IN2010 uke 2-3

Jakob Hansen
jakobkha@uio.no

https://github.com/jakobkhansen/IN2010

Ting vi kan gå gjennom

- Bli litt kjent, corona info, gruppetimens struktur
- Big O notasjon + oppgaver
- Binærsøk 🔎 + Evt livekode binærsøk
- Trær 🌲 + Evt livekode binært tre
- Traversering av binære trær
- Balanserte trær (ukens tema)

DU SKAL TIL ENHVER TID…

Ha god håndhygiene, vasking og antibac

Holde avstand = 1 m

… Og føler du deg syk skal du gå hjem/holde deg hjemme

HVIS JEG FØLER MEG SYK

GÅ HJEM UMIDDEL-BART / HOLD DEG HJEMME

Kontakt helsevesenet gjennom fastlege / koronatelefon – de avklarer om testing er nødvendig

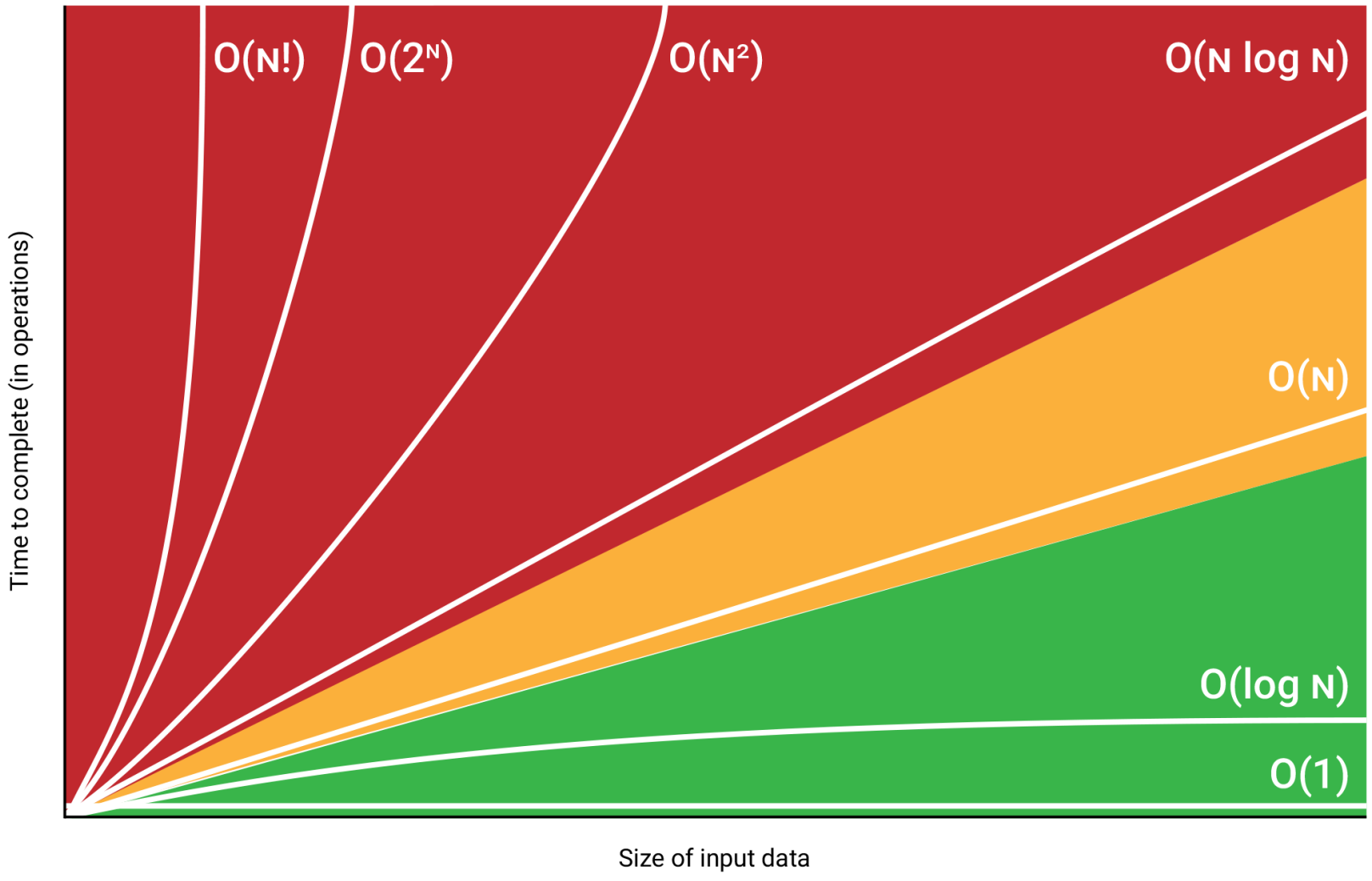
UiO er i kontakt med helsevesenet og omvendt

Big O notasjon

- Hva er målet med Big O?
- Analysere kjøretid! Hvilken algoritme er raskest? (Grovt)
- Abstrahere bort små forskjeller
- (Primitive operasjoner -> generelt)

```
1         boolean numExists(int[] array, int numToFind) {
2
3           // Itererer over array, hvis vi finner tallet, true
4           for (int i = 0; i < array.length; i++) {
5             if (array[i] == numToFind) {
6               return true;
7             }
8           }
9
10          // Ingen elementer er lik tallet, false
11          return false;
12        }
```

Big O notasjon



Konstant tid

- $O(1)$
- Tar samme tid uansett
- Eksempel: Hente første element av et array
- $O(1000) = O(1)$

Lineær tid

- $O(n)$
- Vokser direkte med input størrelse
- Eksempel: Iterere og printe ut hvert element i et array
- $O(100n) = O(n)$

Polynomiell tid

- For hver n, for hver n...
- $O(n^x)$, for eksempel $O(n^2)$
- Eksempel: To løkker som sjekker om det finnes en duplikat i arrayet.
- Eksempel: “Bruteforce“ en kodelås

Logaritmisk tid

- O(log(n))
- Litt tricky
- $log_2(n) = x$ hvis $2^x = n$
- Antall ganger du må halvere n for å få 1
- Eksempel: Lete i telefonbok, Binary search

Tregere enn polynomiell tid

- Begynner å bli for tregt til å utnyttes
- $O(2^n)$, $O(n!)$
- Som oftest naive løsninger der man prøver alle muligheter
- For $n = 100$ er $O(n!)$ lenger enn universets levetid

Algorithm Loop1(n):

$s \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
    $s \leftarrow s + i$

Algorithm Loop2(n):

$p \leftarrow 1$
for $i \leftarrow 1$ **to** $2n$ **do**
    $p \leftarrow p \cdot i$

Algorithm Loop4(n):

$s \leftarrow 0$
for $i \leftarrow 1$ **to** $2n$ **do**
    for $j \leftarrow 1$ **to** i **do**
        $s \leftarrow s + i$

```
1  for (int i = n; i > 1; i = i/2) {
2    System.out.println(i);
3  }
```

```
1  for (int i = 0; i < n; i++) {
2    for (int j = 0; j < n; j++) {
3      System.out.println(i+j);
4    }
5  }
```

Søk

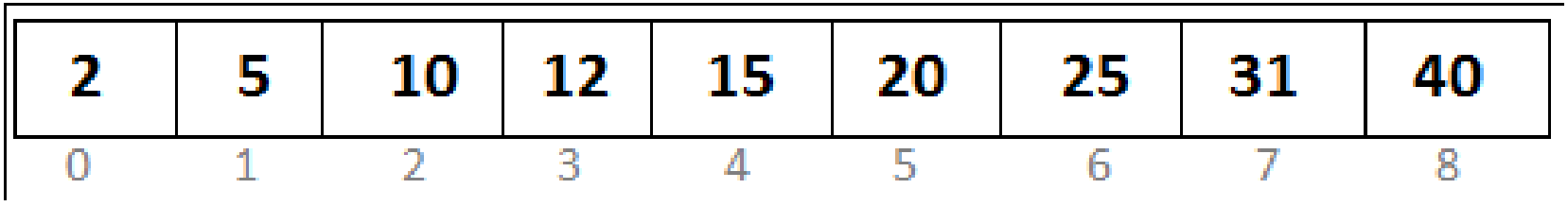
Hvordan finne ut om et element eksisterer i en array?

```java
// Den enkle måten å sjekke om et array inneholder et tall
public boolean numExists(int[] array, int numToFind) {

  // Itererer over array, hvis vi finner tallet, true
  for (int i = 0; i < array.length; i++) {
    if (array[i] == numToFind) {
      return true;
    }
  }

  // Ingen elementer er lik tallet, false
  return false;
}
```

- Er det mulig å gjøre dette raskere hvis arrayet er sortert?
- Binary Search 🤯

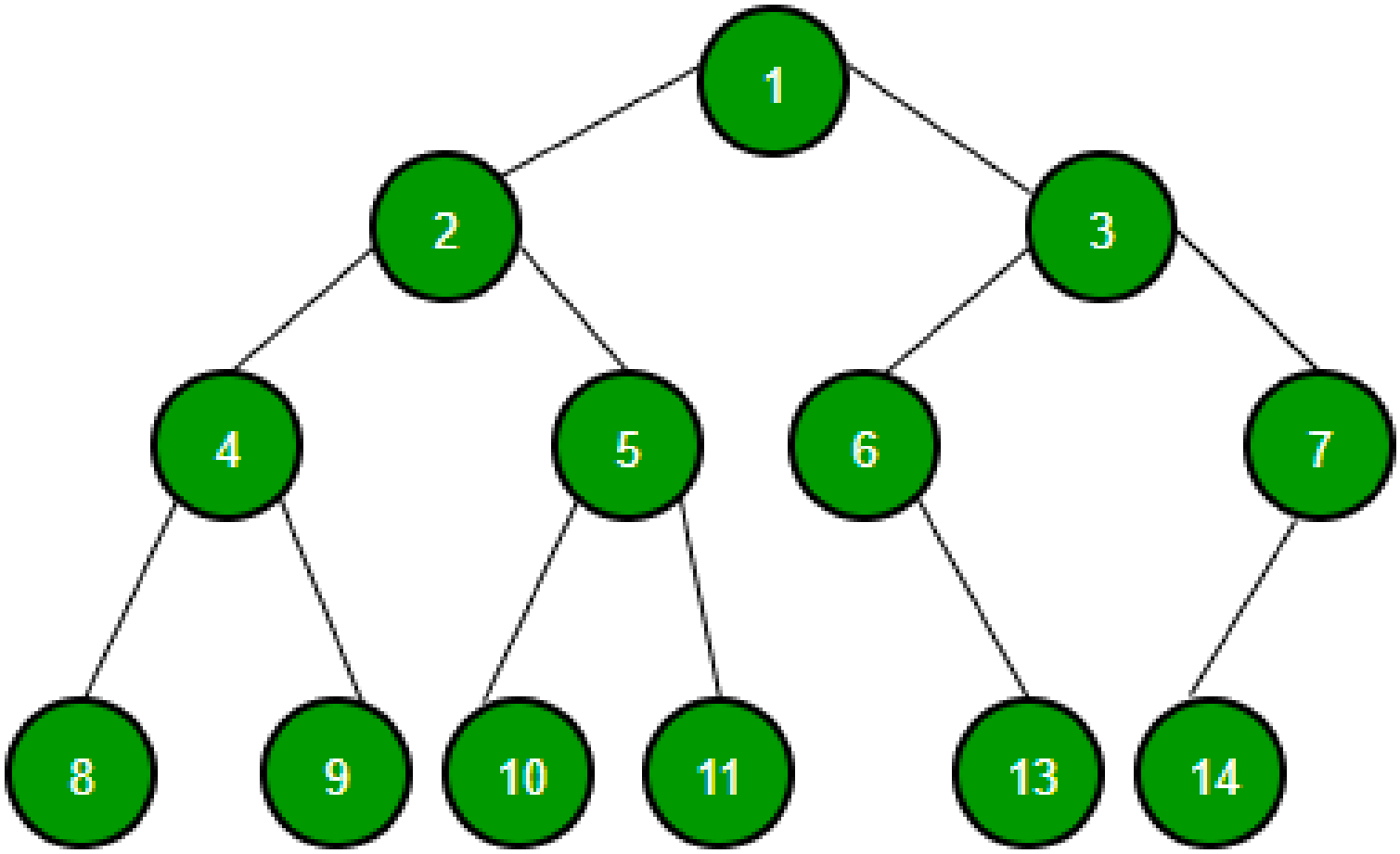
Binary Search

- Hva vet vi om et element i et sortert array?
- Alle elementer til venstre er mindre, alle elementer til høyre er større!
- Når vi vet dette, hvordan kan vi utnytte det?
- Sjekk midten, eliminer halve arrayet hver gang!


2
5
10
12
15
20
25
31
40
0
1
2
3
4
5
6
7
8

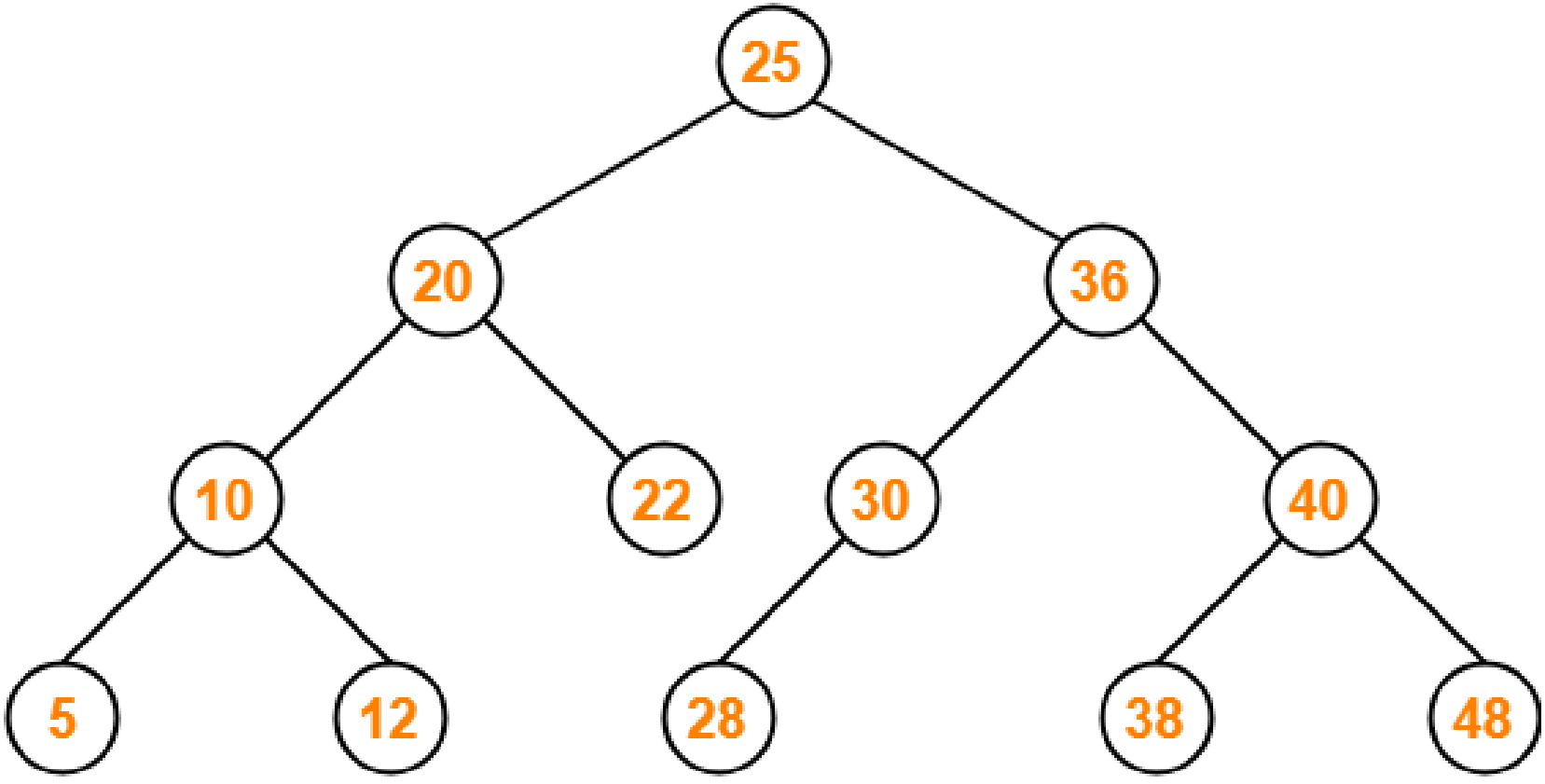
Trær

- Datastruktur som ser ut som et opp ned tre
- Består av et sett med noder, som har andre noder som barn
- Noder kan ha en verdi, objekt, etc knyttet til seg
- Noen regler på hva som er lov og ikke



Binære søketrær

- Binært tre med tallverdier som følger reglene til et tre
- Ekstra regel!
  - Alle noder i høyre subtre til en node må være større
  - Alle noder i venstre subtre til en node må være mindre
- Bruker flere operasjoner på binære søketre
  - Sette inn
  - Slette
  - Finne et tall



Binary Search Tree

Balanserte trær

- Worst case for innsetting/finne i binært søketre -> $O(n)$
- Worst case for innsetting/finne i et komplett binært tre ->$O(log(n))$
- Vi ønsker så balanserte trær som mulig!
- AVL trær
- Rød-svarte trær

Rotasjoner

Flytt på noder, slik at subtreet får 1 mindre høyde

AVL-trær

- Selvbalanserende tre der enhver node har en “balanseverdi“ og en høydeverdi
- Høydeverdi = max(venstrepeker.høyde, høyrepeker.høyde) + 1
  NB: hvis en node er null, har den høydeverdi -1
- Balanseverdi = venstrepeker.høyde - høyrepeker.høyde
  (evt motsatt)

Innsetting i AVL-trær

- Sett inn som et vanlig BST (rekursivt traverser treet nedover til en null)
- Når kallene terminerer, oppdater høydeverdier og sjekk balanse
- Hvis $|\text{balansen}| > 1$, roter
- Kan maks skje 1 rotasjon per innsetting

Sletting i AVL-trær

- Slett som i et vanlig BST
- Når kallene terminerer, oppdater høydeverdier og sjekk balanse
- Hvis $|\text{balansen}| > 1$, roter (Roterer på andre siden)
- Kan skje flere rotasjoner per sletting

Mine tips for å lykkes i IN2010

- IN2010 er et modningsfag, sett av nok tid, jobb jevnt.
- Visualisering er key, utnytt online ressurser for det.
- Mål: implementer alle algoritmene som er pensum og forstå kompleksiteten
- Fokuser på forståelse, ikke pugging.
- Diskuter med andre, hjelp andre, få hjelp av andre.
- Ekstra: Jobb med problemløsning ved siden av (Kattis osv)