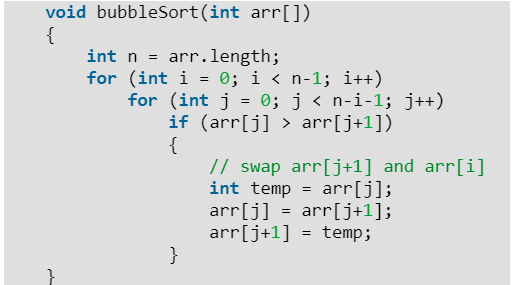
# Sortering

**Enkle**: insettingssortering, velgesortering, boblesortering

**Kompliserte**: quicksort, flettesortering, heapsort.

**Spesielle**: radixsortering, tellesortering.

**Boblesortering**

Denne sorteringsalgoritmen går gjennom tabellen n-1 ganger.

* Bytter om naboer som står feil.
* Små tall «bobler opp»
* Store tall synker ned.
* n2 kompleksitet.

**Velgesortering**

Denne algoritmen finner største verdi og setter den på plass n-1. Deretter setter den neststørste på plass n-2.

**Innsettingssortering**

Setter ett og ett element på plass blant de sorterte.

* Enkel, effektiv på små datasett.
* Spesielt god hvis data er delvis sortert fra før.
* O(n2), Ω(n)

*Når man sorterer en tabell, må man se på alle elementene for å sjekke at de står på riktig plass. Derfor vil* Ω(n) være beste kjøretid.

**Sortering ved å bytte naboer**

* Bobblesortering og velgesortering ovenfor.
* Algoritmer som bare bytter nabotall er Ω(n2).
* Invertering: to tall som står feil
  + En sortert tabell har ingen inverteringer.
  + En baklengstabell har n(n-1) / 2 inverteringer.
  + En random tabell har n(n-1) / 3 inverteringer (gjennomsnittstabell).
  + Å bytte to naboer, retter bare en invertering.
  + Å rette dette blir en kvadratisk jobb (n2) (gjennomsnittstid).

**Flettesortering**

Dette er en rekursiv algoritme som deler tabellen inn i to halvparter (del-tabeller). Disse to halvpartene sorteres rekursivt med flettesortering.

* Rekursjonen stopper når del-tabell har størrelse 1.
* Etter dette så «flettes» de sorterte deltabellene sammen til en sortert tabell.

Ulempe: trenger en ekstra tabell, som tar plass.

Kompleksitet: T(n) = 2T (n/2) + n

**Quicksort**

Den raskeste algoritmen man kjenner til, og er mest brukt.

* Plukker ut en «delingsverdi»
  + Alt som er mindre plasseres nedenfor.
  + Alt som er større plasseres ovenfor.
* Har nå tre deler:
  + Delingsverdien, som allerede står på rett plass.
  + Små verdier på den ene siden.
  + Store verdier på den andre siden.
* De to sidene sorteres rekursivt med quicksort
  + Rekursjonen stopper når en side har størrelse 1.

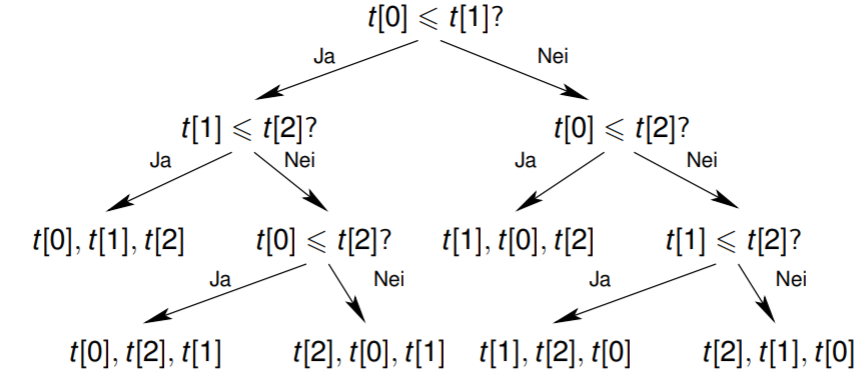
**Kompleksitet**: I verste fall får man skjevdeling, bare ett tall på den ene siden.

* Skjer *ekstremt* sjeldent.
* T(n) = n + T(n-1) + 1 => T(n) € O(n2).

I gjennomsnitt: T(n) = n + 2T(n/2)

**Sortering ved å sammenligne nøkler**

Sammenligningene kan settes opp i et binærtre. Gjelder uansett hvilken algoritme som brukes.



* Kjøretid tilsvarer høyden i treet.
* Høyden er i beste fall log2 til antall noder i treet.

**Nedre grense for kompleksitet**

* En usortert tabell med n tall har n! mulige rekkefølger.
  + Hver rekkefølge tilsvarer et endepunkt i treet (ovenfor).
  + Et tre med høyde h har max 2h endepunkter.
  + Fordi log(n!) £ O(nlogn), har vi h £ O(nlogn)
* Slik sortering blir ikke raskere enn Ω (nlogn)

**Tellesortering**

Algoritme som er raskere enn (n logn), men kan kun sortere heltall, i et intervall fra 0 til k.

* Virkemåte: teller hvor mange det fins av hvert tall og finner ut hvor mange som er <= hvert tall. Deretter bruker den denne informasjonen til å sette hvert tall på rett plass.
* Legg merke til: algoritmen sammenligner ikke tallene, den bruker tallene som index i hjelpetabellen. Det er bare heltall som kan brukes som index!

**Radixsortering**

Denne algoritmen ble først funnet opp for å sortere hullkort. Den brukes derimot enda for å oppnå rask sortering.

Virkemåte:

* Deler nøklene opp i «sifre»
  + Sorterer på minst signifikante siffer først.
  + Deretter nestminst, osv.
  + Bruker en stabil hjelpealgoritme.
* Tellesortering som hjelpealgoritme
  + O (s \* n), hvor s er antall «sifre» i nøkkel.
  + Forutsetter at «sifre» har smalere spredning enn n.