Algorithms, Correctness and Efficiency

Algorithms and Datastructures, F25, Lecture 1

Andreas Holck Høeg-Petersen

Department of Computer Science
Aalborg University

February 11, 2025

AALBORG UNIVERSITET

Outline

AALBORG UNIVERSITET

Outline

AALBORG UNIVERSITET

Læringsmål

VIDEN

Den studerende skal opnå viden om følgende teorier og metoder:

- matematiske grundbegreber såsom rekursion, induktion, konkret og abstrakt kompleksitet

FÆRDIGHEDER

- bestemme abstrakt kompleksitet for konkrete funktioner
- gennemføre kompleksitets- og korrekthedsanalyse på simple algoritmer, herunder rekursive algoritmer

KOMPETENCER

Den studerende skal, stillet overfor en ikke-standard programmeringsopgave kunne:

- analysere de udviklede algoritmer

AALBORG UNIVERSITET

Læringsmål

Mere uformelt, mere brugbart

I skal lære:

Læringsmål

Mere uformelt, mere brugbart

I skal lære:

- Hvad algoritmer er, hvilke problemer man kan løse med dem, og hvordan (smarte) datastrukturer spiller en central rolle i den opgave

AALBORG UNIVERSITET

Læringsmål

Mere uformelt, mere brugbart

I skal lære:

- Hvad algoritmer er, hvilke problemer man kan løse med dem, og hvordan (smarte) datastrukturer spiller en central rolle i den opgave
- Hvordan man analyserer algoritmer i forhold til deres køretid, pladsforbrug og korrekthed

AALBORG UNIVERSITET

Læringsmål

Mere uformelt, mere brugbart

I skal lære:

- Hvad algoritmer er, hvilke problemer man kan løse med dem, og hvordan (smarte) datastrukturer spiller en central rolle i den opgave
- Hvordan man analyserer algoritmer i forhold til deres køretid, pladsforbrug og korrekthed
- Teknikker til at designe algoritmer til specifikke problemer

AALBORG UNIVERSITET

Læringsmål

Mere uformelt, mere brugbart

I skal lære:

- Hvad algoritmer er, hvilke problemer man kan løse med dem, og hvordan (smarte) datastrukturer spiller en central rolle i den opgave
- Hvordan man analyserer algoritmer i forhold til deres køretid, pladsforbrug og korrekthed
- Teknikker til at designe algoritmer til specifikke problemer
- Algoritmisk tænkning, så I kan formulere abstrakte problemer som algoritmiske problemer, I kan løse med de værktøjer, I lærer i kurset

AALBORG UNIVERSITET

Kursets struktur

- 10 forelæsninger med tilhørende exercises

Kursets struktur

- 10 forelæsninger med tilhørende exercises
- 1 dejlig TA i form af Jakob Rossander Kristensen

AALBORG UNIVERSITET

Kursets struktur

- 10 forelæsninger med tilhørende exercises
- 1 dejlig TA i form af Jakob Rossander Kristensen
- 2 'self-study' sessioner med tidligere eksamenssæt

AALBORG UNIVERSITET

Kursets struktur

- 10 forelæsninger med tilhørende exercises
- 1 dejlig TA i form af Jakob Rossander Kristensen
- 2 'self-study' sessioner med tidligere eksamenssæt
- 3 programmeringsopgaver

AALBORG UNIVERSITET

Kursets struktur

- 10 forelæsninger med tilhørende exercises
- 1 dejlig TA i form af Jakob Rossander Kristensen
- 2 'self-study' sessioner med tidligere eksamenssæt
- 3 programmeringsopgaver
  - Skal afleveres i grupper
  - Træner kreativ problemløsning og konkretiserer algoritmerne
  - Python-undervisning tilbydes

AALBORG UNIVERSITET

Kursets struktur

Forelæsningerne og exercise-sessionerne ligger fra 12-16 om torsdagen. De vil nogenlunde være arrangeret på følgende måde:

Kursets struktur

Forelæsningerne og exercise-sessionerne ligger fra 12-16 om torsdagen. De vil nogenlunde være arrangeret på følgende måde:

- Forelæsning part 1 fra 12-13
- Exercises med hjælp og vejledning fra 13-14
- Forelæsning part 2 fra 14-15
- Exercises med hjælp og vejledning fra 15-16

AALBORG UNIVERSITET

Kursets struktur

HUSK:

Kurset kræver mere tid end de 4 timer, der er sat af hver torsdag!

Det giver mening (og kan være sjovt) at læse!

En del af det at studere er at terpe!

AALBORG UNIVERSITET

Eksamen

- 4 timers skriftlig eksamen (fedt!)

Eksamen

- 4 timers skriftlig eksamen (fedt!)
- Alle hjælpemidler er tilladte!

AALBORG UNIVERSITET

Eksamen

- 4 timers skriftlig eksamen (fedt!)
- Alle hjælpemidler er tilladte! Pånær...
  - ChatGPT og anden generativ AI...
  - Kommunikation med andre mennesker
  - Værktøjer specifik designet til at løse visse problemer (mere om disse, når vi kommer dertil)

AALBORG UNIVERSITET

Eksamen

- 4 timers skriftlig eksamen (fedt!)
- Alle hjælpemidler er tilladte! Pånær...
  - ChatGPT og anden generativ AI...
  - Kommunikation med andre mennesker
  - Værktøjer specifik designet til at løse visse problemer (mere om disse, når vi kommer dertil)
- Eksamenssæt fra tidligere år bliver offentliggjort og også brugt i exercises

AALBORG UNIVERSITET

Hvem er jeg?

Yours truly

Obligatorisk slide om mig selv:

- Jeg er 33 år og bor på Nørrebro med min kæreste og datter på 2 år

AALBORG UNIVERSITET

Hvem er jeg?

Yours truly

Obligatorisk slide om mig selv:

- Jeg er 33 år og bor på Nørrebro med min kæreste og datter på 2 år
- Har en bachelor i Softwareudvikling og en kandidat i Computervidenskab fra ITU

AALBORG UNIVERSITET

Hvem er jeg?

Yours truly

Obligatorisk slide om mig selv:

- Jeg er 33 år og bor på Nørrebro med min kæreste og datter på 2 år
- Har en bachelor i Softwareudvikling og en kandidat i Computervidenskab fra ITU
- Blev forelsket i studiet, da jeg selv havde Algoritmer og Datastrukturer på mit 2. semester

AALBORG UNIVERSITET

Hvem er jeg?

Yours truly

Obligatorisk slide om mig selv:

- Jeg er 33 år og bor på Nørrebro med min kæreste og datter på 2 år
- Har en bachelor i Softwareudvikling og en kandidat i Computervidenskab fra ITU
- Blev forelsket i studiet, da jeg selv havde Algoritmer og Datastrukturer på mit 2. semester
- Er på 3. år af min PhD, som omhandler Explainable Reinforcement Learning

AALBORG UNIVERSITET

Hvem er jeg?

Yours truly

Obligatorisk slide om mig selv:

- Jeg er 33 år og bor på Nørrebro med min kæreste og datter på 2 år
- Har en bachelor i Softwareudvikling og en kandidat i Computervidenskab fra ITU
- Blev forelsket i studiet, da jeg selv havde Algoritmer og Datastrukturer på mit 2. semester
- Er på 3. år af min PhD, som omhandler Explainable Reinforcement Learning
- Uhørt stor fan af gyserfilm, melodi grand prix og brætspil

AALBORG UNIVERSITET

Hvem er jeg?

Yours truly

Obligatorisk slide om mig selv:

- Jeg er 33 år og bor på Nørrebro med min kæreste og datter på 2 år
- Har en bachelor i Softwareudvikling og en kandidat i Computervidenskab fra ITU
- Blev forelsket i studiet, da jeg selv havde Algoritmer og Datastrukturer på mit 2. semester
- Er på 3. år af min PhD, som omhandler Explainable Reinforcement Learning
- Uhørt stor fan af gyserfilm, melodi grand prix og brætspil
- Diskuterer gerne politik (og alt andet) med dem som gider

AALBORG UNIVERSITET

Outline

AALBORG UNIVERSITET


BEHOLD
THE ALGORITHM!

Hvad er en algoritme?

Din familie spørger...

Hvad er en algoritme?

Din familie spørger...

- En opskrift der kan få en ellers dum computer til at udføre en bestemt opgave korrekt (og nogle gange hurtigt!)

Hvad er en algoritme?

Din familie spørger...

- En opskrift der kan få en ellers dum computer til at udføre en bestemt opgave korrekt (og nogle gange hurtigt!)
- En veldefineret procedure til at løse et specifikt problem

AALBORG UNIVERSITET

Hvad er en algoritme?

Din familie spørger...

- En opskrift der kan få en ellers dum computer til at udføre en bestemt opgave korrekt (og nogle gange hurtigt!)
- En veldefineret procedure til at løse et specifikt problem
- En sekvens af operationer der transformerer et givent input til et bestemt output
  - Fra en usorteret liste (input) til en sorteret liste (output)
  - Fra et id (input) til et data-element (output)
  - Fra et kort og en destination (input) til en rute (output)

Typiske algoritmiske problemer

Sortering

Input En sekvens A af n tal $(a_1, a_2, \ldots, a_n)$

Typiske algoritmiske problemer

Sortering

Input En sekvens A af n tal $(a_1, a_2, \ldots, a_n)$

Output En permutation $(a'_1, a'_2, \ldots, a'_n)$ af A således at $a'_1 \leq a'_2, \leq, \ldots, \leq a'_n$

AALBORG UNIVERSITET

Typiske algoritmiske problemer

Sortering

Input En sekvens A af n tal $(a_1, a_2, \ldots, a_n)$

Output En permutation $(a'_1, a'_2, \ldots, a'_n)$ af A således at $a'_1 \leq a'_2, \leq, \ldots, \leq a'_n$

Example

- Input sekvens $(17, 2, 19, 6, 4, 21)$
- Output sekvens $(2, 4, 6, 17, 19, 21)$

AALBORG UNIVERSITET

Typiske algoritmiske problemer

Sortering

Input En sekvens A af n tal $(a_1, a_2, \ldots, a_n)$

Output En permutation $(a'_1, a'_2, \ldots, a'_n)$ af A således at $a'_1 \leq a'_2, \leq, \ldots, \leq a'_n$

Example

- Input sekvens $(17, 2, 19, 6, 4, 21)$
- Output sekvens $(2, 4, 6, 17, 19, 21)$

Betyder det noget om vi bruger $\leq$ eller $<$?

Typiske algoritmiske problemer

Sortering

Input En sekvens A af n tal $(a_1, a_2, \ldots, a_n)$

Output En permutation $(a'_1, a'_2, \ldots, a'_n)$ af A således at $a'_1 \leq a'_2, \leq, \ldots, \leq a'_n$

Example

- Input sekvens $(17, 2, 19, 6, 4, 21)$
- Output sekvens $(2, 4, 6, 17, 19, 21)$

Betyder det noget om vi bruger $\leq$ eller $<$?
Kan vi også sortere strings?

Typiske algoritmiske problemer

Sortering

Input En sekvens A af n tal $(a_1, a_2, \ldots, a_n)$

Output En permutation $(a'_1, a'_2, \ldots, a'_n)$ af A således at $a'_1 \leq a'_2, \leq, \ldots, \leq a'_n$

Example

- Input sekvens $(17, 2, 19, 6, 4, 21)$
- Output sekvens $(2, 4, 6, 17, 19, 21)$

Betyder det noget om vi bruger $\leq$ eller $<$?
Kan vi også sortere strings?
Hvad med hunde?

Typiske algoritmiske problemer

Find element

Input En sekvens A af n karakterer $(a_1, a_2, \ldots, a_n)$ og en karakter a, som vi leder efter

Output Et index i hvor vi kan finde elementet a i input sekvensen, 0 hvis a ikke er i A

AALBORG UNIVERSITET

Typiske algoritmiske problemer

Find element

Input En sekvens A af n karakterer $(a_1, a_2, \ldots, a_n)$ og en karakter a, som vi leder efter

Output Et index i hvor vi kan finde elementet a i input sekvensen, 0 hvis a ikke er i A

Example

- Input: ('m', 'f', 'a', 'b', 'k'), 'b'
- Output: 4

Bemærk at bogen generelt bruger 1-indexing (dvs. indicies starter fra 1 i stedet for 0).

Typiske algoritmiske problemer

Shortest path

Input En graf G, en start-knude s_0 og en destinations-knude s_d

Output En liste af knuder, der giver den korteste rute fra s_0 til s_d

Typiske algoritmiske problemer

Shortest path

Input En graf G, en start-knude s_0 og en destinations-knude s_d

Output En liste af knuder, der giver den korteste rute fra s_0 til s_d

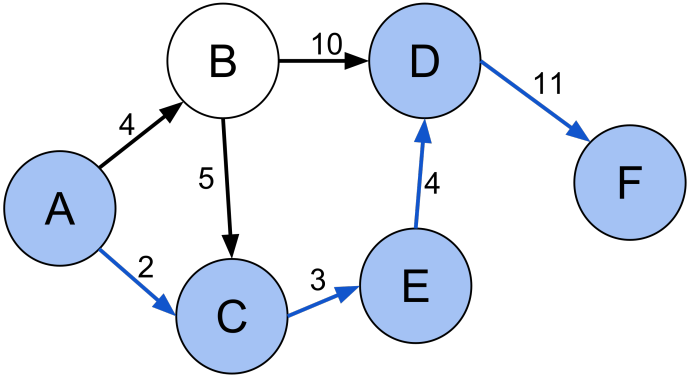

Figure: Korteste rute fra A til F er givet ved de blå kanter (fra Wikipedia)

AALBORG UNIVERSITET

Pseudo-kode

Når vi arbejder algoritmer beskriver vi dem (typisk) i pseudo-kode. Det vil sige kode, der ikke har samme rigide regler og strukturer, som C, C++, Java, Python, etc. men som ligner disse sprog og bruger samme kontrolstrukturer (loops, if/else og så videre).

Målet er at beskrive tydeligt og koncist, hvilke skridt der udføres i algoritmen — hvordan vi går fra et input til et output.

Pseudo-kode

Eksempel: 'Find element'

- Gennemgå alle elementer i A en af gangen og sammenlign med a
- Hvis vi finder a i listen, gem det index i variablen j
- Når vi er færdige med hele sekvensen, returner j

```
Find-Element(A, a)
1  j = 0
2  for i = 1 to A.length
3       if A[i] = a
4            j = i
5  return j
```

AALBORG UNIVERSITET

Pseudo-kode

Eksempel: 'Find element'

- For pseudo-kode tænker vi ikke på software-problemer såsom:
  - Data-abstraktion
  - Modularitet
  - Fejlhåndtering
  - Testning
- Vi kan endda også finde på at bruge tekst fremfor 'kode'

```
Find-Element(A, a)
1  j = 0
2  for i = 1 to A.length
3        if A[i] and a are the same
4              j = i
5  return j
```

Pseudo-kode

At kunne læse, forstå og følge alle skridt i pseudo-kode er absolut nødvendigt for at kunne klare sig godt i kurset og til eksamen.

Pseudo-kode

At kunne læse, forstå og følge alle skridt i pseudo-kode er absolut nødvendigt for at kunne klare sig godt i kurset og til eksamen.

Derfor kommer der træning i dette i dagens exercises.

AALBORG UNIVERSITET

Outline

AALBORG UNIVERSITET

Excercises

Findes på Moodle — arbejd i grupper — spørg ENDELIG!

AALBORG UNIVERSITET

Outline

AALBORG UNIVERSITET

Hvordan studerer man algoritmer?

Hvad er vi egentlig interesserede i?

I computervidenskab har vi massere af problemer, hvor vi kender vores udgangspunkt/input (f.eks. en usorteret liste) og vores ønskede mål/output (f.eks. en sorteret liste). Studiet af algoritmer handler om, hvordan vi bedst kommer fra inputtet til outputtet.

Hvordan studerer man algoritmer?

Hvad er vi egentlig interesserede i?

Kurset her har overordnet set 3 fokuspunkter:

Hvordan studerer man algoritmer?

Hvad er vi egentlig interesserede i?

Kurset her har overordnet set 3 fokuspunkter:

- Introduktion til klassiske algoritmiske problemer, løsninger og teknikker
  - Sorteringsproblemet, shortest-path, look-up
  - Datastrukturer, såsom heaps, binære søgetræer og grafer
  - Divide-and-conquer, greedy algorithms, dynamic programming

AALBORG UNIVERSITET

Hvordan studerer man algoritmer?

Hvad er vi egentlig interesserede i?

Kurset her har overordnet set 3 fokuspunkter:

- Introduktion til klassiske algoritmiske problemer, løsninger og teknikker
  - Sorteringsproblemet, shortest-path, look-up
  - Datastrukturer, såsom heaps, binære søgetræer og grafer
  - Divide-and-conquer, greedy algorithms, dynamic programming
- Korrekthed — hvordan kan vi overbevise os selv om, at algoritmen virker, som forventet?

AALBORG UNIVERSITET

Hvordan studerer man algoritmer?

Hvad er vi egentlig interesserede i?

Kurset her har overordnet set 3 fokuspunkter:

- Introduktion til klassiske algoritmiske problemer, løsninger og teknikker
  - Sorteringsproblemet, shortest-path, look-up
  - Datastrukturer, såsom heaps, binære søgetræer og grafer
  - Divide-and-conquer, greedy algorithms, dynamic programming
- Korrekthed — hvordan kan vi overbevise os selv om, at algoritmen virker, som forventet?
- Kompleksitet
  - Hvor meget tid tager algoritmen?
  - Hvor meget plads (hukommelse) kræver algoritmen?

AALBORG UNIVERSITET

Korrekthed

Definition (Korrekthed)

En algoritme er korrekt, hvis den — givet et korrekt input — med garanti returnerer det korrekte output. Vi siger, at algoritmen løser det givne 'computational' problem.

Korrekthed

Definition (Korrekthed)

En algoritme er korrekt, hvis den — givet et korrekt input — med garanti returnerer det korrekte output. Vi siger, at algoritmen løser det givne 'computational' problem.

Hvis din sorteringsalgoritme, f.eks., kun returnerer en korrekt sorteret liste, hvis inputtet ikke indeholder dupletter, så er algoritmen ikke korrekt — medmindre det var en del af problemspecifikationen!

AALBORG UNIVERSITET

Korrekthed

Eksempel: 'Find-Element'

Input En sekvens A og et element a

Output Det første index i A hvor a kan findes og 0, hvis ikke a er i A

Korrekthed

Eksempel: 'Find-Element'

Input En sekvens A og et element a

Output Det første index i A hvor a kan findes og 0, hvis ikke a er i A

Find-Element(A, a)

```
1  j = 0
2  for i = 1 to A.length
3        if A[i] = a
4              j = i
5  return j
```

Find-Element-v2(A, a)

```
1  j = A.length
2  while i > 0
3        if A[i] = a
4              j = i
5        i = i − 1
6  return j
```

Er begge algoritmer korrekte?

AALBORG UNIVERSITET

Korrekthed

Eksempel: 'Find-Element'

Input En sekvens A og et element a

Output Det første index i A hvor a kan findes og 0, hvis ikke a er i A

Find-Element(A, a)

```
1  j = 0
2  for i = 1 to A.length
3        if A[i] = a
4              j = i
5  return j
```

Find-Element-v2(A, a)

```
1  j = A.length
2  while i > 0
3        if A[i] = a
4              j = i
5        i = i − 1
6  return j
```

Er begge algoritmer korrekte? Nej!

AALBORG UNIVERSITET

Korrekthed

Eksempel: 'Find-Element'

Input En sekvens A og et element a

Output Det første index i A hvor a kan findes og 0, hvis ikke a er i A

Find-Element(A, a)

```
1  j = 0
2  for i = 1 to A.length
3        if A[i] = a
4              j = i
5  return j
```

Find-Element-v2(A, a)

```
1  j = A.length
2  while i > 0
3        if A[i] = a
4              j = i
5        i = i − 1
6  return j
```

Returnerer det sidste index, hvor a optræder.

Er begge algoritmer korrekte? Nej!

AALBORG UNIVERSITET

Korrekthed

Eksempel: 'Find-Element'

Input En sekvens A og et element a

Output Det første index i A hvor a kan findes og 0, hvis ikke a er i A

Find-Element(A, a)

```
1  j = 0
2  for i = 1 to A.length
3        if A[i] = a
4              j = i
5  return j
```

Returnerer det sidste index, hvor a optræder.

Find-Element-v2(A, a)

```
1  j = A.length
2  while i > 0
3        if A[i] = a
4              j = i
5        i = i − 1
6  return j
```

Returnerer det første index, hvor a optræder.

Er begge algoritmer korrekte? Nej!

Korrekthed

Eksempel: 'Find-Element'

Input En sekvens A og et element a

Output Det sidste index i A hvor a kan findes og 0, hvis ikke a er i A

Korrekthed

Eksempel: 'Find-Element'

Input En sekvens A og et element a

Output Det sidste index i A hvor a kan findes og 0, hvis ikke a er i A

Find-Element(A, a)

```
1  j = 0
2  for i = 1 to A.length
3        if A[i] = a
4              j = i
5  return j
```

Find-Element-v3(A, a)

```
1  i = A.length
2  while i > 0 and A[i] ≠ a
3        i = i − 1
4  return i
```

Er begge algoritmer korrekte?

AALBORG UNIVERSITET

Korrekthed

Eksempel: 'Find-Element'

Input En sekvens A og et element a

Output Det sidste index i A hvor a kan findes og 0, hvis ikke a er i A

Find-Element(A, a)

```
1  j = 0
2  for i = 1 to A.length
3        if A[i] = a
4              j = i
5  return j
```

Find-Element-v3(A, a)

```
1  i = A.length
2  while i > 0 and A[i] ≠ a
3        i = i − 1
4  return i
```

Er begge algoritmer korrekte? Ja — der findes mange korrekte algoritmer for at løse et problem.

Næste gang ser vi på loop invarianter, som er en måde at bevise, at en iterativ algoritme er korrekt.

Kompleksitet

Tid og plads

- Computere er hurtige, men ikke uendeligt hurtige - og de har meget hukommelse, men ikke uendelig hukommelse
- Når vi taler om en algoritmes kompleksitet, så taler vi om den tid (runtime) og plads (space) den kræver
- Nogle gange (men IKKE altid!) er der et trade-off mellem kompleksiteten i tid og kompleksiteten i plads
- Forskellige algoritmer kan løse det samme problem korrekt, men have meget forskellig kompleksitet

AALBORG UNIVERSITET

Når vi analyserer algoritmer, er der en række ting, vi er interesserede i:

Når vi analyserer algoritmer, er der en række ting, vi er interesserede i:

- Forudsige performance
  - Hvor lang tid/meget plads kræver min algoritme?
  - Er det overhovedet muligt at løse problemet med den tid og den plads, jeg har til rådighed?

Når vi analyserer algoritmer, er der en række ting, vi er interesserede i:

- Forudsige performance
  - Hvor lang tid/meget plads kræver min algoritme?
  - Er det overhovedet muligt at løse problemet med den tid og den plads, jeg har til rådighed?
- Sammenligne algoritmer
  - Hvilken algoritme er bedst i en given situation?

Når vi analyserer algoritmer, er der en række ting, vi er interesserede i:

- Forudsige performance
  - Hvor lang tid/meget plads kræver min algoritme?
  - Er det overhovedet muligt at løse problemet med den tid og den plads, jeg har til rådighed?
- Sammenligne algoritmer
  - Hvilken algoritme er bedst i en given situation?
- Give garantier
  - Algoritmen vil aldrig bruge mere tid end X
  - Algoritmen kræver som minimum X GB hukommelse

Men... Vi gider faktisk sjældent blive særligt konkrete (vi er jo videnskabsfolk, ikke praktikere!). Hvorfor ikke?

Men... Vi gider faktisk sjældent blive særligt konkrete (vi er jo videnskabsfolk, ikke praktikere!). Hvorfor ikke?

- Den præcise tid en algoritme tager afhænger af mange ting: computerens hardware, programmeringssproget, mængden af andre processer igang, temperaturen i rummet og ikke mindst inputstørrelsen
- Den præcise plads en algoritme kræver afhænger også af mange ting: computerarkitekturen, data repræsentationen og ikke mindst inputstørrelsen

Hvad vi derfor går efter er, at bestemme kompleksiteten af en algoritme som en funktion af inputstørrelsen. Det vil sige, hvordan udvikler tiden (og pladsen) sig, når størrelsen af inputtet stiger?

Hvad vi derfor går efter er, at bestemme kompleksiteten af en algoritme som en funktion af inputstørrelsen. Det vil sige, hvordan udvikler tiden (og pladsen) sig, når størrelsen af inputtet stiger?

- Husk, at en algoritme er bare en række operationer (computational steps), der hver især tager en vis mængde tid at udføre (som afhænger af alle de der mærkelige ting, vi ikke har kontrol over)
- Vi siger derfor, at tiden som det tager en algoritme at køre, gives ved antallet af operationer, den skal igennem for at generere sit output
- Ofte er inputtet en mængde af elementer (f.eks. en liste). Hvis der er n elementer i inputtet siger vi, at inputstørrelsen er n
- Når tiden er funktion af n noterer vi den som $T(n)$

Kompleksitet

Eksempel: 'Find-Element'

Vi siger af hver linie i er 1 operation og tager konstant tid c_i. For afgøre tiden $T(n)$ skal vi tælle, hvor mange gange hver linie udføres:

Find-Element(A, a)	tid $\times$ antal gange
1 $j = 0$	
2 **for** $i = 1$ **to** $A.length$	
3 **if** $A[i] = a$	
4 $j = i$	
5 **return** j	

$$T(n) = \quad + \quad + \quad + \quad +$$
$$=$$

Kompleksitet

Eksempel: 'Find-Element'

Vi siger af hver linie i er 1 operation og tager konstant tid c_i. For afgøre tiden $T(n)$ skal vi tælle, hvor mange gange hver linie udføres:

Find-Element(A, a)	tid $\times$ antal gange
1 $j = 0$	$c_1 \times 1$
2 **for** $i = 1$ **to** $A.length$	
3 **if** $A[i] = a$	
4 $j = i$	
5 **return** j	

$$T(n) = c_1 + \quad + \quad + \quad +$$
$$=$$

Kompleksitet

Eksempel: 'Find-Element'

Vi siger af hver linie i er 1 operation og tager konstant tid c_i. For afgøre tiden $T(n)$ skal vi tælle, hvor mange gange hver linie udføres:

Find-Element(A, a)	tid $\times$ antal gange
1 $j = 0$	$c_1 \times 1$
2 **for** $i = 1$ **to** $A.length$	$c_2 \times n + 1$
3 **if** $A[i] = a$	
4 $j = i$	
5 **return** j	

$$T(n) = c_1 + c_2(n+1) + \quad + \quad + \quad$$
$$= \quad$$

Kompleksitet

Eksempel: 'Find-Element'

Vi siger af hver linie i er 1 operation og tager konstant tid c_i. For afgøre tiden $T(n)$ skal vi tælle, hvor mange gange hver linie udføres:

Find-Element(A, a)	tid $\times$ antal gange
1 $j = 0$	$c_1 \times 1$
2 **for** $i = 1$ **to** $A.length$	$c_2 \times n + 1$
3 **if** $A[i] = a$	$c_3 \times n$
4 $j = i$	
5 **return** j	

$$T(n) = c_1 + c_2(n+1) + c_3 \cdot n + \qquad +$$
$$= \qquad$$

Kompleksitet

Eksempel: 'Find-Element'

Vi siger af hver linie i er 1 operation og tager konstant tid c_i. For afgøre tiden $T(n)$ skal vi tælle, hvor mange gange hver linie udføres:

Find-Element(A, a)	tid $\times$ antal gange
1 $j = 0$	$c_1 \times 1$
2 **for** $i = 1$ **to** $A.length$	$c_2 \times n + 1$
3 **if** $A[i] = a$	$c_3 \times n$
4 $j = i$	$c_4 \times n_a$
5 **return** j	

$$T(n) = c_1 + c_2(n+1) + c_3 \cdot n + c_4 \cdot n_a +$$
$$=$$

Kompleksitet

Eksempel: 'Find-Element'

Vi siger af hver linie i er 1 operation og tager konstant tid c_i. For afgøre tiden $T(n)$ skal vi tælle, hvor mange gange hver linie udføres:

Find-Element(A, a)	tid $\times$ antal gange
1 $j = 0$	$c_1 \times 1$
2 **for** $i = 1$ **to** $A.length$	$c_2 \times n + 1$
3 **if** $A[i] = a$	$c_3 \times n$
4 $j = i$	$c_4 \times n_a$
5 **return** j	$c_5 \times 1$

$$T(n) = c_1 + c_2(n+1) + c_3 \cdot n + c_4 \cdot n_a + c_5$$
$$= $$

Kompleksitet

Eksempel: 'Find-Element'

Vi siger af hver linie i er 1 operation og tager konstant tid c_i. For afgøre tiden $T(n)$ skal vi tælle, hvor mange gange hver linie udføres:

Find-Element(A, a)	tid $\times$ antal gange
1 $j = 0$	$c_1 \times 1$
2 **for** $i = 1$ **to** $A.length$	$c_2 \times n + 1$
3 **if** $A[i] = a$	$c_3 \times n$
4 $j = i$	$c_4 \times n_a$
5 **return** j	$c_5 \times 1$

$$\begin{aligned} T(n) &= c_1 + c_2(n+1) + c_3 \cdot n + c_4 \cdot n_a + c_5 \\ &= n(c_2 + c_3) + c_4 \cdot n_a + c_1 + c_2 + c_5 \end{aligned}$$

AALBORG UNIVERSITET

- Den eksakte køretid for Find-Element er dermed:
  $T(n) = n(c_2 + c_3) + c_4 \cdot n_a + c_1 + c_2 + c_5$

- Den eksakte køretid for Find-Element er dermed:
  $T(n) = n(c_2 + c_3) + c_4 \cdot n_a + c_1 + c_2 + c_5$
- Men i best case, hvor a ikke er i A, er $n_a = 0$, så: $T(n) = n(c_2 + c_3) + c_1 + c_2 + c_5$

- Den eksakte køretid for Find-Element er dermed:
  $T(n) = n(c_2 + c_3) + c_4 \cdot n_a + c_1 + c_2 + c_5$
- Men i best case, hvor a ikke er i A, er $n_a = 0$, så: $T(n) = n(c_2 + c_3) + c_1 + c_2 + c_5$
- I worst case, hvor alle elementer i A er a, har vi: $T(n) = n(c_2 + c_3 + c_4) + c_1 + c_2 + c_5$

Kompleksitet

Worst case analyse

- Vi er (næsten altid) kun interesseret i en algoritmes worst case
- Dette gør analysen lettere — vi behøver ikke forholde os til alle de forskellige måder inputtet kunne påvirke tiden (og pladsen), kun den værst tænkelige
- Det giver os en garanti for, at algoritmen aldrig vil tage længere tid end det
- For mange algoritmer er worst case næsten det samme som average case

Nu gør vi noget frækt.

Nu gør vi noget frækt. Som sagt behandler vi tiden hver enkelt operation tager som en konstant. Det vil sige, den ændrer sig ikke. En konstant plus en konstant er også en konstant. Derfor siger, at $c_2 + c_3 + c_4 = a$ (her er a ikke vores element, men en ny konstant). Og vi siger at $c_1 + c_2 + c_5 = b$, hvor b også er en ny konstant.

$$T(n) = (c_2 + c_3 + c_4)\ n + c_1 + c_2 + c_5$$

Nu gør vi noget frækt. Som sagt behandler vi tiden hver enkelt operation tager som en konstant. Det vil sige, den ændrer sig ikke. En konstant plus en konstant er også en konstant. Derfor siger, at $c_2 + c_3 + c_4 = a$ (her er a ikke vores element, men en ny konstant). Og vi siger at $c_1 + c_2 + c_5 = b$, hvor b også er en ny konstant.

$$T(n) = an + c_1 + c_2 + c_5$$

Nu gør vi noget frækt. Som sagt behandler vi tiden hver enkelt operation tager som en konstant. Det vil sige, den ændrer sig ikke. En konstant plus en konstant er også en konstant. Derfor siger, at $c_2 + c_3 + c_4 = a$ (her er a ikke vores element, men en ny konstant). Og vi siger at $c_1 + c_2 + c_5 = b$, hvor b også er en ny konstant.

$$T(n) = an + b$$

Nu gør vi noget frækt. Som sagt behandler vi tiden hver enkelt operation tager som en konstant. Det vil sige, den ændrer sig ikke. En konstant plus en konstant er også en konstant. Derfor siger, at $c_2 + c_3 + c_4 = a$ (her er a ikke vores element, men en ny konstant). Og vi siger at $c_1 + c_2 + c_5 = b$, hvor b også er en ny konstant.

$$T(n) = an + b$$

Lineær vækst

Bemærk at T er en lineær funktion af n med den klassiske form $an + b$. Det vil sige, når n vokser med 1 vokser $T(n)$ med a og når $n = 0$ er $T(n) = b$.

Nu gør vi noget endnu mere frækt.

Nu gør vi noget endnu mere frækt.

- Vi har allerede abstraheret det konkrete tidsforbrug væk ved at introducere ukendte konstanter ($c_1, c_2, \ldots$)

Nu gør vi noget endnu mere frækt.

- Vi har allerede abstraheret det konkrete tidsforbrug væk ved at introducere ukendte konstanter ($c_1, c_2, \ldots$)
- Vi kan abstrahere endnu mere ved kun at kigge på order of growth

Nu gør vi noget endnu mere frækt.

- Vi har allerede abstraheret det konkrete tidsforbrug væk ved at introducere ukendte konstanter ($c_1, c_2, \ldots$)
- Vi kan abstrahere endnu mere ved kun at kigge på order of growth
- Vi ignorerer konstanter og ser kun på, om tiden udvikler sig lineært, logaritmisk, kubisk, etc. med inputtet

Nu gør vi noget endnu mere frækt.

- Vi har allerede abstraheret det konkrete tidsforbrug væk ved at introducere ukendte konstanter ($c_1, c_2, \ldots$)
- Vi kan abstrahere endnu mere ved kun at kigge på order of growth
- Vi ignorerer konstanter og ser kun på, om tiden udvikler sig lineært, logaritmisk, kubisk, etc. med inputtet
- For Find-Element får vi dermed en worst case køretid $T(n) = O(n)$

Nu gør vi noget endnu mere frækt.

- Vi har allerede abstraheret det konkrete tidsforbrug væk ved at introducere ukendte konstanter ($c_1, c_2, \ldots$)
- Vi kan abstrahere endnu mere ved kun at kigge på order of growth
- Vi ignorerer konstanter og ser kun på, om tiden udvikler sig lineært, logaritmisk, kubisk, etc. med inputtet
- For Find-Element får vi dermed en worst case køretid $T(n) = O(n)$
- I næste forelæsning går vi dybere ind i forskellige notationer for køretid, såsom O, Θ, Ω

Opsamling
Dagens temaer

- Hvad er algoritmer?
- Problemspecificering ved inputs og outputs
- Pseudo-kode til at beskrive algoritmer
- Algoritmers korrekthed
- Kompleksitet og worst case analyse

AALBORG UNIVERSITET

Tak for i dag!

Flere exercises..

Den bedste måde ikke at snyde sig selv på er lave exercises!

AALBORG UNIVERSITET