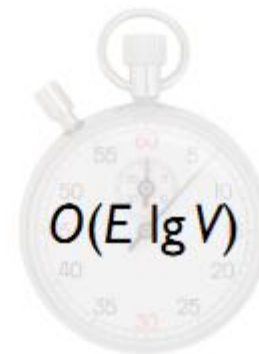
Evt. «plukk noder uten inn-  
kanter og legg dem sist».  
(DFS-varianten finner  
egentlig noder uten ut-  
kanter og legger dem først;  
blir jo det samme.)



$O(E + V)$

Topologisk sortering





Kruskals algoritme



Finn MST

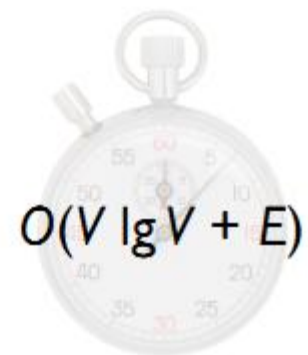


Traversering  
Neste: Kortest



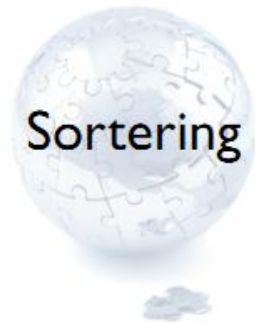
Raskest i praksis

Akkurat det er ikke  
pensum, men jeg har sett  
studier som tyder på det :-)  
(Med vanlig binær heap.)



$O(V \lg V + E)$

Prims algoritme



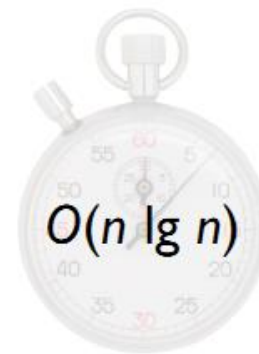
Sortering



«Selection sort»  
... med en heap



In-place, enkel



$O(n \lg n)$

Heapsort



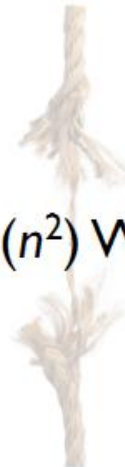
Sortering



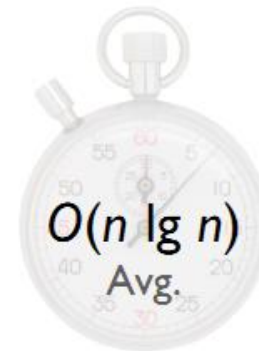
Store/små for seg  
Rekursivt



Kjapp



$O(n^2)$  WC



$O(n \lg n)$   
Avg.

Quicksort



Sortering



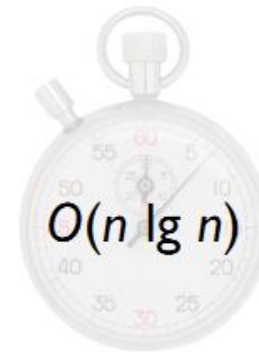
Splitt og hersk  
... og flett



Stabil



Ikke in-place



$O(n \lg n)$

Mergesort



Sorter heltall  
Eller lignende



Verdier  $O(n)$



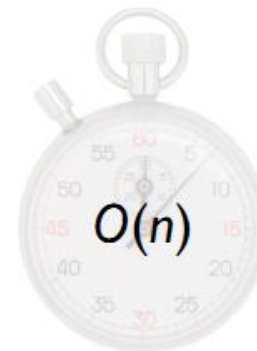
Tell forekomster  
Regn ut indeks



Lineær




Strengt krav



$O(n)$

Tellesortering



Sorter heltall  
Eller lignende



Verdier  $O(n^k)$



Sorter siffer  
Stabilt



Lineær



Ganske strenge krav



$O(kn)$

Evt. "gruntallssortering" :-)

Radikssortering



Sortering



Uniformt [0,1)



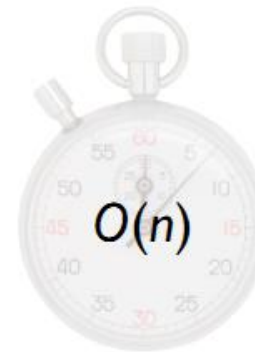
Ordnet hashing



Lineær



Krav



$O(n)$

Bøttesortering





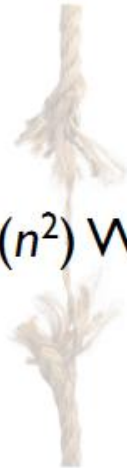
Finn median  
Og lignende



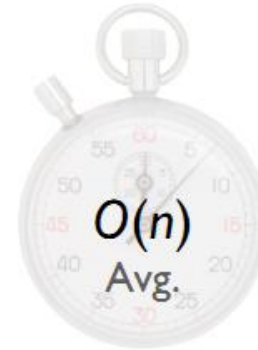
Halv Quicksort



Kjapp

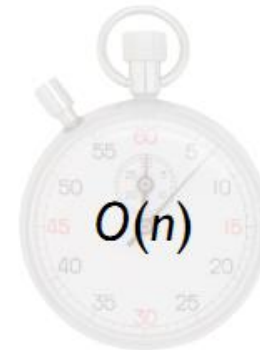


$O(n^2)$  WC



$O(n)$   
Avg.

Randomized Select



Select




Korteste vei  
Én til alle



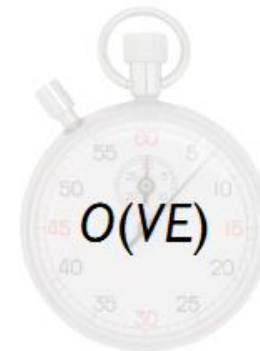
Ingen neg. sykler



Relax alle kanter  
V-I ganger



Neg. vektor  
Sykelvarsel



Bellman–Ford

**Evt. lengste!**

Korteste vei  
Én til alle



Ingen sykler

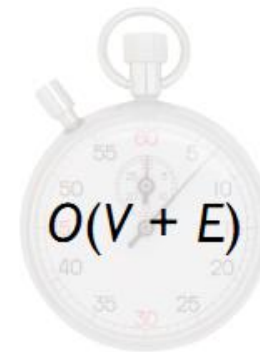
Top-sort + Relax  
... inn eller ut



Effektiv...



... men kravstor

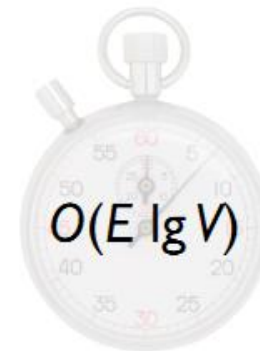


$O(V + E)$

DAG-Shortest-Path



Dette er jo en gjenganger. Jo sterkere krav vi stiller, jo mer effektive er algoritmene.



Dijkstras algoritme



Korteste vei  
Alle til alle



Gå via  $v_1 \dots v_k$   
Med eller uten  $v_k$



Floyd-Warshall



## Maks flyt

I graf m kapasiteter



Heltallsvektet: Gir oss heltallsflyt. (Selv om heltallsprogrammering generelt er NP-komplett.)



## Forøkende stier

Bruk BFS

Hvis vi kan bruke vilkårlig søk (og ikke BFS) så kan vi få eksponentiell kjøretid.

Den generelle metoden (uten at vi bestemmer traverserings-metoden) heter Ford-Fulkerson.

$O(VE^2)$

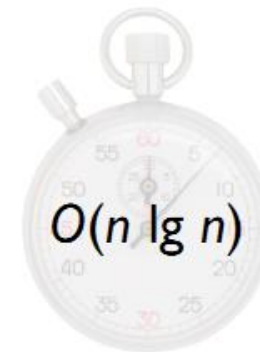
Hver sti:  $O(E)$ . Fyller minst én av  $E$  kanter. Hver kant kan fylles flere ganger (ulike retninger), men avstanden til starten langs stien må øke. Kan maks øke  $V$  ganger.

Edmonds–Karp

Min. forv. koding



Kombiner minste



Huffmans algoritme