Computere, Algoritmer og Kompleksitet.

UNF foredrag 27. april, 2023

Rolf Fagerberg

Professor i Datalogi

Institut for Matematik og Datalogi, SDU

Plan for i dag

Tre kapitler:

1. Computeren inderst inde.

Plan for i dag

Tre kapitler:

1. Computeren inderst inde.
2. Algoritmer: Få tingene gjort!

Plan for i dag

Tre kapitler:

1. Computeren inderst inde.
2. Algoritmer: Få tingene gjort!
3. Kompleksitet: Hvor svært kan det være?

Computeren inderst inde

Bitmønstre

Information = valg mellem forskellig muligheder.

Simpleste situation: valg mellem to muligheder. Kald dem 0 og 1. Denne valgmulighed kaldes en bit.

Bitmønstre

Information = valg mellem forskellig muligheder.

Simpleste situation: valg mellem to muligheder. Kald dem 0 og 1. Denne valgmulighed kaldes en bit.

> Relevant for computere fordi to-delte valg er nemmest at repræsentere rent fysisk: 1 = strøm, 0 = ikke strøm.

Bitmønstre

Information = valg mellem forskellig muligheder.

Simpleste situation: valg mellem to muligheder. Kald dem 0 og 1. Denne valgmulighed kaldes en bit.

> Relevant for computere fordi to-delte valg er nemmest at repræsentere rent fysisk: 1 = strøm, 0 = ikke strøm.

Større samling information: brug flere bits.

F.eks. giver 8 bits et valg mellem

$$2 \cdot 2 \cdot \cdots \cdot 2 = 2^8 = 256 \text{ muligheder:}$$

00000000
10000000
01000000
11000000
⋮
11111111

Bitmønstre

Bitmønstre skal *fortolkes* for at have en betydning.

$$01101011 = ?$$

Der er brug for et system, som angiver, hvilke betydninger der vælges mellem af de forskellige bitmønstre.

Bitmønstre

Bitmønstre skal *fortolkes* for at have en betydning.

$$01101011 = ?$$

Der er brug for et system, som angiver, hvilke betydninger der vælges mellem af de forskellige bitmønstre.

Der er lavet sådanne systemer for f.eks.:

- Tal (heltal, kommatal)
- Bogstaver
- Computerinstruktion (program)
- Pixels (billedfil)
- Amplitude (lydfil)
- $\vdots$

Bitmønstre

Bitmønstre skal *fortolkes* for at have en betydning.

$$01101011 = ?$$

Der er brug for et system, som angiver, hvilke betydninger der vælges mellem af de forskellige bitmønstre.

Der er lavet sådanne systemer for f.eks.:

- Tal (heltal, kommatal)
- Bogstaver
- Computerinstruktion (program)
- Pixels (billedfil)
- Amplitude (lydfil)
- $\vdots$

Eksempel på de næste sider: heltal.

Talsystemer

Ti-tal-systemet:

4532

Talsystemer

Ti-tal-systemet:

$$4532 = 4 \cdot 1000 + 5 \cdot 100 + 3 \cdot 10 + 2 \cdot 1$$

Talsystemer

Ti-tal-systemet:

$$\begin{aligned} 4532 &= 4 \cdot 1000 + 5 \cdot 100 + 3 \cdot 10 + 2 \cdot 1 \\ &= 4 \cdot 10^3 + 5 \cdot 10^2 + 3 \cdot 10^1 + 2 \cdot 10^0 \end{aligned}$$

Talsystemer

Ti-tal-systemet:

$$\begin{aligned} 4532 &= 4 \cdot 1000 + 5 \cdot 100 + 3 \cdot 10 + 2 \cdot 1 \\ &= 4 \cdot 10^3 + 5 \cdot 10^2 + 3 \cdot 10^1 + 2 \cdot 10^0 \end{aligned}$$

Grundtal: 10
Cifre: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (fordi $10 \cdot 10^i = 10^{i+1}$)

Talsystemer

Ti-tal-systemet:

$$\begin{aligned} 4532 &= 4 \cdot 1000 + 5 \cdot 100 + 3 \cdot 10 + 2 \cdot 1 \\ &= 4 \cdot 10^3 + 5 \cdot 10^2 + 3 \cdot 10^1 + 2 \cdot 10^0 \end{aligned}$$

Grundtal: 10
Cifre: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (fordi $10 \cdot 10^i = 10^{i+1}$)

To-tal-systemet:

$$\begin{aligned} 1011_2 &= 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 \\ &= 11 \end{aligned}$$

Grundtal: 2
Cifre: 0, 1 (fordi $2 \cdot 2^i = 2^{i+1}$)

To-tal-systemet

To-tal-systemet er relevant for computere fordi to-delte valg er nemmest at repræsentere rent fysisk (1 = strøm, 0 = ikke strøm).

Total-systemet kaldes også det *binære talsystem*.

Det giver en naturlig fortolkning af bitmønstre som ikke-negative hele tal.

Addition

Addition fungerer ens i alle talsystemer, blot med grundtal udskiftet.

Tital-systemet:

$$\begin{array}{r} {\scriptstyle 1\,1\,1\,1} \\ 5432 \\ +96781 \\ \hline = 102213 \end{array}$$

Addition

Addition fungerer ens i alle talsystemer, blot med grundtal udskiftet.

Tital-systemet:

$$\begin{array}{r} {\scriptstyle 1\,1\,1\,1} \\ 5432 \\ +96781 \\ \hline = 102213 \end{array}$$

Total-systemet:

$$\begin{array}{r} {\scriptstyle 1\,1\,1} \\ 1110_2 \\ +11100_2 \\ \hline = 101010_2 \end{array}$$

Addition

Addition fungerer ens i alle talsystemer, blot med grundtal udskiftet.

Tital-systemet:

$$\begin{array}{r} {\scriptstyle 1\,1\,1\,1} \\ 5432 \\ +96781 \\ \hline = 102213 \end{array}$$

Total-systemet:

$$\begin{array}{r} {\scriptstyle 1\,1\,1} \\ 1110_2 \\ +11100_2 \\ \hline = 101010_2 \end{array}$$

(Subtraktion, multiplikation, division fungerer også ens.)

Beregning

Information = valg mellem forskellig muligheder.

Beregning

Information = valg mellem forskellig muligheder.

Beregning: Ny information fra gammel.

Dvs. nye bits fra gamle bits.

Dvs. funktioner fra bits til bits.

Funktioner

Funktioner kan gives via regneforskrifter:

$$y = f(x_1, x_2) = 3x_1 + 5x_2 + 7$$

Funktioner

Funktioner kan gives via regneforskrifter:

$$y = f(x_1, x_2) = 3x_1 + 5x_2 + 7$$

Funktioner kan også beskrives ved en tabel:

x_1	x_2	y
0	1	12
1	1	15
2	1	18
0	2	17
⋮	⋮	⋮

Funktioner

Funktioner kan gives via regneforskrifter:

$$y = f(x_1, x_2) = 3x_1 + 5x_2 + 7$$

Funktioner kan også beskrives ved en tabel:

x_1	x_2	y
0	1	12
1	1	15
2	1	18
0	2	17
⋮	⋮	⋮

For almindelige tal er der uendeligt mange input, så en tabel vil altid være en ufuldstændig beskrivelse.

Funktioner

Funktioner kan gives via regneforskrifter:

$$y = f(x_1, x_2) = 3x_1 + 5x_2 + 7$$

Funktioner kan også beskrives ved en tabel:

x_1	x_2	y
0	1	12
1	1	15
2	1	18
0	2	17
⋮	⋮	⋮

For almindelige tal er der uendeligt mange input, så en tabel vil altid være en ufuldstændig beskrivelse.

Men for funktioner af bits har hver input-variabel kun to mulige værdier. Derfor er der et endeligt antal forskellige input, så tabeller kan beskrive funktioner fuldstændigt.

Beregning

Beregning = en funktion fra bits til bits.

Beregning

Beregning = en funktion fra bits til bits.

For to bits ind (x_1 og x_2) og én bit ud (y) er mulighederne:

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	0

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	0

x_1	x_2	y
0	0	0
0	1	1
1	0	0
1	1	0

16 stk

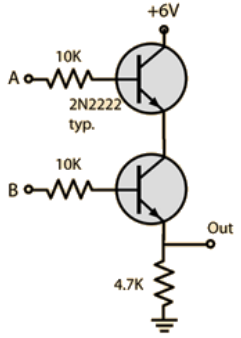
.....

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	1

[Der er $2^2 = 4$ forskellige input (rækker), og derfor $2^4 = 16$ forskellige muligheder for valg af output (sidste søjle)]

Disse funktioner har navne (AND, OR, NAND, ...).

Fact: De kan laves med transistorer og andre simple elektroniske komponente. Her er f.eks. en AND:



Beregning

F.eks. kan addition

$$\begin{array}{r} {\scriptstyle 1\,1\,1} \\ 101010_2 \\ +001111_2 \\ \hline = 111001_2 \end{array}$$

på hver ciffer-position beskrives ved to funktioner

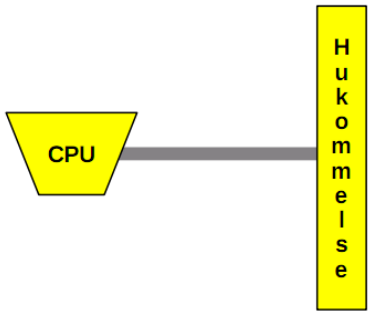
- $\textsc{Resultat}(x_1,x_2,x_3)$, som ud fra de to input-cifre x_1 og x_2 samt menten x_3 på denne plads giver output-cifferet på denne plads.
- $\textsc{Mente}(x_1,x_2,x_3)$, som ud fra de to input-cifre x_1 og x_2 samt menten x_3 på denne plads giver menten på næste plads.

Computere

De basale operationer (plus, gange, division, invertér bits, . . .) i en computer kan beskrives via bit-funktioner og kan derfor implementeres via elektroniske kredsløb.

Computere er essentielt et (stort) system af ledninger og simpel elektronik, som kan repræsentere bits og som kan lave nye bits fra gamle.

Computere



- CPU bygget til at udføre forskellige typer kommandoer:
  1. *Plus*, *minus*, *gange*, *bit-shift*, *sammenlign*,...
  2. *Flyt data rundt i hukommelsen*
  3. *Hop i program* (*løkke*, *forgrening*, *metodekald*).
- Hukommelse (række af celler, hver f.eks. 64 bits).

Program = række af kommandoer til CPU'en.

Både data og program ligger i hukommelsen. Kommando angives også med bitmønstre.

Programmering

Programmeringssprog	**Maskinkode**
`if...then...else...`	`01110111`
Tæt på engelsk(?)	Binære koder
Bedre overblik	Dårligt overblik
Sværere at lave fejl	Nemt at lave fejl
Hurtig programmering	Langsom programmering

Oversætter-program: Højniveau sprog $\rightarrow$ maskinkode.

Programmeringssprog

Eksempler på programmeringssprog:

> Python, Java, C, C++, C#, Javascript, Perl, Ruby, Pascal, Basic, Fortran, Cobol, Ada, ...

De grundlæggende elemener i disse sprog er de samme. Vi viser nogle konkrete eksempler i Python.

Algoritmer: Få tingene gjort!

Det store billede

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

Relaterede spørgsmål i Datalogi:

- Hvordan skrives programmer?
  Programmering, programmeringssprog, software engineering.
- Hvordan skal programmet løse opgaven?
  Algoritmer, 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.

Det store billede

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

Relaterede spørgsmål i Datalogi:

- Hvordan skrives programmer?
  Programmering, programmeringssprog, software engineering.
- Hvordan skal programmet løse opgaven? $\Leftarrow$
  Algoritmer, 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.

Hvordan skal programmet løse opgaven?

Algoritme = løsningsmetode.

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

Hvordan skal programmet løse opgaven?

Algoritme = løsningsmetode.

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

Eksempler fra hverdagen:

Slå 121 (127) 133 (139) m op på p nr. 2 med grå, og strik 5 p ret, frem og tilbage på p. Første p er vr-siden af arb.

Skift til p nr. 2,5, og fortsæt i glat med striber (se ovenfor), idet der samtidig tages ud og ind på hver p fra r-siden således: 1 r, udt (= strik lænken mellem m op, og strik den drejet ret), 19 (20) 21 (22) r, strik 3 m sm (= tag 2 m løst af på én gang, 1 r, træk de løse m over på én gang), 20 (21) 22 (23) r, udt, 1 r, udt, 15 (16) 17 (18) r, 3 m sm, 15 (16) 17 (18) r, udt, 1 r, udt, 20 (21) 22 (23) r, 3 m sm, 19 (20) 21 (22) r, udt, 1 r.

Tag ud og ind på denne måde på hver p fra r-siden i alt 14 (15) 16 (17) gange.

Strik 1 p vrang. * Næste p: 20 (21) 22 (23) ret, strik 3 m sm, 36 (38) 40 (42) ret, 3 m sm, 36 (38) 40 (42) ret, 3 m sm, 20 (21) 22 (23) ret. Strik 1 p vrang. Strik de to p endnu 1 gang, idet der for hver p med indt bliver 2 m mindre mellem indt. Strik 2 p glat *.

Gentag disse 6 p fra * til * i alt 3 gange. Bemærk, at indt fortsat skal ligge lige over hinanden. Fortsæt derefter med indt på hver p fra r-siden, til alle m mellem indt r taget ind. Strik 3 cm lige op med de resterende 7 m. Bryd garnet, træk tråden gennem m, stram til, og hæft ende.

Strikkeopskrift

Citrongræsrejer i wok med grønt

Tid i alt **20 min.** Arbejdstid **20 min.** Antal **4 pers.**

Ingredienser

- 1 stængel citrongræs
- 3 fed hvidløg, pressede
- 1 tsk ingefær, fintrevet
- 1 tsk sriracha chilisauce
- 2 spsk honning
- 3 spsk sesamolie
- 300 g vannamei tigerrejer
- 30 g smør
- 1 broccoli, i små buketter
- 250 g spinat
- 3 gulerod, strimlet
- 30 g cashewnødder
- 3 spsk olivenolie
- salt
- sort peber, friskkværnet

Fremgangsmåde

Pil det yderste lag af citrongræsset og skær enderne af. Snit resten af citrongræsstilken helt fint. Kom citrongræs, hvidløg, ingefær, sriracha, honning og sesamolie i en skål og rør det godt sammen. Vend rejerne godt i blandingen.

Varm en wok eller pande op og og rist rejerne under omrøring i et minuts tid. Kom smør, salt og peber på panden og steg videre til rejerne er lyserøde og har en let og smuk stegeskorpe. Sæt rejerne til side på en tallerken til senere.

Kom broccoli, spinat, gulerødder og cashewnødder på wok/panden med lidt ekstra olie og steg det et par minutter.

Server i dybe skåle med ris toppet med wok med både rejer og grønt. Drys basilikum, forårsløg og chiliflager over og server med limebåde til at dryppe over.

Madopskrift

Eksempler på algoritmer

Problem: Find største tal i en liste L = [8,2,11,5,42,1]

Eksempler på algoritmer

Problem: Find største tal i en liste L = [8,2,11,5,42,1]

Algoritme:

```
sæt max-kandidat = 0
for alle tal x i L:
  hvis x > max-kandidat:
    sæt max-kandidat = x
returner max-kandidat
```

Eksempler på algoritmer

Problem: Find største tal i en liste L = [8,2,11,5,42,1]

Algoritme:

```
sæt max-kandidat = 0
for alle tal x i L:
  hvis x > max-kandidat:
    sæt max-kandidat = x
returner max-kandidat
```

Problem: Sorter en liste L = [8,2,11,5,42,1]

Eksempler på algoritmer

Problem: Find største tal i en liste L = [8,2,11,5,42,1]

Algoritme:

```
sæt max-kandidat = 0
for alle tal x i L:
  hvis x > max-kandidat:
    sæt max-kandidat = x
returner max-kandidat
```

Problem: Sorter en liste L = [8,2,11,5,42,1]

Algoritme:

```
gentag n gange:
  find og fjern det største tal i L
```

Eksempler på algoritmer

Problem: Find største tal i en liste L = [8,2,11,5,42,1]

Algoritme:

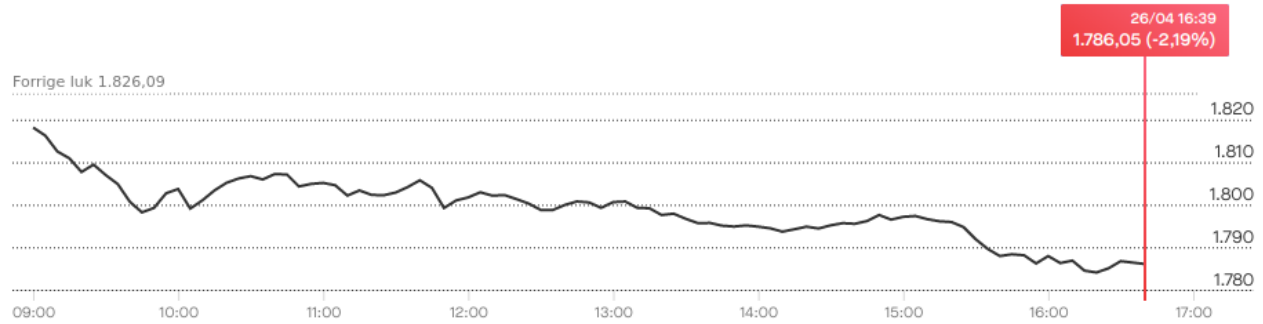
```
sæt max-kandidat = 0
for alle tal x i L:
  hvis x > max-kandidat:
    sæt max-kandidat = x
returner max-kandidat
```

Problem: Sorter en liste L = [8,2,11,5,42,1]

Algoritme:

```
for position i = 1,2,...,n i L:
  find største tal efter i
  byt dette med plads i
```

Eksempler på algoritmer: aktieanalyse



TOP		BUND	
Netcompany Group	2,24% ▲ 251,00 DKK	Novo Nordisk B	-4,14% ▼ 1.106,80 DKK
Demant	1,26% ▲ 288,80 DKK	Bavarian Nordic	-4,14% ▼ 180,75 DKK
Tryg	0,16% ▲ 159,05 DKK	Novozymes B	-3,61% ▼ 354,90 DKK
Ørsted	-0,20% ▼ 597,00 DKK	Chr. Hansen Holding	-3,14% ▼ 536,00 DKK
Jyske Bank	-0,50% ▼ 495,50 DKK	Vestas Wind Systems	-3,07% ▼ 179,28 DKK

Kompleksitet: Hvor svært kan det være?

Måle ressourceforbrug

For en givet størrelse n af input er der ofte mange forskellige konkrete input.

Eksempel: for sortering af $n = 8$ tal er følgende nogle af de mulige input:

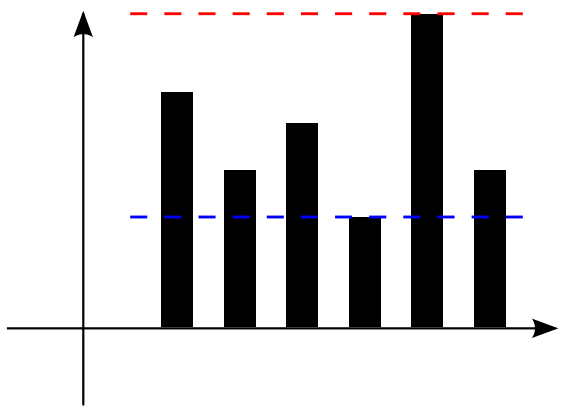
7,2,3,1,8,5,4,6

1,8,2,7,3,6,4,5

1,2,3,4,5,6,7,8

8,7,6,5,4,3,2,1

$\vdots$



Køretid for de forskellige input af størrelse n

Måle ressourceforbrug

For en givet størrelse n af input er der ofte mange forskellige konkrete input.

Eksempel: for sortering af $n = 8$ tal er følgende nogle af de mulige input:

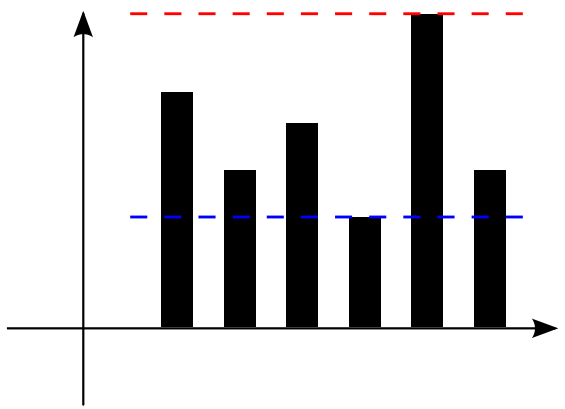
7,2,3,1,8,5,4,6

1,8,2,7,3,6,4,5

1,2,3,4,5,6,7,8

8,7,6,5,4,3,2,1

$\vdots$

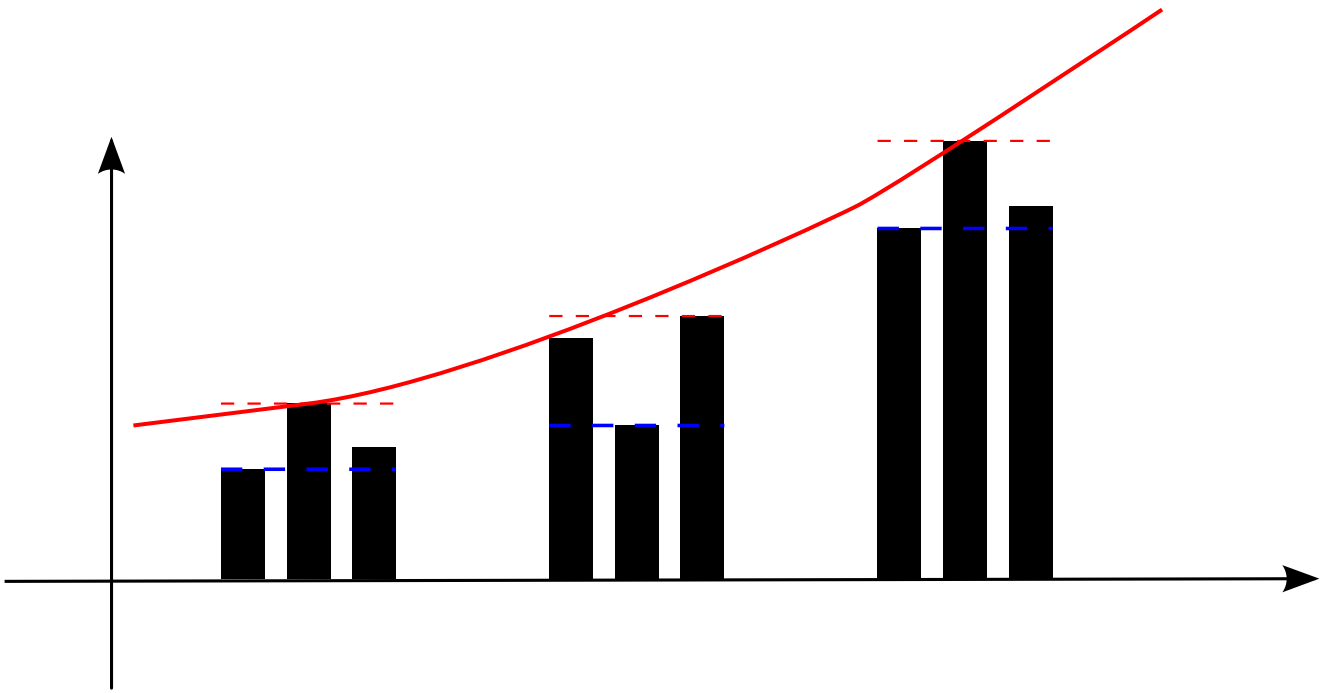


Køretid for de forskellige input af størrelse n

En algoritme har som regel forskelligt ressourceforbrug på hver af disse.

Forskellige inputstørrelser

Worstcase køretid er normalt en voksende funktion af inputstørrelsen n:



Køretid for de forskellige input af stigende størrelse n

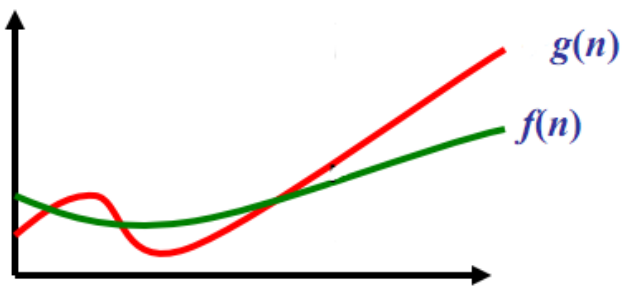
Voksehastighed

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):

$$1, \quad \log n, \quad \sqrt{n}, \quad n, \quad n \log n,$$

$$n\sqrt{n}, \quad n^2, \quad n^3, \quad n^{10}, \quad 2^n$$

Barregninger

Iron Maiden (NOK)

LeBron James (USD)

Dele en barregning i to

Problem: givet en række tal $\{34, 20, 456, 932, 110, \ldots, 57\}$, findes der en delmængde af disse tal, hvis sum er præcis halvdelen af den samlede sum?

Dele en barregning i to

Problem: givet en række tal $\{34, 20, 456, 932, 110, \ldots, 57\}$, findes der en delmængde af disse tal, hvis sum er præcis halvdelen af den samlede sum?

Eksempler:

1) $\{1, 2, 3, 4\}$?

Dele en barregning i to

Problem: givet en række tal $\{34, 20, 456, 932, 110, \ldots, 57\}$, findes der en delmængde af disse tal, hvis sum er præcis halvdelen af den samlede sum?

Eksempler:

1) $\{1, 2, 3, 4\}$? Ja: $\{2, 3\}$.

Dele en barregning i to

Problem: givet en række tal $\{34, 20, 456, 932, 110, \ldots, 57\}$, findes der en delmængde af disse tal, hvis sum er præcis halvdelen af den samlede sum?

Eksempler:

1) $\{1, 2, 3, 4\}$? Ja: $\{2, 3\}$.

2) $\{1, 3, 8\}$?

Dele en barregning i to

Problem: givet en række tal $\{34, 20, 456, 932, 110, \ldots, 57\}$, findes der en delmængde af disse tal, hvis sum er præcis halvdelen af den samlede sum?

Eksempler:

1) $\{1, 2, 3, 4\}$? Ja: $\{2, 3\}$.

2) $\{1, 3, 8\}$? Nej (prøv alle muligheder).

Dele en barregning i to

Algoritme?

Dele en barregning i to

Algoritme?

Idé: Prøv alle muligheder (alle delmængder af tal).

Dele en barregning i to

Algoritme?

Idé: Prøv alle muligheder (alle delmængder af tal).

Tid: For n tal er der 2^n delmængder at prøve. Meget, meget langsom algoritme.

Dele en barregning i to

Algoritme?

Idé: Prøv alle muligheder (alle delmængder af tal).

Tid: For n tal er der 2^n delmængder at prøve. Meget, meget langsom algoritme.

Findes der en bedre algoritme (f.eks. tid n^2 eller n^3 eller bare n^{100})?

Dele en barregning i to

Algoritme?

Idé: Prøv alle muligheder (alle delmængder af tal).

Tid: For n tal er der 2^n delmængder at prøve. Meget, meget langsom algoritme.

Findes der en bedre algoritme (f.eks. tid n^2 eller n^3 eller bare n^{100})?

Ingen ved det.

Dele en barregning i to

Algoritme?

Idé: Prøv alle muligheder (alle delmængder af tal).

Tid: For n tal er der 2^n delmængder at prøve. Meget, meget langsom algoritme.

Findes der en bedre algoritme (f.eks. tid n^2 eller n^3 eller bare n^{100})?

Ingen ved det.

Millenium prize (Clay Mathematics Institute): 1 million USD prize

NP vs. P

NP: en løsning kan checkes hurtigt (polynomiel tid)

P: en løsning kan findes hurtigt (polynomiel tid)

NP vs. P

NP: en løsning kan checkes hurtigt (polynomiel tid)

P: en løsning kan findes hurtigt (polynomiel tid)

Bemærk: barregningproblemet er i NP.

NP vs. P

NP: en løsning kan checkes hurtigt (polynomiel tid)

P: en løsning kan findes hurtigt (polynomiel tid)

Bemærk: barregningproblemet er i NP.

$P \subseteq NP$ er relativt klart ("check en foreslået løsning mod den vi finder selv").

NP vs. P

NP: en løsning kan checkes hurtigt (polynomiel tid)

P: en løsning kan findes hurtigt (polynomiel tid)

Bemærk: barregningproblemet er i NP.

$P \subseteq NP$ er relativt klart ("check en foreslået løsning mod den vi finder selv").

Gælder $P = NP$ eller $P \subsetneq NP$?

NP-fuldstændighed

Reduktioner: viser at et problem er mindst lige så hårdt som et andet.

NP-fuldstændighed

Reduktioner: viser at et problem er mindst lige så hårdt som et andet.

NP-fuldstændigt problem: er mindst lige så hårdt som *alle* problemer i NP.

NP-fuldstændighed

Reduktioner: viser at et problem er mindst lige så hårdt som et andet.

NP-fuldstændigt problem: er mindst lige så hårdt som *alle* problemer i NP.

Lang liste af praktisk relevante problemer er (desværre) blevet vist NP-fuldstændige.

Herunder barregningeproblemet (med vilkårligt store tal).

NP-fuldstændighed

Reduktioner: viser at et problem er mindst lige så hårdt som et andet.

NP-fuldstændigt problem: er mindst lige så hårdt som *alle* problemer i NP.

Lang liste af praktisk relevante problemer er (desværre) blevet vist NP-fuldstændige.

Herunder barregningeproblemet (med vilkårligt store tal).

> Hvis bare ét NP-fuldstændigt problem vises enten at være i P (dvs. at have en effektiv algoritme) eller vises ikke at have være det, bliver N vs. NP problemet løst.

Længere ud: Uafgørlige problemer

Problemer som INGEN algoritme har (uanset køretid).

Længere ud: Uafgørlige problemer

Problemer som INGEN algoritme har (uanset køretid).

Eksempel: Halting problemet:

Input: Et python-program (tekst) P og en streng I.

Output: True, hvis P stopper på input I, ellers False.

Ingen algoritme kan løse Halting problemet

Antag der findes et program Halting(P, I), som løser halting problemet.
Dvs. Halting(P, I) returnerer True hvis og kun hvis P stopper på input I.

Ingen algoritme kan løse Halting problemet

Antag der findes et program Halting(P, I), som løser halting problemet.
Dvs. Halting(P, I) returnerer True hvis og kun hvis P stopper på input I.

Vi kan så lave følgende program Z:

```
ProgramZ(String x)
  if Halting(x, x) then
       Loop Forever
  else stop.
```

Ingen algoritme kan løse Halting problemet

Antag der findes et program Halting(P, I), som løser halting problemet. Dvs. Halting(P, I) returnerer True hvis og kun hvis P stopper på input I.

Vi kan så lave følgende program Z:

```
ProgramZ(String x)
  if Halting(x, x) then
       Loop Forever
  else stop.
```

Hvad sker der, når programmet Z gives input Z? Der er to muligheder:

Case 1: Program Z stopper på input Z. Så vil Halting returnere Sandt på input (Z, Z). Derfor looper Z evigt (se if-delen). Modstrid.

Case 2: Program Z looper evigt på input Z. Så vil Halting returnere Falsk på input (Z, Z). Derfor stopper program Z på input Z (se else-delen). Modstrid.