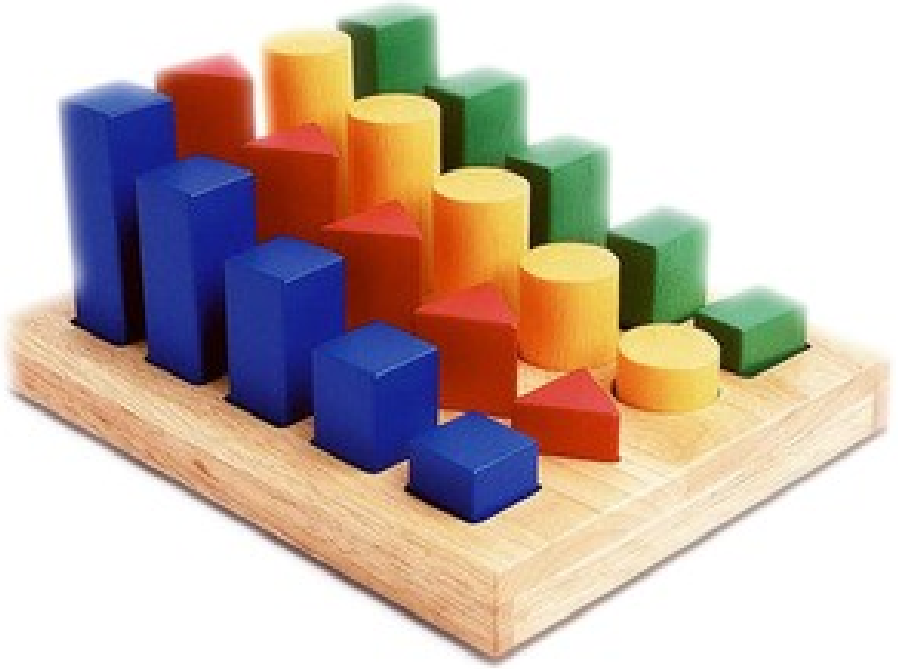
Sortering



Sorteringsproblemet

- Gitt en array *A* med *n* elementer som kan sammenlignes med hverandre:

  Finn en ordning (eller permutasjon) av elementene i *A* slik at de står i stigende (evt. avtagende) rekkefølge

- Det er viktig å kunne sortere effektivt:
  - Sortering er alltid #1 forbruker av prosessortid
  - Raske algoritmer for søking, innsetting og fjerning i datastrukturer krever oftest at dataene er sortert
  - Analyse og utvikling av effektive sorteringsalgoritmer er et sentralt forskningsområde i (klassisk) informatikk

Hvor raskt klarer vi å sortere?

- Bare det å sjekke om dataene er sortert er $O(n)$
- "Rett-frem" algoritmer er $O(n^2)$
- Hvis vi sorterer ved å sammenligne og bytte om to og to elementer, kan det bevises at dette ikke kan gjøres raskere enn $O(n \log(n))$
- "Smarte" sorteringsalgoritmer er $O(n \log(n))$
- Hvis vi *vet* mer om dataene kan vi lage $O(n)$ algoritmer som sorterer med andre mekanismer enn parvise sammenligninger og swapping*

*: F.eks. counting sort og radix sort fra kapitlet om køer.

Sortering av arrayer – forenkling

- Læreboka bruker *generiske* metoder i Java som kan sortere arrayer som inneholder "alt", så lenge dataene er `Comparable`
- For å fokusere på algoritmene og effektiviteten, og ikke på Java, forenkler vi sorteringsproblemet og programkoden til:
  - Sortering av arrayer som bare innholder heltall

Tre klasser av sorteringsalgoritmer

- Sekvensielle – sorterer fra starten av array:
  - Bruker typisk to løkker inne i hverandre, $O(n^2)$
  - Utplukksortering, innstikksortering, bubble sort (Shell sort?)
- Logaritmiske:
  - Deler arrayen i to deler, sorterer disse rekursivt, $O(n \log(n))$
  - Quicksort, flettesortering
- Basert på bruk av spesielle datastrukturer:
  - Alle dataene legges inn i en rask datastruktur og tas ut igjen i sortert rekkefølge, typisk $O(n \log(n))$
  - Treesort, heapsort (kommer senere i kurset)