<3 Algdat<3

Sortering ved sammenligning:

Bubblesort(boblesortering): Bytter plass på elementer, to og to av gangen.
Kjøretid $O(n^2)$, en av de minst effektive, men noe effektiv på små mengder.

Insertionsort(sortering ved innsetting): Sjekker et og et element, og sorteres i forhold til de eldre elementene.
Kjøretid $O(n^2)$, og i praksis ofte bedre enn bubblesort. Egner seg perfekt hvor man setter inn et og et element.

Heapsort (haugsortering): Lager et tre av alle elementene, hvor rotnoden er det største elementet, og man sorterer nedover. Fjerner så alltid den øverste noden, og erstatter med den ant største.
Kjøretid O(n lgn)

Quicksort: Velger et pivotelement, og splitter inn i større og mindre enn dette elementet. Fortsette å dele opp på samme måten, helt til gruppene er så små (minst 3) at man kan sortere in-place. Så setter man sammen gruppene igjen med pivotelement i midten hver gang. WorstCase: $O(n^2)$
Kjøretid O(n lgn)

Mergesort(flettesortering): Splitter i to, så det er omtrent like mange på hver side. Gjør det samme helt til man har ca 2-3 elementer igjen. Så sammenligner man to og to oppover, og fletter sammen igjen.
Kjøretid O(n lgn)

Sortering i linær tid:

Countingsort(tellesortering): Har en mengde n heltall med verdier fra 0 til k. Setter opp ei liste fra 0 til k og teller hvor mange det er av hvert element.
Kjøretid O(n)

Radixsort(radikssortering): Sorterer på det minst signifikantet sifferet først, og jobber seg mot det mest signifikantet sifferet.
Kjøretid O(kn)

Bucketsort(bøttesortering): Noe likt countingsort, men bucketsort kan også ta med seg andre rasjonelle tall. Lager intervaller, hvor elementene så plasseres i riktige «bøtter».
Kjøretid O(n), hvis tallene er uniformt fordelt. Antall bøtter må også være omtrent like store som antall elementer.

Finne i'te verdi:

Hashing:

Tildeler verdier indekser. Går rett til den angitte indeksen for å hente verdien. Kan kollidere, men er til gjengjeld veldig raskt når det stemmer.
Kjøretid O(1)

Traversering:

Generell beskrivelse for å besøke noder rekursivt. Fordel er grundig, downside er retningsløs.

Binærsøk: Søker etter bestemt tall, men må være sortert. Bruker rekursiv halvering og fungerer spesielt bra på tabell, men dårlig på liste.
Kjøretid O(lg n)

Søk i søkertre uten balansering: Søker gjennom tilfeldige verdier vha. rekursiv halvering. Bra på tilfeldige verdier, funker dårlig på sorterte verdier(da bruker man heller binærsøk).
Kjøretid Avg O(lg n), Ω(1), O(n)

Dybde-først-søk: Søker i dybde på den vektede grafen. Gå til det «dybeste» elementet, og når du er ferdig med det elementet, skriver du det opp. Gjør det samme til du er tilbake ved rotnoden. Hvis det er en node som ikke er tilkoblet, skriver du den til slutt.
Kjøretid O(E+V)

Bredde-først søk: Begynn med rotnoden og skriv opp hvor mange trinn fra rotnoden de forskjellige nodene er. Skriv så opp rekkefølgen, og hvis den er en ukoblet node, tar du ikke med den.
Kjøretid O(E+V)

Topologisk sortering: Brukes når det er ensrettede kanter, og er asyklisk. Sorter etter –f. Begynn alltid med en node uten piler til seg. Bruker DFS.
Kjøretid O(E+V)

Korteste vei-problemer:

Bellmann-Ford: Finner korteste vei, én-til-alle Håndterer negative kanter, men ikke negative sykler. Relaxer allekanter V-1 ganger. For å finne ut om man har en negativ sykel sier vi at vi har en kant fra node *u* til node *v* (*u,v*). Hvis summen av kostadene fra startnoden til u og node v er mindre enn kostnaden fra startnoden til v, er det en negativ sykel.
Kjøretid O(VE)

Dijkstra: Grådig, og finner korteste vei, én-til-alle. Tar kun positive kanter (også 0). Sjekker et og et nivå og sammenligner summen av kantene så langt. Den veien med minst kostnad blir da den raskeste. Uten heap er den søppel.
Kjøretid $O(V^2)$, eller O(V lgV) hvis man har heap.

DAG-Shortest-Path: Finner korteste vei, én-til-alle Topologisk sorterer først, for å så relaxe. Fungerer altså kun på DAG'er, så derfor er den kravstor. Husk å snu lista etter topologisk sortering!
Kjøretid O(E+V)

Floyd-Warshall: Finner korteste vei, alle-til-alle.Setter opp en nabomatrise og setter rekkefølge. Antall matriser er lik antall noder. Relax alle kanter. Hvor det ikke er en kant mellom setter vi ∞
Kjøretid $O(V^3)$, pga. V noder man skal innom, og kan variere startnoden med V-1 noder.

Minimale spenntrær:

Kruskals algoritme: Finner minimalt spenntre. Sorterer kanter fra minste til største. Kan hoppe til kanter selv om de tilhørende nodene ikke er besøkt.
Kjøretid O(E lgV)

Prims algoritme: Finner minimalt spenntre. Sorterer kanter fra minste til største. Setter en startnode, og går da alltid korteste vei til neste node. Kan kun bruke kanter som er tilkoblet noder du allerede har besøkt.
Kjøretid (V lgV+E)

Maks-flyt problemer:

Residualnettverk: Et flytnettverk som viser de kantene som tillatter mer flyt.

Flytforøkende vei: Enkel vei fra kilden *s* til sluket *t* i residualnettverket G^f.

Snitt: Partisjon av nodene V i S og T = V-S slik at kilden s ligger i S og sluket t ligger i T.

Ford-Fulkerson-metoden: Vi starter med et residualnettverk. Finner så flytforøkende vei *p* og øker flyten *f* på hver kant på *p*. Vi setter altså all den flyten den flytforøkende veien kan tåle. Deretter gjør man det samme opp igjen, helt til man ikke har flere flytforøkende veier. Da er maksimal flyt oppnådd! Grunnen til at det kalles en metode, og ikke en algoritme er fordi traverseringsmetoden ikke er bestemt, som kan ha en drastisk innvirkning på kjøretiden.
Kjøretid $O(V|f^*|)$, hvor f^* er summen av all flyten.

Edmonds-Karp-algoritmen: Om man kombinerer *Ford-Fulkerson-metoden* med *BFS* får man *Edmonds-Karp-algoritmen.*
Kjøretid $O(VE^2)$

Komprimeringsalgoritmer:

Huffmankode: Bygger et tre av de oppgitte verdiene, hvor man slår sammen to og to av de minste elementene. Setter 0 og 1 ettersom hvilken som er minst.
Kjøretid O(n lgn)

BestCase: Den beste inputen man kan få, som gir beste kjøretiden.

WorstCase: Den værste inputen man kan få, som gir værste kjøretiden.

AverageCase: Den mest normale kjøretiden man har, som krever at input'en er forventet.

O-notasjon:Øvre grense for kjøretid. Ingen kjøretider kan kjøre tregere enn dette, uansett.

Θ-notasjon: Ligger alltid mellom O og Ω. Om vi vet Θ, vet vi også O og Ω.

Ω-notasjon: Nedre grense for kjøretid. Ingen kjøretider kan kjøre raskere enn dette, uansett.

Masterteoremet:

$T(n) = aT(n/b) + f(n)$ $\quad a,b \geq 1$

Case1:

$f(n) = O(n^{\log_b a - \varepsilon})$ $\quad \varepsilon > 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a})$

Case2:

$f(n) = \Theta(n^{\log_b a})$

$\Rightarrow T(n) = \Theta(n^{\log_b a} \times \log n)$

Case3:

$f(n) = \Omega(n^{\log_b a + \varepsilon})$ $\quad \varepsilon > 0$

og $af(n/b) \leq cf(n)$ $\quad c < 1$

$\Rightarrow T(n) = \Theta(f(n))$

NPC: