

DM507 Algoritmer og datastrukturer

Introduktion til kurset

Rolf Fagerberg

Forår 2020

Hvem er vi?

Underviser:

- ▶ Rolf Fagerberg, Institut for Matematik og Datalogi (IMADA)
Forskningsområde: algoritmer og datastrukturer

Hvem er vi?

Underviser:

- ▶ Rolf Fagerberg, Institut for Matematik og Datalogi (IMADA)
Forskningsområde: algoritmer og datastrukturer

Deltagere:

- ▶ BA i Datalogi
- ▶ BA i Software Engineering
- ▶ BA i Matematik-Økonomi
- ▶ BA i Anvendt Matematik
- ▶ BA sidefag i Datalogi
- ▶ KA sidefag i Data Science

Hvem er vi?

Underviser:

- ▶ Rolf Fagerberg, Institut for Matematik og Datalogi (IMADA)
Forskningsområde: algoritmer og datastrukturer

Deltagere:

- ▶ BA i Datalogi
- ▶ BA i Software Engineering
- ▶ BA i Matematik-Økonomi
- ▶ BA i Anvendt Matematik
- ▶ BA sidefag i Datalogi
- ▶ KA sidefag i Data Science

Stor diversitet: forskellige semestre (2./4./6./8.) i uddannelsen, forskellige mængder af programmering og af matematiske fag på uddannelsen.

Kursets format

Kursets format

Forudsætninger:

Programmering i Java eller Python, lidt matematisk modenhed

Kursets format

Forudsætninger:

Programmering i Java eller Python, lidt matematisk modenhed

Format:

Forelæsninger (I-timer). Ofte 3 x 30 min.

Opgaveregning (E-timer). Med instruktør.

Arbejde selv og i studiegrupper

Kursets format

Forudsætninger:

Programmering i Java eller Python, lidt matematisk modenhed

Format:

Forelæsninger (I-timer). Ofte 3 x 30 min.

Opgaveregning (E-timer). Med instruktør.

Arbejde selv og i studiegrupper

Eksamenform:

Skriftlig eksamen (juni), 8 ECTS:

Multiple-choice (med bøger, noter, computer). Karakter efter 7-skala. Mål: check af kendskab til stoffet. [NB: reeksamen er mundtlig.]

Projekt undervejs, 2 ECTS:

I flere dele. Karakter B/IB. Skal ikke (længere) bestå for at gå til skriftlig eksamen. Mål: træne overførsel af stoffet til praksis (programmering).

Materialer

Lærebog:

Cormen, Leiserson, Rivest, Stein:
Introduction to Algorithms, 3rd edition, 2009.

Materialer

Lærebog:

Cormen, Leiserson, Rivest, Stein:
Introduction to Algorithms, 3rd edition, 2009.

Materialer

Lærebog:

Cormen, Leiserson, Rivest, Stein:
Introduction to Algorithms, 3rd edition, 2009.

Andet læremateriale på kursets webside:

- Slides fra forelæsninger
- Links til videoer
- Opgaver til øvelsestimer
- Tidligere eksamenssæt
- Projektet

Materialer

Lærebog:

Cormen, Leiserson, Rivest, Stein:
Introduction to Algorithms, 3rd edition, 2009.

Andet læremateriale på kursets webside:

- Slides fra forelæsninger
- Links til videoer
- Opgaver til øvelsestimer
- Tidligere eksamenssæt
- Projektet

Stoffet findes i fuld detalje i *tre* udgaver: lærebog, slides, video. Brug de dele, som virker bedst for dig (gerne flere dele).

Forventet arbejdsindsats

- ▶ Skim stof før forelæsning: 0,5 timer
- ▶ Forelæsning: 2 timer
- ▶ Læs stof efter forelæsning: 1,5 timer
- ▶ Opgaveregning (forberedelse): 3 timer
- ▶ Opgaveregning (klasse): 2 timer

Forventet arbejdsindsats

- ▶ Skim stof før forelæsning: 0,5 timer ⇐ **mindst vigtig**
- ▶ Forelæsning: 2 timer
- ▶ Læs stof efter forelæsning: 1,5 timer
- ▶ Opgaveregning (forberedelse): 3 timer ⇐ **mest vigtig**
- ▶ Opgaveregning (klasse): 2 timer

Forventet arbejdsindsats

- ▶ Skim stof før forelæsning: 0,5 timer \Leftarrow mindst vigtig
- ▶ Forelæsning: 2 timer
- ▶ Læs stof efter forelæsning: 1,5 timer
- ▶ Opgaveregning (forberedelse): 3 timer \Leftarrow mest vigtig
- ▶ Opgaveregning (klasse): 2 timer

Ovenstående i gennemsnit 1.5 gang per uge over 14 uger. Dertil følgende én gang:

- ▶ Projektet: 15+15+25 timer
- ▶ Eksamenslæsning: 40 timer
- ▶ Spørgetime og eksamen: 6 timer

Forventet arbejdsindsats

- ▶ Skim stof før forelæsning: 0,5 timer \Leftarrow mindst vigtig
- ▶ Forelæsning: 2 timer
- ▶ Læs stof efter forelæsning: 1,5 timer
- ▶ Opgaveregning (forberedelse): 3 timer \Leftarrow mest vigtig
- ▶ Opgaveregning (klasse): 2 timer

Ovenstående i gennemsnit 1.5 gang per uge over 14 uger. Dertil følgende én gang:

- ▶ Projektet: 15+15+25 timer
- ▶ Eksamenslæsning: 40 timer
- ▶ Spørgetime og eksamen: 6 timer

I alt: $14 \cdot 1.5 \cdot 9 + 55 + 40 + 6 = 290$ timer

Forventet arbejdsindsats

- ▶ Skim stof før forelæsning: 0,5 timer \Leftarrow mindst vigtig
- ▶ Forelæsning: 2 timer
- ▶ Læs stof efter forelæsning: 1,5 timer
- ▶ Opgaveregning (forberedelse): 3 timer \Leftarrow mest vigtig
- ▶ Opgaveregning (klasse): 2 timer

Ovenstående i gennemsnit 1.5 gang per uge over 14 uger. Dertil følgende én gang:

- ▶ Projektet: 15+15+25 timer
- ▶ Eksamenslæsning: 40 timer
- ▶ Spørgetime og eksamen: 6 timer

I alt: $14 \cdot 1.5 \cdot 9 + 55 + 40 + 6 = 290$ timer

10 ECTS = $1/6$ årsværk = $1650/6$ timer = 275 timer

Kursets formål og plads i det store billede

Kursets formål og plads i det store billede

Generelt mål i IT: Få computer til at udføre en opgave.

Kursets formål og plads i det store billede

Generelt mål i IT: Få computer til at udføre en opgave.

Relaterede spørgsmål:

Kursets formål og plads i det store billede

Generelt mål i IT: Få computer til at udføre en opgave.

Relaterede spørgsmål:

- ▶ **Hvordan skrives programmer?**

Programmering, programmeringssprog, software engineering.

Kursets formål og plads i det store billede

Generelt mål i IT: Få computer til at udføre en opgave.

Relaterede spørgsmål:

- ▶ **Hvordan skrives programmer?**

Programmering, programmeringssprog, software engineering.

- ▶ **Hvordan skal programmet løse opgaven?**

Algoritmer og datastrukturer, databasesystemer, lineær algebra med anvendelser, data mining og machine learning.

Kursets formål og plads i det store billede

Generelt mål i IT: Få computer til at udføre en opgave.

Relaterede spørgsmål:

- ▶ **Hvordan skrives programmer?**
Programmering, programmeringssprog, software engineering.
- ▶ **Hvordan skal programmet løse opgaven?**
Algoritmer og datastrukturer, databasesystemer, lineær algebra med anvendelser, data mining og machine learning.
- ▶ **(Hvor godt) er det overhovedet muligt at løse opgaven?**
Nedre grænser, kompleksitet, beregnelighed.

Kursets formål og plads i det store billede

Generelt mål i IT: Få computer til at udføre en opgave.

Relaterede spørgsmål:

- ▶ **Hvordan skrives programmer?**
Programmering, programmeringssprog, software engineering.
- ▶ **Hvordan skal programmet løse opgaven?**
Algoritmer og datastrukturer, databasesystemer, lineær algebra med anvendelser, data mining og machine learning.
- ▶ **(Hvor godt) er det overhovedet muligt at løse opgaven?**
Nedre grænser, kompleksitet, beregnelighed.
- ▶ **Hvordan fungerer maskinen der udfører opgaven?**
Baggrundsviden om computerarkitektur og operativsystemer.

Kursets formål og plads i det store billede

Generelt mål i IT: Få computer til at udføre en opgave.

Relaterede spørgsmål:

- ▶ **Hvordan skrives programmer?**
Programmering, programmeringssprog, software engineering.
- ▶ **Hvordan skal programmet løse opgaven?** \Leftarrow DM507
Algoritmer og datastrukturer, databasesystemer, lineær algebra med anvendelser, data mining og machine learning.
- ▶ **(Hvor godt) er det overhovedet muligt at løse opgaven?**
Nedre grænser, kompleksitet, beregnelighed.
- ▶ **Hvordan fungerer maskinen der udfører opgaven?**
Baggrundsviden om computerarkitektur og operativsystemer.

Fokus: *Hvordan* skal programmet løse opgaven?

Fokus: *Hvordan* skal programmet løse opgaven?

Algoritme = løsningsmetode.

Fokus: *Hvordan* skal programmet løse opgaven?

Algoritme = løsningsmetode.

Tilpas præcist skrevet ned: præcis tekst, pseudo-kode, flow-diagrammer, formler, ...

Fokus: *Hvordan* skal programmet løse opgaven?

Algoritme = løsningsmetode.

Tilpas præcist skrevet ned: præcis tekst, pseudo-kode, flow-diagrammer, formler, ...

Datastruktur = data + effektive operationer herpå.

Fokus: *Hvordan* skal programmet løse opgaven?

Algoritme = løsningsmetode.

Tilpas præcist skrevet ned: præcis tekst, pseudo-kode, flow-diagrammer, formler,...

Datastruktur = data + effektive operationer herpå.

Forskellige datastrukturer gemmer forskellige typer data og/eller tilbyder forskellige operationer. Har stor anvendelse som delement i algoritmer.

Fokus: *Hvordan* skal programmet løse opgaven?

Algoritme = løsningsmetode.

Tilpas præcist skrevet ned: præcis tekst, pseudo-kode, flow-diagrammer, formler,...

Datastruktur = data + effektive operationer herpå.

Forskellige datastrukturer gemmer forskellige typer data og/eller tilbyder forskellige operationer. Har stor anvendelse som delement i algoritmer.

Relevante opgaver for enhver beregningsopgave:

1. **Find** (mindst) én algoritme der løser opgaven.

Fokus: *Hvordan* skal programmet løse opgaven?

Algoritme = løsningsmetode.

Tilpas præcist skrevet ned: præcis tekst, pseudo-kode, flow-diagrammer, formler, ...

Datastruktur = data + effektive operationer herpå.

Forskellige datastrukturer gemmer forskellige typer data og/eller tilbyder forskellige operationer. Har stor anvendelse som delement i algoritmer.

Relevante opgaver for enhver beregningsopgave:

1. **Find** (mindst) én algoritme der løser opgaven.
2. **Sammenlign** flere algoritmer der løser opgaven.

Fokus: *Hvordan* skal programmet løse opgaven?

Algoritme = løsningsmetode.

Tilpas præcist skrevet ned: præcis tekst, pseudo-kode, flow-diagrammer, formler, ...

Datastruktur = data + effektive operationer herpå.

Forskellige datastrukturer gemmer forskellige typer data og/eller tilbyder forskellige operationer. Har stor anvendelse som delement i algoritmer.

Relevante opgaver for enhver beregningsopgave:

1. **Find** (mindst) én algoritme der løser opgaven.
2. **Sammenlign** flere algoritmer der løser opgaven.
3. Hvad er **den bedste** algoritme der kan findes?

Udvikling og vurdering af algoritmer

1. **Find** (mindst) én algoritme der løser opgaven.
2. **Sammenlign** flere algoritmer der løser opgaven.
3. Hvad er **den bedste** algoritme der kan findes?

Udvikling og vurdering af algoritmer

1. **Find** (mindst) én algoritme der løser opgaven.
2. **Sammenlign** flere algoritmer der løser opgaven.
3. Hvad er **den bedste** algoritme der kan findes?

Punkt 1: Kræver ideer, tænkearbejde, erfaring, og en værktøjskasse af kendte algoritmer. Korrekthed: ved analyse eller implementation/afprøvning?

Udvikling og vurdering af algoritmer

1. **Find** (mindst) én algoritme der løser opgaven.
2. **Sammenlign** flere algoritmer der løser opgaven.
3. Hvad er **den bedste** algoritme der kan findes?

Punkt 1: Kræver ideer, tænkearbejde, erfaring, og en værktøjskasse af kendte algoritmer. Korrekthed: ved analyse eller implementation/afprøvning?

Punkt 2: Kræver definition af hvad er kvalitet. Sammenligning: ved analyse eller implementation/afprøvning?

Udvikling og vurdering af algoritmer

1. **Find** (mindst) én algoritme der løser opgaven.
2. **Sammenlign** flere algoritmer der løser opgaven.
3. Hvad er **den bedste** algoritme der kan findes?

Punkt 1: Kræver ideer, tænkearbejde, erfaring, og en værktøjskasse af kendte algoritmer. Korrekthed: ved analyse eller implementation/afprøvning?

Punkt 2: Kræver definition af hvad er kvalitet. Sammenligning: ved analyse eller implementation/afprøvning?

Analyse: Giver høj sikkerhed for korrekthed. Sparer implementationsarbejde. Sammenligning upåvirket af: maskine, sprog, programmør, konkrete input. **Afprøvning:** giver konkret feedback, fanger andre ting end analyse.

Udvikling og vurdering af algoritmer

1. **Find** (mindst) én algoritme der løser opgaven.
2. **Sammenlign** flere algoritmer der løser opgaven.
3. Hvad er **den bedste** algoritme der kan findes?

Punkt 1: Kræver ideer, tænkearbejde, erfaring, og en værktøjskasse af kendte algoritmer. Korrekthed: ved analyse eller implementation/afprøvning?

Punkt 2: Kræver definition af hvad er kvalitet. Sammenligning: ved analyse eller implementation/afprøvning?

Analyse: Giver høj sikkerhed for korrekthed. Sparer implementationsarbejde. Sammenligning upåvirket af: maskine, sprog, programmør, konkrete input. **Afprøvning:** giver konkret feedback, fanger andre ting end analyse.

I alle byggefag analyserer og planlægger man før man bygger (tænk storebæltsbro). Din fremtidige chef vil forlange det!

Udvikling og vurdering af algoritmer

1. **Find** (mindst) én algoritme der løser opgaven.
2. **Sammenlign** flere algoritmer der løser opgaven.
3. Hvad er **den bedste** algoritme der kan findes?

Punkt 1: Kræver ideer, tænkearbejde, erfaring, og en værktøjskasse af kendte algoritmer. Korrekthed: ved analyse eller implementation/afprøvning?

Punkt 2: Kræver definition af hvad er kvalitet. Sammenligning: ved analyse eller implementation/afprøvning?

Analyse: Giver høj sikkerhed for korrekthed. Sparer implementationsarbejde. Sammenligning upåvirket af: maskine, sprog, programmør, konkrete input. **Afprøvning:** giver konkret feedback, fanger andre ting end analyse.

I alle byggefag analyserer og planlægger man før man bygger (tænk storebæltsbro). Din fremtidige chef vil forlange det!

(Bemærk: **Punkt 3** kan *kun* afklares med analyse.)

Udvikling og vurdering af algoritmer

1. **Find** (mindst) én algoritme der løser opgaven.
2. **Sammenlign** flere algoritmer der løser opgaven.
3. Hvad er **den bedste** algoritme der kan findes?

Punkt 1: Kræver ideer, tænkearbejde, erfaring, og en værktøjskasse af kendte algoritmer. Korrekthed: ved analyse eller implementation/afprøvning?

Punkt 2: Kræver definition af hvad er kvalitet. Sammenligning: ved analyse eller implementation/afprøvning?

Analyse: Giver høj sikkerhed for korrekthed. Sparer implementationsarbejde. Sammenligning upåvirket af: maskine, sprog, programmør, konkrete input. **Afprøvning:** giver konkret feedback, fanger andre ting end analyse.

I alle byggefag analyserer og planlægger man før man bygger (tænk storebæltsbro). Din fremtidige chef vil forlange det!

(Bemærk: **Punkt 3** kan *kun* afklares med analyse.)

DM507 vil have mest fokus på analyse, lidt mindre på implementation.

Målsætning for kurset

Målsætning for kurset

DM507 giver dig en værktøjskasse af algoritmer for fundamentale opgaver, samt metoder til at udvikle og analysere nye algoritmer og varianter af eksisterende.



Målsætning for kurset

Regneøvelser og programmeringsprojekter øger din forståelse for værktøjerne og træner dig i brug af værktøjskassen.



Målsætning for kurset

Regneøvelser og programmeringsprojekter øger din forståelse for værktøjerne og træner dig i brug af værktøjskassen.



Undervejs begejstres du måske også over smarte og elegante ideer i algoritmer og analyser.



Konkret indhold af kurset

Konkret indhold af kurset

Algoritmer:

- ▶ Analyse af algoritmer: korrekthed og køretid
- ▶ Del og hersk algoritmer
- ▶ Grådige algoritmer
- ▶ Dynamisk programmering
- ▶ Sortering
- ▶ Graf-algoritmer
- ▶ Huffman-kodning

Datastrukturer:

- ▶ Ordbøger (søgetræer og hashing)
- ▶ Prioritetskøer (heaps)
- ▶ Disjunkte mængder

Algorithmanalyse

Algoritmeanalyse

Mindstekrav til algoritmer for at løse et problem:

- ▶ Stopper for alle input.
- ▶ Korrekt output når stopper.

Algoritmeanalyse

Mindstekrav til algoritmer for at løse et problem:

- ▶ Stopper for alle input.
- ▶ Korrekt output når stopper.

Kvalitet af algoritmer som opfylder mindstekrav:

- ▶ Hastighed
- ▶ Pladsforbrug
- ▶ Komplexitet af implementation
- ▶ Ekstra egenskaber (problemspecifikke), f.eks. stabilitet af sortering.

Algoritmeanalyse

Mindstekrav til algoritmer for at løse et problem:

- ▶ Stopper for alle input.
- ▶ Korrekt output når stopper.

Kvalitet af algoritmer som opfylder mindstekrav:

- ▶ Hastighed
- ▶ Pladsforbrug
- ▶ Komplexitet af implementation
- ▶ Ekstra egenskaber (problemspecifikke), f.eks. stabilitet af sortering.

For dette kræves følgende ingredienser: klar beskrivelse af problem og maskine (modeller), en definition af kvalitet, samt en værktøjskasse af analyseredskaber.

Ingredienser i algoritmeanalyse

Ingredienser i algoritmeanalyse

- ▶ Model af problem. Individuelt for hvert problem.

Ingredienser i algoritmeanalyse

- ▶ Model af problem. Individuelt for hvert problem.
- ▶ Model af maskine. Ofte RAM-modellen (alias von Neumann modellen).

Ingredienser i algoritmeanalyse

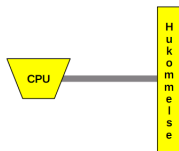
- ▶ Model af problem. Individuelt for hvert problem.
- ▶ Model af maskine. Ofte RAM-modellen (alias von Neumann modellen).
- ▶ Mål for ressourceforbrug (tid og plads).

Ingredienser i algoritmeanalyse

- ▶ Model af problem. Individuelt for hvert problem.
- ▶ Model af maskine. Ofte RAM-modellen (alias von Neumann modellen).
- ▶ Mål for ressourceforbrug (tid og plads).
- ▶ Matematiske analyseværktøjer: Løkkeinvarianter, induktion, rekursionsligninger.

RAM-modellen

RAM-modellen



- ▶ En CPU
- ▶ En hukommelse (\sim uendeligt array af celler).
- ▶ Et antal basale operationer: *add*, *sub*, *mult*, *shift*, *compare*, *flyt dataelement*, *jump i program* (løkke, forgrening, metodekald). Disse antages alle at tage samme tid.
- ▶ **Tid** for en algoritme: antal basale operationer udført.
- ▶ **Plads** for en algoritme: maks antal optagne hukommelsesceller.

Måle ressourceforbrug

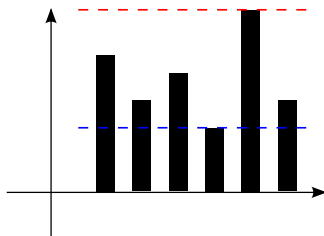
Måle ressourceforbrug

For en givet størrelse n af input er der ofte mange forskellige input instanser. Algoritmen har som regel forskelligt ressourceforbrug på hver af disse. Hvilket skal vi bruge til at vurdere ressourceforbruget?

Måle ressourceforbrug

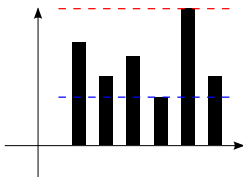
For en givet størrelse n af input er der ofte mange forskellige input instanser. Algoritmen har som regel forskelligt ressourceforbrug på hver af disse. Hvilket skal vi bruge til at vurdere ressourceforbruget?

- ▶ **Worst case** (max over alle input af størrelse n)
- ▶ Average case (gennemsnit over en fordeling af input af størrelse n)
- ▶ **Best case** (min over alle input af størrelse n)



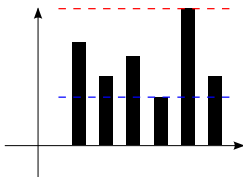
Køretid for de forskellige input af størrelse n

Worst case ressourceforbrug



Worst case giver garanti. Ofte repræsentativ for average case (men nogen gange betydeligt mere pessimistisk).

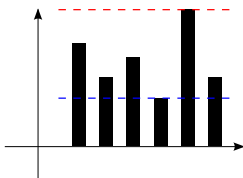
Worst case ressourceforbrug



Worst case giver garanti. Ofte repræsentativ for average case (men nogen gange betydeligt mere pessimistisk).

Average case: Hvilken fordeling? Er den realistisk? Ofte svær analyse at gennemføre (matematisk svær).

Worst case ressourceforbrug

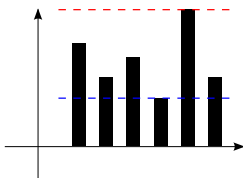


Worst case giver garanti. Ofte repræsentativ for average case (men nogen gange betydeligt mere pessimistisk).

Average case: Hvilken fordeling? Er den realistisk? Ofte svær analyse at gennemføre (matematisk svær).

Best case: Giver ofte ikke så megen relevant information.

Worst case ressourceforbrug



Worst case giver garanti. Ofte repræsentativ for average case (men nogen gange betydeligt mere pessimistisk).

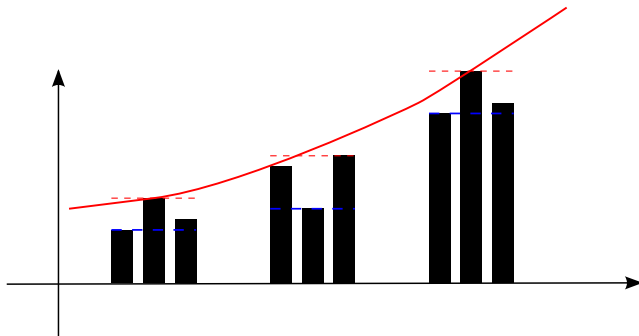
Average case: Hvilken fordeling? Er den realistisk? Ofte svær analyse at gennemføre (matematisk svær).

Best case: Giver ofte ikke så megen relevant information.

Næsten alle analyser i dette kursus er worst case.

Forskellige inputstørrelser

Worstcase køretid er normalt en voksende funktion af inputstørrelsen n :



Køretid for de forskellige input af stigende størrelse n

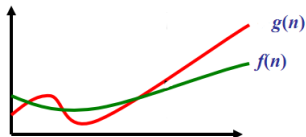
Voksehastighed

Forbruget skal derfor ses som en funktion $f(n)$ af inputstørrelsen n .

Voksehastighed

Forbruget skal derfor ses som en **funktion** $f(n)$ af inputstørrelsen n .

Vi har derfor brug for at **sammenligne funktioner**. Det relevante mål er **voksehastighed** - en hurtigere voksende funktion vil altid overhale en langsomt voksende funktion når n bliver stor nok. Og for små n er (næsten) alle algoritmer hurtige.



Voksehastighed

Eksempler (stigende voksehastighed):

$$1, \log n, \sqrt{n}, n, n \log n,$$
$$n\sqrt{n}, n^2, n^3, n^{10}, 2^n$$

Voksehastighed

Eksempler (stigende voksehastighed):

$$1, \log n, \sqrt{n}, n, n \log n, \\ n\sqrt{n}, n^2, n^3, n^{10}, 2^n$$

Næste gang: mere præcis definition af [asymptotisk voksehastighed](#) og sammenligninger heraf.

Konkret eksempel på algoritmeanalyse

Ombytningspuslespil . . .