

# UOFFISIELL MIDTSEMESTERFORELESNING

Algoritmer og datastrukturer - TDT4120  
Høst 2020

Henrik Hørlück Berg og Theodor Astrup Wiik

18. oktober 2020

# Tips og triks - Hva du bør gjøre når du ikke har meg

- ▶ Se forelesningene
- ▶ Les boka, ofte nødvendig siden algoritmer har ulike definisjoner, det er den som står i boka dere skal kunne.
- ▶ Sjekk pensumheftet! Der står det **alt** du skal kunne
- ▶ Ikke se rett på løsningsforslaget, prøv å løse oppgaven selv.
- ▶ Usikker? Spør studass!



Pensumhefte, 2020

Algoritmer og datastrukturer



# Asymptotisk notasjon - Kort fortalt

## Viktig å få med seg

1. Vi ser på forskjeller i en kosmisk skala, når  $n$  blir *veldig* stor. Detaljer er ikke viktig, vi har en konstant  $c$  som kan styres.

# Asymptotisk notasjon - Kort fortalt

## Viktig å få med seg

1. Vi ser på forskjeller i en kosmisk skala, når  $n$  blir *veldig* stor. Detaljer er ikke viktig, vi har en konstant  $c$  som kan styres.
2. O-ene betegner *klasser* med funksjoner. Disse kan ha ulike tolkninger. F.eks. kan  $\Omega(n^2)$  tolkes som  $n^2$ ,  $n!$  eller  $n^3 + \pi$ .

# Asymptotisk notasjon - Kort fortalt

## Viktig å få med seg

1. Vi ser på forskjeller i en kosmisk skala, når  $n$  blir *veldig* stor. Detaljer er ikke viktig, vi har en konstant  $c$  som kan styres.
2. O-ene betegner *klasser* med funksjoner. Disse kan ha ulike tolkninger. F.eks. kan  $\Omega(n^2)$  tolkes som  $n^2$ ,  $n!$  eller  $n^3 + \pi$ .
3. Notasjonsmisbruk:  $f(n) = O(g(n))$  betyr *egentlig*  $f(n) \in O(g(n))$ , altså at  $f(n)$  er en tolkning av klassen. Dette er *ikke* symmetrisk<sup>1</sup>, så vi kan ikke nødvendigvis skrive  $g(n) = O(f(n))$ .  
Tenk selv: hva om  $f(n) = n^2$  og  $g(n) = n^3$ ?

---

<sup>1</sup>Unntatt for  $\Theta$

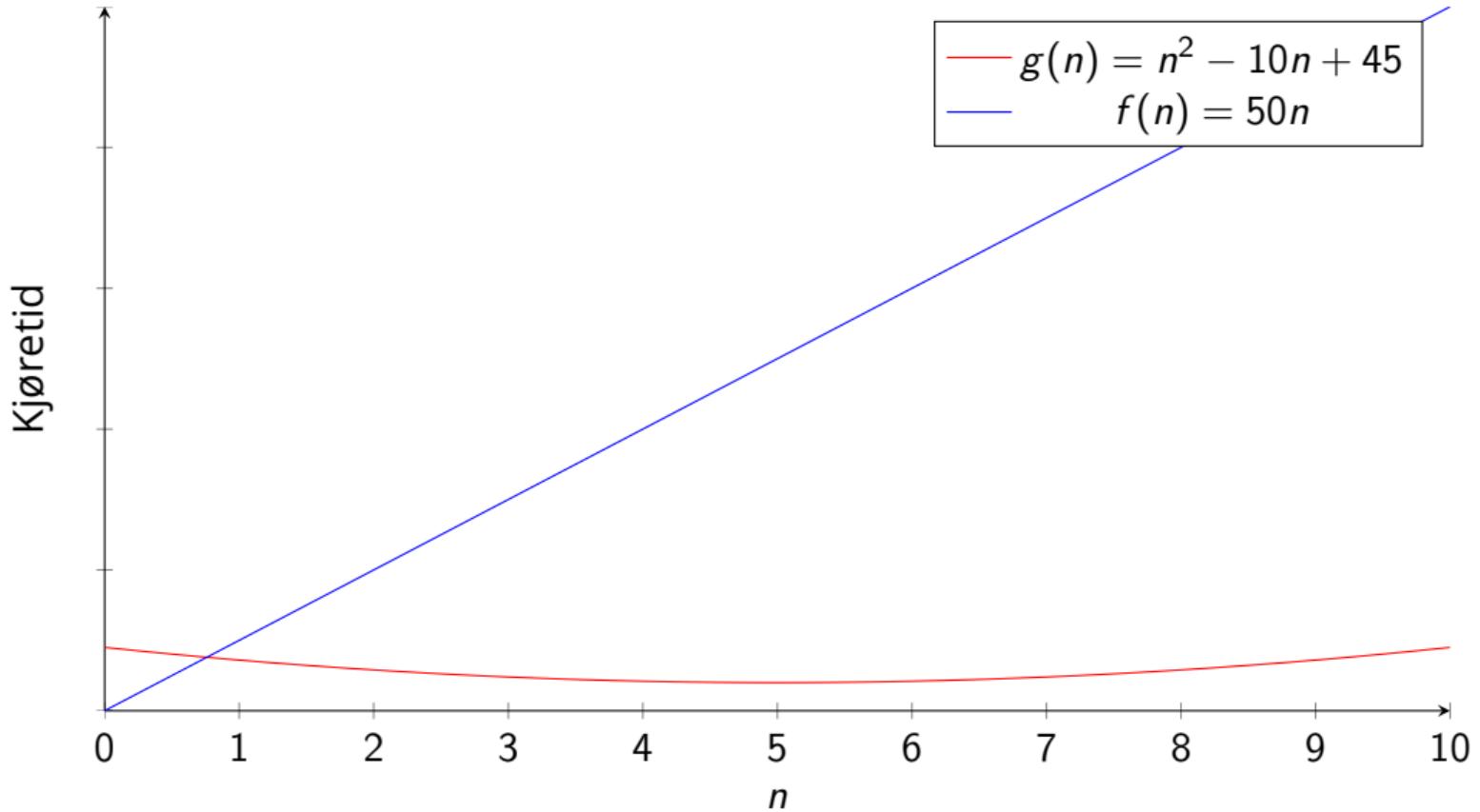
# Asymptotisk notasjon - Kort fortalt

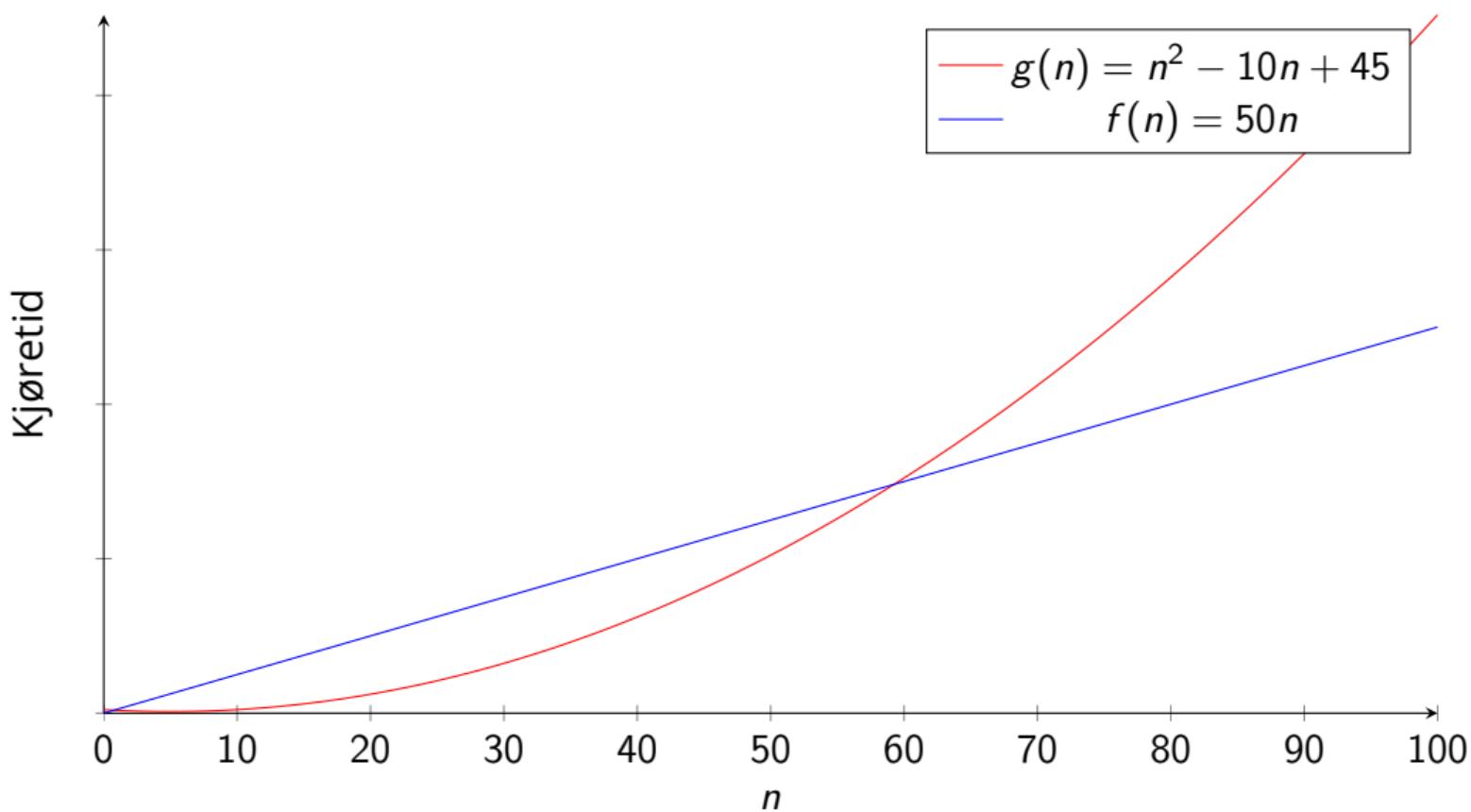
## Viktig å få med seg

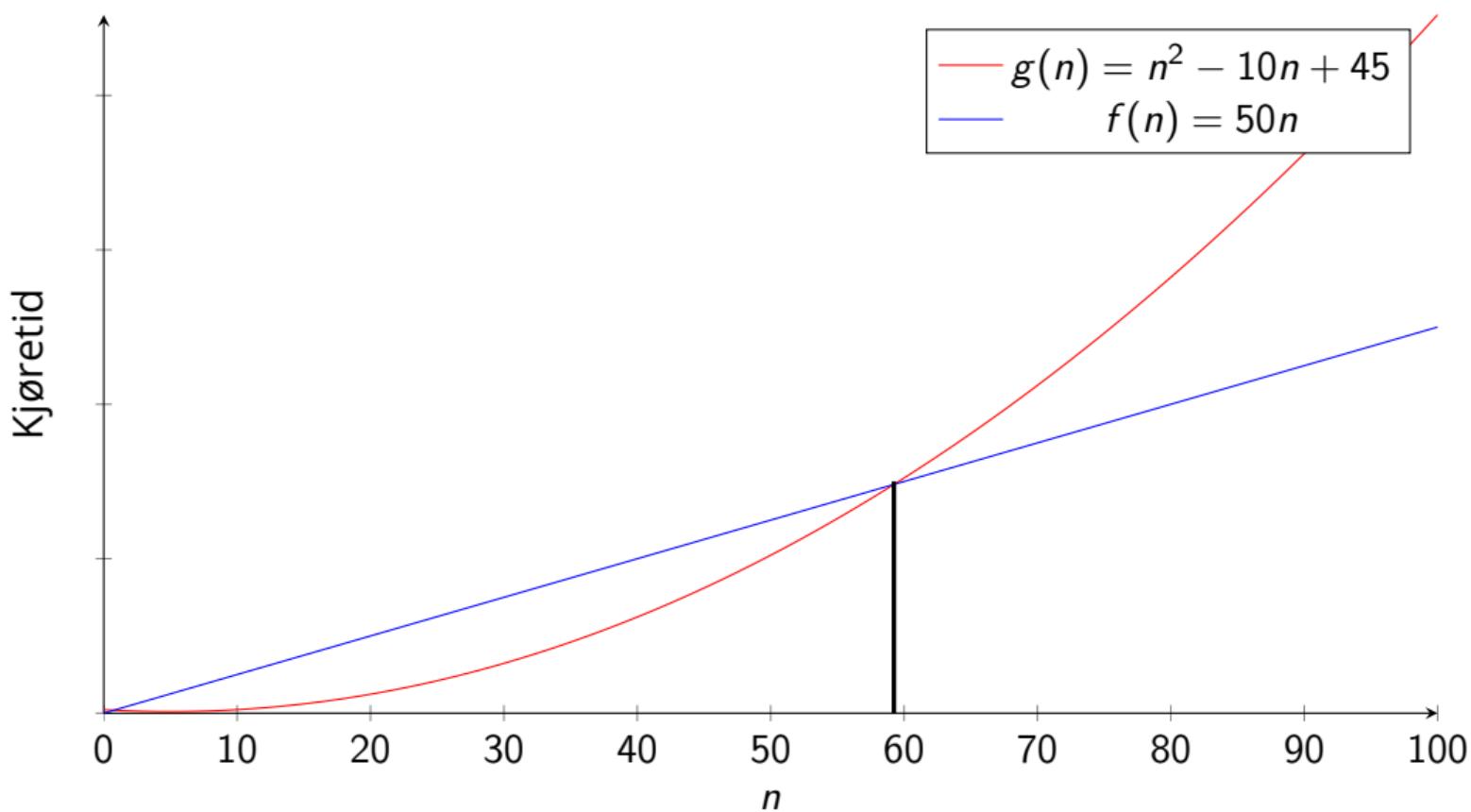
1. Vi ser på forskjeller i en kosmisk skala, når  $n$  blir *veldig* stor. Detaljer er ikke viktig, vi har en konstant  $c$  som kan styres.
2. O-ene betegner *klasser* med funksjoner. Disse kan ha ulike tolkninger. F.eks. kan  $\Omega(n^2)$  tolkes som  $n^2$ ,  $n!$  eller  $n^3 + \pi$ .
3. Notasjonsmisbruk:  $f(n) = O(g(n))$  betyr *egentlig*  $f(n) \in O(g(n))$ , altså at  $f(n)$  er en tolkning av klassen. Dette er *ikke* symmetrisk<sup>1</sup>, så vi kan ikke nødvendigvis skrive  $g(n) = O(f(n))$ .  
Tenk selv: hva om  $f(n) = n^2$  og  $g(n) = n^3$ ?
4. Selv om det er flere ledd, så er det kun den som vokser raskest som teller.  
Eksempel:  $O(3 + 4n + n^4 + n! + \log^{10} n) \equiv O(n!)$

---

<sup>1</sup>Unntatt for  $\Theta$







## Asymptotisk notasjon - Kan se på det som intervaller

Kan erstatte  $n^2$  her med en hver funksjon  $f(n)$ . Merk forskjell mellom inklusiv og ekslusiv endepunkt.

$$\omega(n^2) = (n^2, \infty)$$

$$\Omega(n^2) = [n^2, \infty)$$

$$\Theta(n^2) = [n^2, n^2]$$

$$O(n^2) = [1, n^2]$$

$$o(n^2) = [1, n^2)$$

Da blir å legge sammen uttrykk det samme som å finne det intervallet som går fra største nedre grense til største øvre grense<sup>2</sup>

---

<sup>2</sup>Kun når du er ute etter det mest presise, ellers blir det *alle* intervaller som inkluderer den grensen vi beskrev.

# Asymptotisk notasjon - Oppgave 2 H2017

(5%) a) Forenkle uttrykket  $\Theta(n) + O(n^2) + \Omega(n^3)$

(5%) b) Forenkle uttrykket  $\Theta(n + \sqrt{n}) + O(n^2 + n) + \Omega(n \cdot (n + \lg n))$

# Asymptotisk notasjon - Oppgave 2 H2017

**(5%) a) Forenkle uttrykket**  $\Theta(n) + O(n^2) + \Omega(n^3)$

► Vi får  $[n, n] + [1, n^2] + [n^3, \infty) = [n^3, \infty] = \Omega(n^3)$

**(5%) b) Forenkle uttrykket**  $\Theta(n + \sqrt{n}) + O(n^2 + n) + \Omega(n \cdot (n + \lg n))$

# Asymptotisk notasjon - Oppgave 2 H2017

**(5%) a) Forenkle uttrykket**  $\Theta(n) + O(n^2) + \Omega(n^3)$

- ▶ Vi får  $[n, n] + [1, n^2] + [n^3, \infty) = [n^3, \infty] = \Omega(n^3)$

**(5%) b) Forenkle uttrykket**  $\Theta(n + \sqrt{n}) + O(n^2 + n) + \Omega(n \cdot (n + \lg n))$

- ▶ Vi forenkler:  $\Theta(n + \sqrt{n}) \equiv \Theta(n)$ ,  $O(n^2 + n) \equiv O(n^2)$ ,  $\Omega(n \cdot (n + \lg n)) \equiv \Omega(n^2)$

# Asymptotisk notasjon - Oppgave 2 H2017

**(5%) a) Forenkle uttrykket**  $\Theta(n) + O(n^2) + \Omega(n^3)$

- ▶ Vi får  $[n, n] + [1, n^2] + [n^3, \infty) = [n^3, \infty] = \Omega(n^3)$

**(5%) b) Forenkle uttrykket**  $\Theta(n + \sqrt{n}) + O(n^2 + n) + \Omega(n \cdot (n + \lg n))$

- ▶ Vi forenkler:  $\Theta(n + \sqrt{n}) \equiv \Theta(n)$ ,  $O(n^2 + n) \equiv O(n^2)$ ,  $\Omega(n \cdot (n + \lg n)) \equiv \Omega(n^2)$
- ▶ Vi får  $[n, n] + [1, n^2] + [n^2, \infty) = [n^2, \infty] = \Omega(n^2)$

# Asymptotisk notasjon - Oppgave 10 Høst 2019

(5%) Hva er  $O(n) + \Omega(n) + \Theta(n) + o(n) + \omega(n)$

# Asymptotisk notasjon - Oppgave 10 Høst 2019

**(5%) Hva er  $O(n) + \Omega(n) + \Theta(n) + o(n) + \omega(n)$**

- ▶  $\Omega(n)$  kan bli vilkårlig stor, men  $\omega(n)$  har et strengere minste-krev enn både  $\Theta(n)$  og  $\Omega(n)$ , og dominerer derfor. Svar:  $\omega(n)$

# Rekurrens - Motivasjon

## Når funksjoner kaller på seg selv

Mystisk( $n$ )

1   **if**  $n > 1$

2           Mystisk( $n - 1$ )

- ▶ Hvordan passer det inn i notasjonen vi nettopp lærte?

# Rekurrens - Motivasjon

## Når funksjoner kaller på seg selv

Mystisk( $n$ )

1   **if**  $n > 1$

2           Mystisk( $n - 1$ )

- ▶ Hvordan passer det inn i notasjonen vi nettopp lærte?
- ▶ Om en setter opp en funksjon, så får vi  $f(n) = f(n - 1) + 1$   
(Merk: vi må alltid sjekke if-en).

# Rekurrens - Motivasjon

## Når funksjoner kaller på seg selv

Mystisk( $n$ )

1   **if**  $n > 1$

2           Mystisk( $n - 1$ )

- ▶ Hvordan passer det inn i notasjonen vi nettopp lærte?
- ▶ Om en setter opp en funksjon, så får vi  $f(n) = f(n - 1) + 1$   
(Merk: vi må alltid sjekke if-en).
- ▶ Hva blir kjøretiden?

# Rekurrens - Muligheter

## Når funksjoner kaller på seg selv

Vi har noen verktøy

- ▶ Iterasjonsmetoden
- ▶ Substitusjon
- ▶ Rekursjonstre
- ▶ Masterteoremet
- ▶ Variabelskifte

# Rekurrens - Iterasjonsmetoden

## Eksempeloppgave

La oss finne kjøretiden til funksjonen i forrige slide!

Mystisk( $n$ )

1   **if**  $n > 1$

2              Mystisk( $n - 1$ )

Vi setter opp en rekurrensrelasjon:  $T(n) = T(n - 1) + 1$

Hvor mange ganger må vi kalle Mystisk før vi når grunntilfellet  $T(1) = 1$ ?

---

# Rekurrens - Iterasjonsmetoden

## Eksempeloppgave

La oss finne kjøretiden til funksjonen i forrige slide!

Mystisk( $n$ )

1   **if**  $n > 1$

2       Mystisk( $n - 1$ )

Vi setter opp en rekurrensrelasjon:  $T(n) = T(n - 1) + 1$

Hvor mange ganger må vi kalle **Mystisk** før vi når grunntilfellet  $T(1) = 1$ ?

Løsning: Vi reduserer  $n$  med 1 hver gang, så vi må løse

$n - k \cdot 1 = 1 \rightarrow k = n - 1$ .<sup>3</sup> Så vi gjennomfører det konstante arbeidet  $n$  ganger.

Til sammen:  $T(n) = n$ , som er  $\Theta(n)$

---

<sup>3</sup>Hvorvidt vi utfører  $n$  eller  $n - 1$  arbeid spiller sjeldent en rolle

# Rekurrens - Substitusjon

## Kan vi ikke bare gjette oss frem?

Vi har fått et rekurrens  $T(n) = 2T(n/2) + n$

Jeg gjetter at løsningen ser ut som dette:

$$T(k) = k \lg k + k, \text{ for } k < n$$

# Rekurrens - Substitusjon

## Kan vi ikke bare gjette oss frem?

Vi har fått et rekurrens  $T(n) = 2T(n/2) + n$

Jeg gjetter at løsningen ser ut som dette:

$$T(k) = k \lg k + k, \text{ for } k < n$$

- ▶ Hvordan kan vi sjekke dette?

# Rekurrens - Substitusjon

## Kan vi ikke bare gjette oss frem?

Vi har fått et rekurrens  $T(n) = 2T(n/2) + n$

Jeg gjetter at løsningen ser ut som dette:

$$T(k) = k \lg k + k, \text{ for } k < n$$

- ▶ Hvordan kan vi sjekke dette?
- ▶ Induksjon!

# Rekurrens - Rekursjonstrær

## Iterasjonsmetoden med flere grener

- ▶ Fin måte å visualisere arbeidet
- ▶ Kan være overveldene i starten

# Rekurrens - Rekursjonstrær

## Iterasjonsmetoden med flere grener

- ▶ Fin måte å visualisere arbeidet
- ▶ Kan være overveldene i starten
- ▶ Vi itererer gjennom hva algoritmen koster for hvert nivå i treet, og legger det sammen, altså får vi en sum (kostnad løvnivå +  $\sum_{i=0}^{h-1}$  pris for nivå  $i$ )

# Rekurrens - Rekursjonstrær

## Iterasjonsmetoden med flere grener

- ▶ Fin måte å visualisere arbeidet
- ▶ Kan være overveldene i starten
- ▶ Vi itererer gjennom hva algoritmen koster for hvert nivå i treet, og legger det sammen, altså får vi en sum (kostnad løvnivå +  $\sum_{i=0}^{h-1}$  pris for nivå  $i$ )
- ▶ Kjøretiden blir høyden til treet er hvor lang tid det tar å komme seg til grunntilfellet. Kjøretiden blir summen av kostnaden til hvert nivå

# Rekurrens - Rekursjonstrær

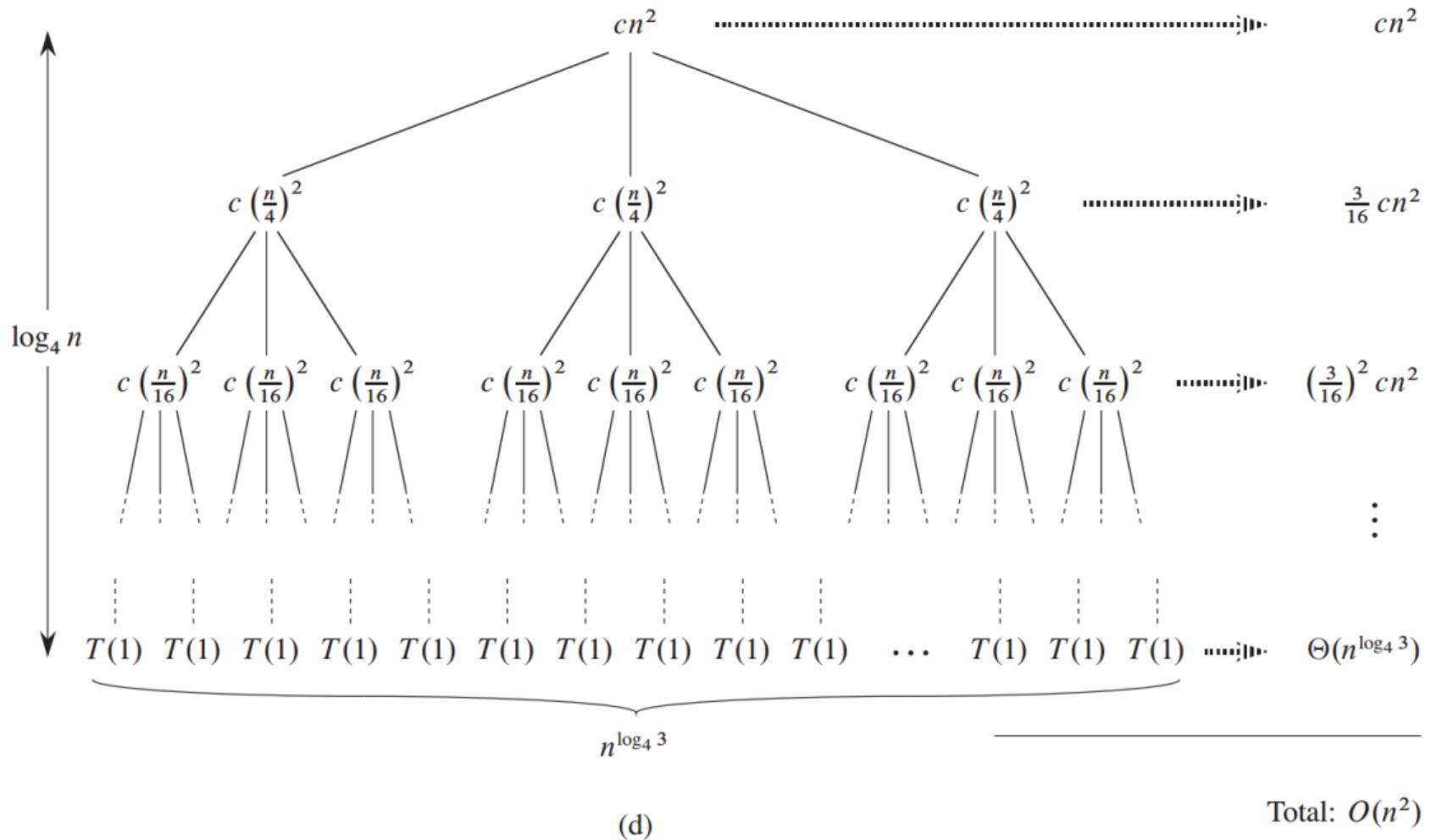
## Iterasjonsmetoden med flere grener

- ▶ Fin måte å visualisere arbeidet
- ▶ Kan være overveldene i starten
- ▶ Vi itererer gjennom hva algoritmen koster for hvert nivå i treet, og legger det sammen, altså får vi en sum (kostnad løvnivå +  $\sum_{i=0}^{h-1}$  pris for nivå  $i$ )
- ▶ Kjøretiden blir høyden til treet er hvor lang tid det tar å komme seg til grunntilfellet. Kjøretiden blir summen av kostnaden til hvert nivå
- ▶ Kan ofte bruke masterteoremet i stedet (men det bevises via rekurrensstrær)

# Rekurrens - Rekursjonstrær

## Iterasjonsmetoden med flere grener

- ▶ Fin måte å visualisere arbeidet
- ▶ Kan være overveldene i starten
- ▶ Vi itererer gjennom hva algoritmen koster for hvert nivå i treet, og legger det sammen, altså får vi en sum (kostnad løvnivå +  $\sum_{i=0}^{h-1}$  pris for nivå  $i$ )
- ▶ Kjøretiden blir høyden til treet er hvor lang tid det tar å komme seg til grunntilfellet. Kjøretiden blir summen av kostnaden til hvert nivå
- ▶ Kan ofte bruke masterteoremet i stedet (men det bevises via rekurrensstrær)
- ▶ Eksempel:  $T(n) = 3T(n/4) + cn^2 = 3T(n/4) + \Theta(n^2)$ . (s. 88 i boka)



# Rekurrens - Rekursjonstrær

## Vi regner på det

- ▶ For detaljer se boka
- ▶ For å telle noder på bunnen, benytter oss av  $a^{\log b} = b^{\log a}$  (samme som vi benytter oss av i Masterteoremet)

# Rekurrens - Rekursjonstrær

## Vi regner på det

- ▶ For detaljer se boka
- ▶ For å telle noder på bunnen, benytter oss av  $a^{\log b} = b^{\log a}$  (samme som vi benytter oss av i Masterteoremet)
- ▶ Antall noder ved høyde  $h$  er  $3^h$

# Rekurrens - Rekursjonstrær

## Vi regner på det

- ▶ For detaljer se boka
- ▶ For å telle noder på bunnen, benytter oss av  $a^{\log b} = b^{\log a}$  (samme som vi benytter oss av i Masterteoremet)
- ▶ Antall noder ved høyde  $h$  er  $3^h$
- ▶ Teller antall noder på løvnivå:  $3^{\log_4 n} = n^{\log_4 3}$ , ved å anta konstant grunntilfelle får vi at kjøretiden blir  $\Theta(n^{\log_4 3})$

# Rekurrens - Rekursjonstrær

## Vi regner på det

- ▶ For detaljer se boka
- ▶ For å telle noder på bunnen, benytter oss av  $a^{\log b} = b^{\log a}$  (samme som vi benytter oss av i Masterteoremet)
- ▶ Antall noder ved høyde  $h$  er  $3^h$
- ▶ Teller antall noder på løvnivå:  $3^{\log_4 n} = n^{\log_4 3}$ , ved å anta konstant grunntilfelle får vi at kjøretiden blir  $\Theta(n^{\log_4 3})$
- ▶ Vi får følgende uttrykk:  $T(n) = \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$

# Rekurrens - Rekursjonstrær

## Vi regner på det

- ▶ For detaljer se boka
- ▶ For å telle noder på bunnen, benytter oss av  $a^{\log b} = b^{\log a}$  (samme som vi benytter oss av i Masterteoremet)
- ▶ Antall noder ved høyde  $h$  er  $3^h$
- ▶ Teller antall noder på løvnivå:  $3^{\log_4 n} = n^{\log_4 3}$ , ved å anta konstant grunntilfelle får vi at kjøretiden blir  $\Theta(n^{\log_4 3})$
- ▶ Vi får følgende uttrykk:  $T(n) = \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$
- ▶ Regn ut selv!

# Rekurrens - Rekursjonstrær

## Vi regner på det

- ▶ For detaljer se boka
- ▶ For å telle noder på bunnen, benytter oss av  $a^{\log b} = b^{\log a}$  (samme som vi benytter oss av i Masterteoremet)
- ▶ Antall noder ved høyde  $h$  er  $3^h$
- ▶ Teller antall noder på løvnivå:  $3^{\log_4 n} = n^{\log_4 3}$ , ved å anta konstant grunntilfelle får vi at kjøretiden blir  $\Theta(n^{\log_4 3})$
- ▶ Vi får følgende uttrykk:  $T(n) = \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$
- ▶ Regn ut selv!
- ▶ Hint: kan sammenligne med uendelig geometrisk rekke, hvordan påvirker det hvilken stor-O vi kan bruke?

# Rekurrens - Masterteoremet

## Master theorem

Let  $a \geq 1$  and  $b < 1$  be constants, let  $f(n)$  be a function, and let  $T(n)$  be defined on nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n).$$

where we interpret  $n/b$  to mean either  $\lfloor n/b \rfloor$  or  $\lceil n/b \rceil$ . Then  $T(n)$  has the following asymptotic bounds:

1. If  $f(n) = O(n^{\log_b a - \epsilon})$  for some constant  $\epsilon > 0$ , then  $T(n) = \Theta(n^{\log_b a})$ .
2. If  $f(n) = \Theta(n^{\log_b a})$ , then  $T(n) = \Theta(n^{\log_b a} \lg n)$ .
3. If  $f(n) = \Omega(n^{\log_b a + \epsilon})$  for some constant  $\epsilon > 0$ , and if  $af(n/b) \leq cf(n)$  for some constant  $c < 1$  and all sufficiently large  $n$ , then  $T(n) = \Theta(f(n))$ .

# Rekurrens - Masterteoremet

## Hva gjør vi?

- ▶ Vi sammenligner veksten mellom funksjon på rotnivå og kjøretiden til løvnivået i treet
  1. Treet dominerer
  2. De er like
  3. Funksjonen dominerer, men OBS vi må sjekke at  $f(n)$  «vokser normalt», slik at treet ikke egentlig dominerer
- ▶ Vi finner ut av hva som dominerer, og dermed får vi en kjøretid
- ▶ Finnes mer generelle utgaver som takler flere rekurrens, disse er *ikke* pensum
- ▶ Flere tilfeller hvor vi ikke kan bruke det, står i boka.  
(Sjeldent det dukker opp i oppgaver)

# Rekurrens - Variabelskifte

## Bytt ut vanskelige uttrykk

- ▶ Fremgangsmåte:
  1. Sett opp ny rekurrens av en annen variabel, f.eks.  $T(n) = T(2^k) = S(k)$
  2. Løs den, f.eks. med masterteoremet
  3. Bytt tilbake<sup>4</sup> til  $T(n)$ , i vårt tilfelle blir det å sette  $k = \log_2 n$  siden  $2^{\log_2 n} = n$ .  
Husk logaritme-base spiller sjeldent en rolle
- ▶ Øving 3 har mye bra her

---

<sup>4</sup>Kan også benyttes dersom vi ikke har  $T(n)$  på venstre side, da slipper vi å bytte tilbake

# Rekurrens - Oppgave 3 K2015

(5%) Løs rekurrensen  $T(n) = T(\sqrt{n}) + \lg n$

# Rekurrens - Oppgave 3 K2015

(5%) Løs rekurrensen  $T(n) = T(\sqrt{n}) + \lg n$

1. Hint:  $m = \lg n$

# Rekurrensen - Oppgave 3 K2015

(5%) Løs rekurrensen  $T(n) = T(\sqrt{n}) + \lg n$

1. Hint:  $m = \lg n$
2. Hint:  $S(m) = T(2^m)$

# Rekurrenser - Å se kjøretid fra kode

## Må øves på

Gitt at vi har en funksjon FunctionN, som har kjøretid  $\Theta(n)$ , hva blir kjøretiden til disse?

MystiskA( $n$ )

```
1  if  $n > 1$ 
2    FunctionN( $n$ )
3     $a = \text{Mystisk}(n/2)$ 
4     $b = \text{Mystisk}(n/2)$ 
5     $c = \text{Mystisk}(n/2)$ 
6    return  $a + b + c$ 
```

Vi får rekurrens  $T(n) = 3T(n/2) + \Theta(n)$ , siden vi kaller funksjonen tre ganger.

MystiskB( $n$ )

```
1  if  $n > 1$ 
2    FunctionN( $n$ )
3    return  $3 \cdot \text{Mystisk}(n/2)$ 
```

Vi får rekurrens  $T(n) = T(n/2) + \Theta(n)$ , vi kaller bare funksjonen en gang!

# *Pause*

*ta en kopp kaffe*



# Analyse - Algoritmeanalyse

## Må øves på

Når vi regnet rekurrensen, så antok vi ingenting om hva  $n$  var, så vi fant den generelle kjøretiden. Denne kjøretiden kan derimot variere veldig, basert på hvordan input ser ut.

- ▶ Best-case, da antar vi noe om hvordan problem-instansene ser ut, utover størrelsen på data-en
- ▶ Worst-case, igjen vi antar noe om problem-instansene
- ▶ Average-case, da kan vi anta en sannsynlighetsfordeling for input, f.eks. vi antar at vi vanligvis får nesten-sorterte lister. Da må vi vite mer om nøyaktig hva algoritmen brukes til
- ▶ Amortisert analyse: Da ser vi på gjennomsnitt av kjøretiden over flere kall av algoritmen, men vi antar ingenting om problem-instansen. Eksempel

# Analyse - Amortisert Analyse

Innsetting i std::vector, eller Java sin ArrayList

Table-Insert( $T, x$ )

- 1 **if**  $T.size == 0$
- 2     allocate  $T.table$  with 1 slot
- 3      $T.size == 1$
- 4 **if**  $T.num == T.size$
- 5     allocate *new-table* with  $2 \cdot T.size$  slots
- 6     insert all items in  $T.table$  into *new-table*
- 7     free  $T.table$
- 8      $T.table = new-table$
- 9      $T.size = 2 \cdot T.size$
- 10 insert  $x$  into  $T.table$
- 11  $T.num = T.num + 1$

# Sortering - Egenskaper

## Forskjellige algoritmer, ulike egenskaper

- ▶ Sammenligningsbasert («generell») eller ikke («spesialisert»)
- ▶ Best / Average / Worst case kjøretid
- ▶ Minnebruk - in-place
- ▶ Stabil: dersom du sorterer en liste med like elementer, havner de på samme plass?
- ▶ Finnes mange flere, som ikke er relevant for dette faget:
  - ▶ Paralleliserbar
  - ▶ Online — betyr at du løser problemet mens du kun har fått deler av det, tenk få ett og ett tall vs. hele listen.

# Sortering - Digresjon

## Hvorfor lærer vi masse sortering

- ▶ I praksis bruker vi oftest bare en innebygd `.sort()`-metode
- ▶ Forskjellige måter å bryte ned et problem på
- ▶ Disse måtene har forskjellige egenskaper
- ▶ Dere skal lære mye mer enn det jeg får fortalt i dag, spesielt å konstruere nye algoritmer, det må en øve på selv! Ikke se på LF først!
- ▶ Sortering er en naturlig del av mange andre problemer

# Sortering - Læringsmål

**Minner om at følgende gjelder for *alle* algoritmer i pensum**

- ▶ Kjenne den formelle definisjonen av det generelle problemet den løser

# Sortering - Læringsmål

**Minner om at følgende gjelder for *alle* algoritmer i pensum**

- ▶ Kjenne den formelle definisjonen av det generelle problemet den løser
- ▶ Kjenne til eventuelle tilleggskrav den stiller for å være korrekt

# Sortering - Læringsmål

**Minner om at følgende gjelder for *alle* algoritmer i pensum**

- ▶ Kjenne den formelle definisjonen av det generelle problemet den løser
- ▶ Kjenne til eventuelle tilleggskrav den stiller for å være korrekt
- ▶ Vite hvordan den oppfører seg; kunne utføre algoritmen, trinn for trinn

# Sortering - Læringsmål

## Minner om at følgende gjelder for *alle* algoritmer i pensum

- ▶ Kjenne den formelle definisjonen av det generelle problemet den løser
- ▶ Kjenne til eventuelle tilleggskrav den stiller for å være korrekt
- ▶ Vite hvordan den oppfører seg; kunne utføre algoritmen, trinn for trinn
- ▶ ! Forstå korrekthetsbeviset; hvordan og hvorfor virker algoritmen egentlig?

# Sortering - Læringsmål

## Minner om at følgende gjelder for *alle* algoritmer i pensum

- ▶ Kjenne den formelle definisjonen av det generelle problemet den løser
- ▶ Kjenne til eventuelle tilleggskrav den stiller for å være korrekt
- ▶ Vite hvordan den oppfører seg; kunne utføre algoritmen, trinn for trinn
- ▶ ! Forstå korrekthetsbeviset; hvordan og hvorfor virker algoritmen egentlig?
- ▶ Kjenne til eventuelle styrker eller svakheter, sammenlignet med andre

# Sortering - Læringsmål

## Minner om at følgende gjelder for *alle* algoritmer i pensum

- ▶ Kjenne den formelle definisjonen av det generelle problemet den løser
- ▶ Kjenne til eventuelle tilleggskrav den stiller for å være korrekt
- ▶ Vite hvordan den oppfører seg; kunne utføre algoritmen, trinn for trinn
- ▶ ! Forstå korrekthetsbeviset; hvordan og hvorfor virker algoritmen egentlig?
- ▶ Kjenne til eventuelle styrker eller svakheter, sammenlignet med andre
- ▶ Kjenne kjøretidene under ulike omstendigheter, og forstå utregningen

# Sortering - Læringsmål

## Minner om at følgende gjelder for *alle* algoritmer i pensum

- ▶ Kjenne den formelle definisjonen av det generelle problemet den løser
- ▶ Kjenne til eventuelle tilleggskrav den stiller for å være korrekt
- ▶ Vite hvordan den oppfører seg; kunne utføre algoritmen, trinn for trinn
- ▶ ! Forstå korrekthetsbeviset; hvordan og hvorfor virker algoritmen egentlig?
- ▶ Kjenne til eventuelle styrker eller svakheter, sammenlignet med andre
- ▶ Kjenne kjøretidene under ulike omstendigheter, og forstå utregningen

**Altså:** Det finnes cheat-sheets, men en må likevel forstå algoritmen til den grad at du kan skrive den i pseudo-kode-form!

# Sortering - Insertion sort

Slik vi vanligvis sorterer en kortstokk.

Sammenligning	Ja
Best-case	$\Theta(n)$
Average-case	$\Theta(n^2)$
Worst-case	$\Theta(n^2)$
Minne	$O(1)$
In-place	Ja
Stabil	Ja

## Sortering - Merge sort

En liste med ett element er sortert, vi kan så flette disse sammen raskt.

Sammenligning	Ja
Best-case	$\Theta(n \lg n)$
Average-case	$\Theta(n \lg n)$
Worst-case	$\Theta(n \lg n)$
Minne	$O(n)$
In-place	Nei
Stabil	Ja

## Sortering - Quicksort

- ▶ Velg et pivot-element, splitt tabellen i to, en halvdel er større, andre er mindre.
- ▶ Splitt og hersk.
- ▶ Problem: allerede sortert liste, og du velger største element.  
Løsning: tilfeldig pivot!

(«Pivot» betyr «dreie», så elementet blir der vi bytter mellom mindre enn og større enn)

# Sortering - Quicksort

Sammenligning	Ja
Best-case	$\Theta(n \lg n)$
Average-case	$\Theta(n \lg n)$
Worst-case	$\Theta(n^2)$
Minne	$O(\lg n)$
In-place	Ja
Stabil	Nei

# Sortering - Heapsort

Lag en max-heap – hent største element – reduser størrelsen på heapen og gjør den til en heap igjen – repeat

Sammenligning	Ja
Best-case	$\Theta(n \lg n)$
Average-case	$\Theta(n \lg n)$
Worst-case	$\Theta(n \lg n)$
Minne	$O(1)$
In-place	Ja
Stabil	Nei

# Sortering - Sorteringsgrense

## Sorteringsgrensen

Any comparison sort algorithm requires  $\Omega(n \lg n)$  comparisons in the worst case.

## Konsekvenser

Det er umulig å sortere raskere, uten å anta egenskaper ved problemet. Bevis i boka.

## Sortering - Counting sort

Anta at du kun kan ha  $k$  distinkte elementer. Tell hvor mange instanser det er av hvert element, gjør tellingen om til indeks. Flytt elementer til en ny liste basert på indeksen du fant.

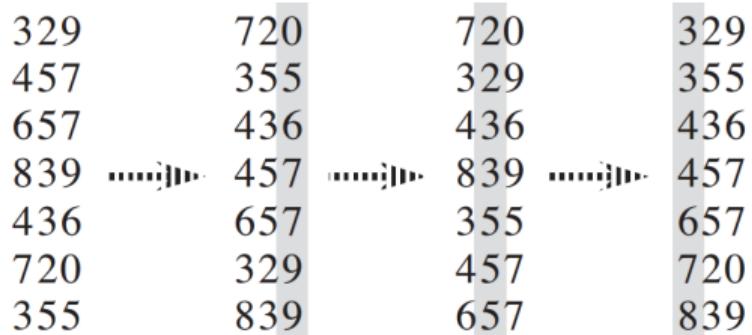
Sammenligning	Nei
Best-case	$\Theta(n + k)$
Average-case	$\Theta(n + k)$
Worst-case	$\Theta(n + k)$
Minne	$O(n + k)$
In-place	Nei
Stabil	Ja

# Sortering - Radix sort

Sorter stabilt på hvert siffer.

Bruker counting-sort behind the scenes. Utnytter at det er et begrenset antall siffer.

Sammenligning	Nei
Best-case	$\Theta(d(n + k))$
Average-case	$\Theta(d(n + k))$
Worst-case	$\Theta(d(n + k))$
Minne	$O(n + k)$
In-place	Nei
Stabil	Ja

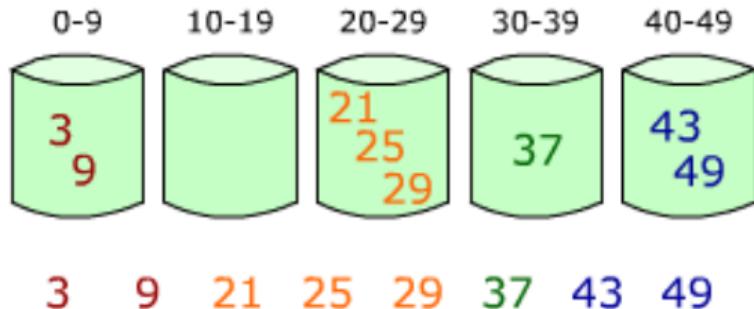


## Sortering - Bucket sort

Uniformt fordelt tallene i bøtter, sorter  
så i hver bøtte og kombiner. NB: krever  
uniform sannsynlighetsfordeling!

\*: avhenger av sub-rutine

Sammenligning	Nei*
Best-case	$\Theta(n)$
Average-case	$\Theta(n)$
Worst-case	$\Theta(n^2)$
Minne	$O(n)$
In-place	Nei
Stabil	Ja*



# Sortering - Oppgave 2 Høst 2011

## (7%) Oppgave c

En venn av deg påstår han har utviklet en generell prioritetskø der operasjonene for å legge til et element, å finne maksimum og å ta ut maksimum alle har kjøretid  $O(1)$  i verste tilfelle. Forklar hvorfor dette ikke kan stemme.

# Sortering - Oppgave 2 Høst 2011

## (7%) Oppgave c

En venn av deg påstår han har utviklet en generell prioritetskø der operasjonene for å legge til et element, å finne maksimum og å ta ut maksimum alle har kjøretid  $O(1)$  i verste tilfelle. Forklar hvorfor dette ikke kan stemme.

- ▶ Løsning: Vi kan brukke dette til å sortere i  $O(n)$  tid generelt, det bryter med fartsgrensen vår.

# Sortering - Oppgave 1 Høst 2010

## (5%) Oppgave g

Heapsort er optimal, men Radix-Sort har bedre asymptotisk kjøretid. Forklar svært kort hvordan dette henger sammen.

# Sortering - Oppgave 1 Høst 2010

## (5%) Oppgave g

Heapsort er optimal, men Radix-Sort har bedre asymptotisk kjøretid. Forklar svært kort hvordan dette henger sammen.

- ▶ Løsning: Heapsort løser det generelle sorteringsproblemet (sammenligningsbasert). Radix-Sort gjør antagelser.

# Sortering - Oppgave 3 Høst 2014

## (5%) Oppgave a

Du ønsker å sortere sekvensen  $A = (a_1, a_2, \dots, a_n)$ . Det er velkjent at for sammenligningsbasert sortering er  $\Omega(n \log n)$  den beste kjøretiden vi kan få, i forventet (*average-case*) og verste tilfelle.

Anta at elementene er reelle tall, distribuert etter en gitt sannsynlighetsfordeling, som kan beregnes i konstant tid for ethvert element. Hva er den beste forventede kjøretiden vi kan få, og hvordan kan du oppnå den? Forklart kort.

# Sortering - Oppgave 3 Høst 2014

## (5%) Oppgave a

Du ønsker å sortere sekvensen  $A = (a_1, a_2, \dots, a_n)$ . Det er velkjent at for sammenligningsbasert sortering er  $\Omega(n \log n)$  den beste kjøretiden vi kan få, i forventet (*average-case*) og verste tilfelle.

Anta at elementene er reelle tall, distribuert etter en gitt sannsynlighetsfordeling, som kan beregnes i konstant tid for ethvert element. Hva er den beste forventede kjøretiden vi kan få, og hvordan kan du oppnå den? Forklart kort.

- ▶ Løsning: Spesialtilfelle av Bucket-sort

Lykke til, dette klarer du!

