TDDI16 – Föreläsning 1

Introduktion och komplexitet

Filip Strömbäck

LINKÖPING
UNIVERSITY

1 Kursinformation
2 Varför DALG?
3 Algoritmer
4 Hur körs ett program – Modell
5 Ordonotation - Tillväxthastighet
6 Beräkna tidskomplexitet
7 Nästa gång

LINKÖPING UNIVERSITY

Resurser

- Kurshemsida: `https://www.ida.liu.se/~TDDI16/`
- Litteratur: OpenDSA, (Introduction to Algorithms)

Kursledare	Filip Strömbäck
	(Magnus Nielsen)
Kursassistent	Christoffer Holm
Assistenter	Axel Frosthagen
	Herman Appelgren
	Rasmus Rynell
	Elin Frankell
Administratör	Annelie Almquist

Examination

UPG1	Uppgifter i OpenDSA, 2hp (U, G)
LAB1	Laborationer, 2hp (U, G) 4 laborationsuppgifter
DAT1	Datortentamen, 2hp (U, 3, 4, 5) Troligtvis hemtentamen HT 2021. Frågor om användandet av datastrukturer och algoritmer. Extrauppgifter och deadline på laborationer ger upp till 10% bonus mot högre betyg.

Distans HT2021

Kursen går huvudsakligen på distans HT2021:

- FÖ: Distans – Zoom
- LE: Distans – Teams
- LA: Hybrid – Teams/På plats (ca 50% av tiden)
- Tenta: Gissningsvis distans

OpenDSA – Digital kursbok

- Digital kursbok med interaktiva övningar
- Logga in med LiU-id, dubbelkolla rubrik
- För att klara UPG1 ska ni innan kursens slut ha löst **alla** interaktiva övningar
- Avklarade kapitel markeras med en bock
- Klicka på ert namn för att kontrollera vad som är kvar

Föreläsningar

- Fokus på hur info från OpenDSA kan *användas*
- Slides finns på kurshemsidan, men är *inte* tänkta att kunna läsas i isolation
- Efter varje föreläsning finns 2 extrauppgifter i KATTIS
  - Relaterade till det som tagits upp
  - Ett enklare och ett svårare
  - Löses individuellt
- Ställ hemskt gärna frågor (antingen via chatten, eller genom att räcka upp handen)!

Laborationer

- Parvis
- Anmälan sker i Webreg (länk på kurshemsidan)
- Anmäl er senast **i morgon** (inför lektionen)
- Innehåll:
  1. AVL-träd
  2. Knäcka lösenord
  3. Ordkedjor
  4. Mönsterigenkänning
- Notera: Givna testfall är inte heltäckande. Skriv också egna testfall (med verktyg på kurshemsidan)

Laborationer

- Finns 6 grupper: A_1/2, B_1/2, IP_1/2
- IP_x är för IP-studenter
- A_x och B_x är för övriga
- I labbsal:
  - Pass märkta 1 eller 2, de delgrupperna får vara på plats.
  - Skärmarna mellan datorer ska vara på plats, 1 student per dator.
  - Samarbete via Zoom, Teams, VS code, etc.

Planering

Vecka	Fö	Lab
35	Komplexitet, Linjära strukturer	----
36	Träd, AVL-träd	1---
37	Hashning, meet-in-the-middle	12--
38	Grafer och kortaste vägen	-23-
39	Fler grafalgoritmer	--3-
40	Sortering	--34
41	Repetition	---4

Se kommentarer i TimeEdit för deadlines!

Ändringar från förra året

- Reviderat avsnitt om selection sort
- Lite mer tid i början av kursen (le1 efter fö2)

1 Kursinformation
2 Varför DALG?
3 Algoritmer
4 Hur körs ett program – Modell
5 Ordonotation - Tillväxthastighet
6 Beräkna tidskomplexitet
7 Nästa gång

LINKÖPING UNIVERSITY

Hur man löser alla problem enligt Richard Feynman:

1. Write down the problem
2. Think really hard
3. Write down the answer

Hur man löser alla problem enligt Richard Feynman:

1. Write down the problem
2. Think really hard
3. Write down the answer

DALG hjälper oss med steg 2!

Liknelse

Du vill gräva en stor grop.

- Utan verktyg: 2 dagar
- Med spade: 5 timmar
- Med grävskopa: 1 timme
- ...

Om du har tillgång till dynamit kan du dessutom lösa ett svårare problem: att gräva en ”grop” i en bergshäll.

I programmering...

Du vill lösa ett svårt problem.

- Utan DALG-kunskap: 1 månad
- Kan använda datastrukturer: 1 vecka
- Känner till lämpliga algoritmer: 1 dag

Om du dessutom vet hur verktygen fungerar, kan du anpassa dem så att du kan lösa mer komplicerade problem, och så att lösningen blir mer effektiv.

Varför DALG?

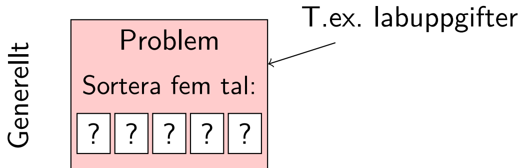
- *Veta* vilka verktyg som finns, och hur de fungerar
- *Kunna använda* verktygen som finns tillgängliga
  - ...för att kunna *implementera* lösningar smidigt
  - ...för att kunna *uttrycka* sig bättre
  - ...för att kunna *resonera* på en högre abstraktionsnivå
- *Kunna välja* rätt verktyg
  - *Kunna analysera* och *värdera* olika lösningar
- *Kunna anpassa* standardalgoritmer- eller datastrukturer så att de kan lösa ditt specifika problem.
- *Känna till* gränserna för vad som är möjligt att göra

För att effektivt kunna lösa svåra problem, eller problem med stora mängder data.

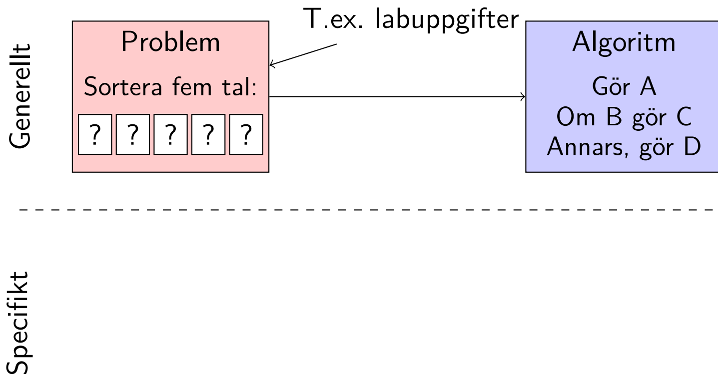
1 Kursinformation
2 Varför DALG?
3 Algoritmer
4 Hur körs ett program – Modell
5 Ordonotation - Tillväxthastighet
6 Beräkna tidskomplexitet
7 Nästa gång

LINKÖPING UNIVERSITY

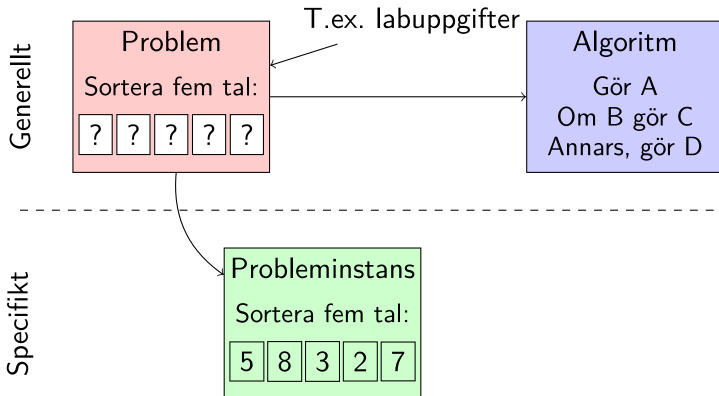
Problem, program och algoritmer



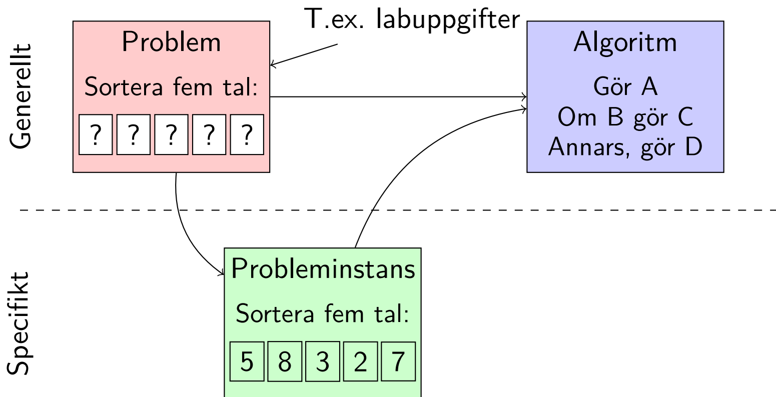
Problem, program och algoritmer



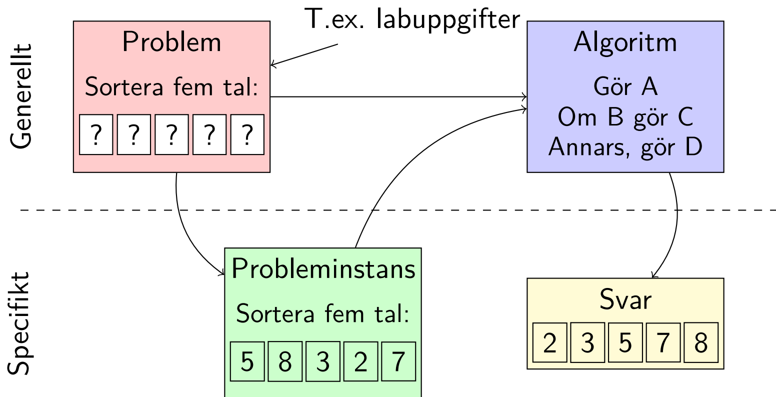
Problem, program och algoritmer



Problem, program och algoritmer



Problem, program och algoritmer



Algoritmanalys?

Det finns oftast flera olika algoritmer som löser ett givet problem. Vilken ska vi välja?

Algoritmanalys?

Det finns oftast flera olika algoritmer som löser ett givet problem. Vilken ska vi välja?

- Den som är snabbast (den som alltid terminerar)
- Den som använder minst minne
- Den som kan köras parallellt
- Den som gör minst antal frågor till externa tjänster
- ...

Algoritmanalys!

Exempel: Algoritm som beräknar fibonaccital

1, 1, 2, 3, 5, 8, 13, 21, ...

$$f(n) = \begin{cases} 1, & \text{om } n = 1 \\ 1, & \text{om } n = 2 \\ f(n-1) + f(n-2) & \text{annars} \end{cases}$$

Vilken implementation är snabbast?

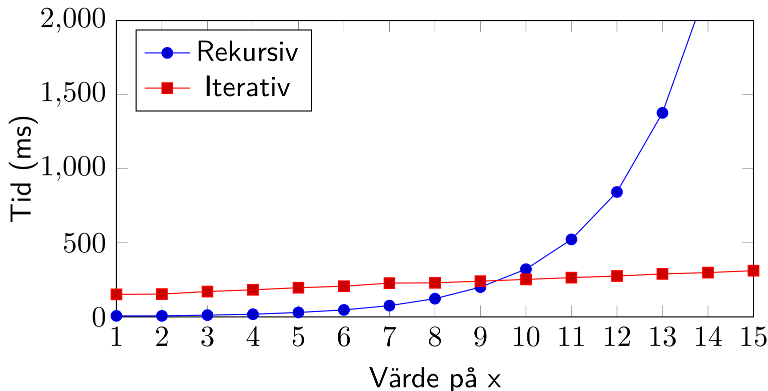
```
int fib1(int n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib1(n - 1)
      + fib1(n - 2);
  }
}
```

```
int fib2(int n) {
  vector<int> res(n+1, 1);

  for (int i=3; i<=n; i++) {
    res[i] = res[i - 1]
        + res[i - 2];
  }

  return res[n];
}
```

Vi mäter! 1 000 000 körningar per indata



Vad är intressant?

Tiden för **små** indata är oftast väldigt liten, knappt mätbar. Alltså:

- Vi är intresserade av vad som händer för **stora** indata
- Vi vill kunna jämföra olika algoritmer
- Vi är intresserade av **helheten**

Vi behöver ett sätt att resonera om detta!

Vad är intressant?

Tiden för **små** indata är oftast väldigt liten, knappt mätbar. Alltså:

- Vi är intresserade av vad som händer för **stora** indata
- Vi vill kunna jämföra olika algoritmer
- Vi är intresserade av **helheten**

Vi behöver ett sätt att resonera om detta!

Notera: Små fall kan också vara viktiga, men börja med en effektiv algoritm och optimera den sen vid behov.

1 Kursinformation
2 Varför DALG?
3 Algoritmer
4 Hur körs ett program – Modell
5 Ordonotation - Tillväxthastighet
6 Beräkna tidskomplexitet
7 Nästa gång

LINKÖPING UNIVERSITY

Hur körs ett program?

Idé: räkna antalet ”operationer” som krävs:

- Aritmetiska operationer
- Tilldelingar
- Läsning/skrivning av minnet
- ...

Vi antar att alla ”enkla” operationer tar lika lång tid

Tidsåtgång – exempel

```
int a(int n) {
  return n * 2;
}
```

Tidsåtgång – exempel

```
int a(int n) {
  return n * 2;
}
```

$\Rightarrow t(n) = 2$

Tidsåtgång – exempel

```
int b(int n) {
  int result = 1;
  for (int i = 1; i < n; i++)
    result *= i;
  return result;
}
```

Tidsåtgång – exempel

```
int b(int n) {
  int result = 1;
  for (int i = 1; i < n; i++)
    result *= i;
  return result;
}
```

$$\Rightarrow t(n) = 3 + 3n$$

Tidsåtgång – exempel

```
int fib2(int n) {
  vector<int> res(n+1, 1);

  for (int i=3; i<=n; i++) {
    res[i] = res[i - 1]
         + res[i - 2];
  }

  return res[n];
}
```

Tidsåtgång – exempel

```
int fib2(int n) {
  vector<int> res(n+1, 1);

  for (int i=3; i<=n; i++) {
    res[i] = res[i - 1]
        + res[i - 2];
  }

  return res[n];
}
```

$$\Rightarrow t(n) = 4 + 7(n - 3)$$

Tidsåtgång – exempel

```
int fib1(int n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib1(n - 1)
      + fib1(n - 2);
  }
}
```

Tidsåtgång – exempel

```
int fib1(int n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib1(n - 1)
      + fib1(n - 2);
  }
}
```

$$\Rightarrow t(n) = \begin{cases} 3 & \text{om } n \leq 2 \\ 4 + t(n-1) + t(n-2) & \text{annars} \end{cases}$$

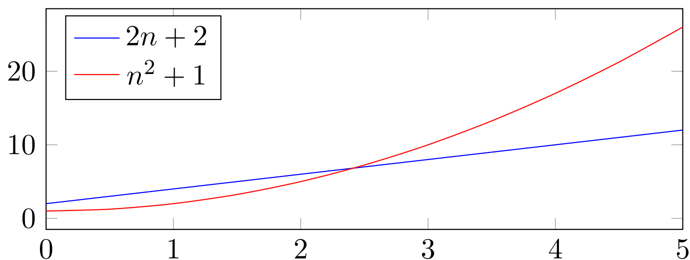
1 Kursinformation
2 Varför DALG?
3 Algoritmer
4 Hur körs ett program – Modell
5 Ordonotation - Tillväxthastighet
6 Beräkna tidskomplexitet
7 Nästa gång

LINKÖPING UNIVERSITY

Idé

- Vi delar in funktioner i olika grupper, där varje grupp växer ungefär lika snabbt för stora n.
- För att veta vilka grupperna är behöver vi kunna jämföra funktioner. Vi säger att $f(n) \in \mathcal{O}(g(n))$ omm det finns några $0 \leq c < \infty$ och $0 \leq n_0 < \infty$ så att $f(n) \leq cg(n)$ för alla $n \geq n_0$.
- Detta innebär att $f(n)$ inte växer snabbare än $g(n)$. Man kan tänka sig att $f(n) \leq g(n)$ gäller för tillräckligt stora n (även om det inte riktigt stämmer).

Exempel

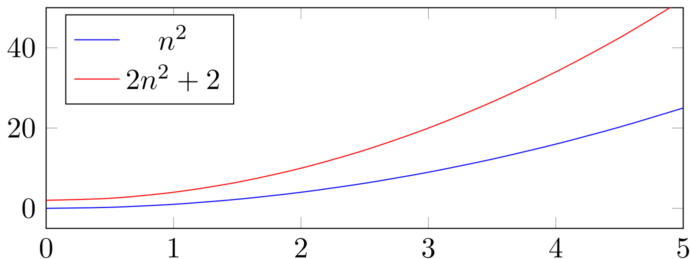


$2n + 2 \leq c(n^2 + 1)$ för $c = 1$ och $n \geq 3$

Alltså är $2n + 2 \in \mathcal{O}(n^2 + 1)$

Dock är $n^2 + 1 \notin \mathcal{O}(2n + 2)$

Exempel

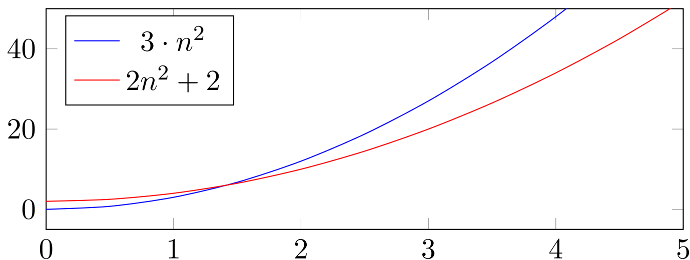


$n^2 \in \mathcal{O}(2n^2 + 2)$ (enkelt att se)

$2n^2 + 2 \in \mathcal{O}(n^2)$ gäller också. Hur?

Alltså: $\mathcal{O}(n^2) = \mathcal{O}(2n^2 + 2)$

Exempel

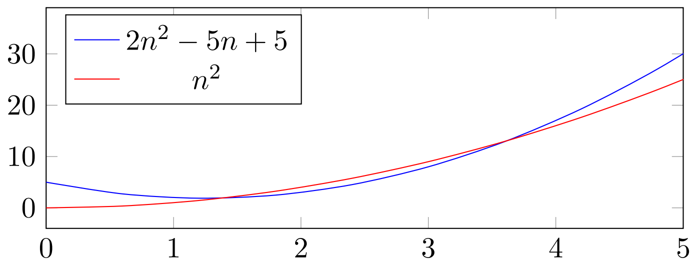


$n^2 \in \mathcal{O}(2n^2 + 2)$ (enkelt att se)

$2n^2 + 2 \in \mathcal{O}(n^2)$ gäller också. Hur?

Alltså: $\mathcal{O}(n^2) = \mathcal{O}(2n^2 + 2)$

Exempel

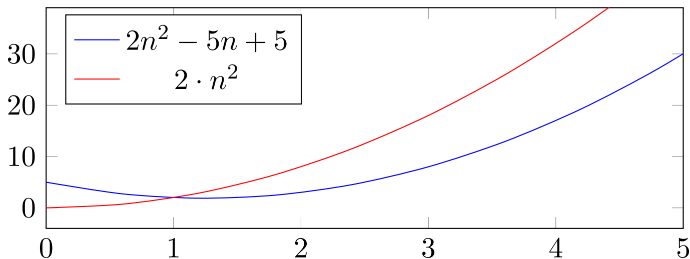


$n^2 \in \mathcal{O}(2n^2 - 5n + 5)$ (enkelt att se)

$2n^2 - 5n + 5 \in \mathcal{O}(n^2)$ gäller också. Hur?

Alltså: $\mathcal{O}(2n^2 - 5n + 5) = \mathcal{O}(n^2)$

Exempel



$n^2 \in \mathcal{O}(2n^2 - 5n + 5)$ (enkelt att se)

$2n^2 - 5n + 5 \in \mathcal{O}(n^2)$ gäller också. Hur?

Alltså: $\mathcal{O}(2n^2 - 5n + 5) = \mathcal{O}(n^2)$

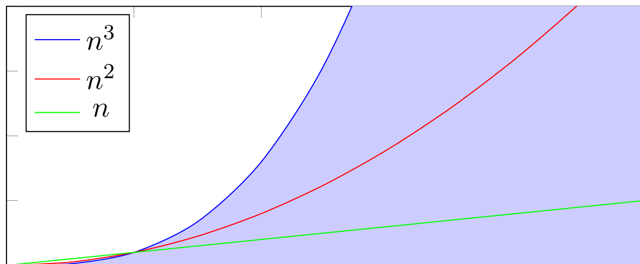
Observationer – Regler

- I en summa av termer kan vi förenkla bort alla termer förutom den snabbast växande termen
  Ex: $n^2 + n + 1 \in \mathcal{O}(n^2)$
- Konstanter, både konstanta termer (ex. $n + 5$) och konstanter framför termer (ex. $5n$) kan förenklas bort
- Om $f(n) \in \mathcal{O}(g(n))$, och $g(n) \in \mathcal{O}(f(n))$ så är $\mathcal{O}(f(n)) = \mathcal{O}(g(n))$. Då gäller även $f(n) \in \Theta(g(n))$ samt $g(n) \in \Theta(f(n))$.
- Alltså kan vi representera våra olika ”grupper” i form av den mest förenklade formeln

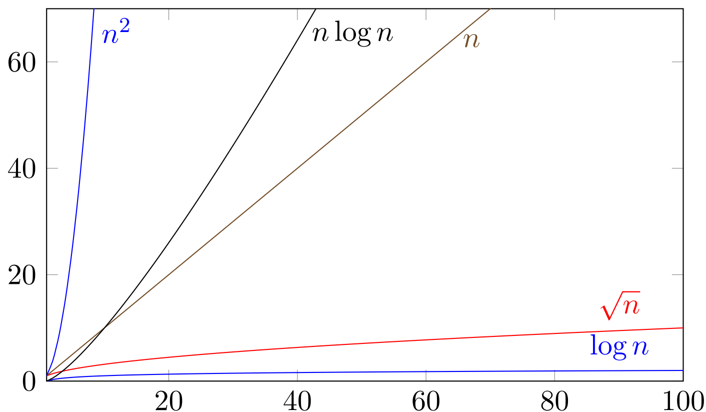
Förenkling med regler

Med hjälp av observationerna kan vi enklare få en uppfattning om förhållandet mellan olika funktioner.

Här kan vi se att $n, n^2 \in \mathcal{O}(n^3)$:



Vanliga uttryck för tidskomplexitet



1 Kursinformation
2 Varför DALG?
3 Algoritmer
4 Hur körs ett program – Modell
5 Ordonotation - Tillväxthastighet
6 Beräkna tidskomplexitet
7 Nästa gång

LINKÖPING UNIVERSITY

Idé

- Vi vill "mäta" tiden det tar att köra en algoritm
- Vi såg att konstanter i uttryck inte spelar någon roll

$\Rightarrow$ Den exakta körtiden spelar ingen ingen roll, vi är bara intresserade av **hur fort den växer**

$\Rightarrow$ Vi kan anta att varje operation tar 1 tidsenhet

Exempel

```
int a(int n) {
  return n * 2;
}
```

$$\Rightarrow t(n) = 2 \in \mathcal{O}(1)$$

Exempel

```
int b(int n) {
  int result = 1;
  for (int i = 1; i < n; i++)
    result *= i;
  return result;
}
```

$$\Rightarrow t(n) = 3 + 3n \in \mathcal{O}(n)$$

Exempel

```
int c(int n) {
  int sum = 0;
  for (int i = 0; i < n; i++)
    sum += b(i);
  return sum;
}
```

Exempel

```
int fib2(int n) {
  vector<int> res(n+1, 1);

  for (int i=3; i<=n; i++) {
    res[i] = res[i - 1]
        + res[i - 2];
  }

  return res[n];
}
```

$$\Rightarrow t(n) = 4 + 7(n - 3) \in \mathcal{O}(n)$$

Den rekursiva varianten

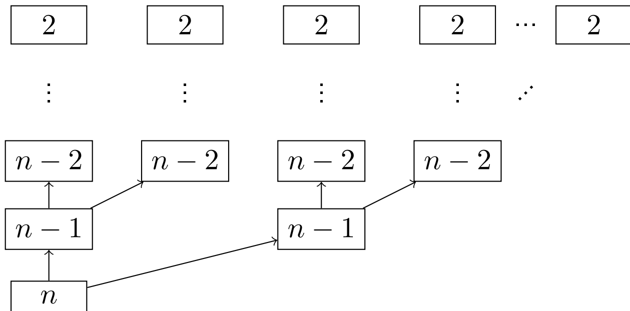
```
int fib1(int n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib1(n - 1)
      + fib1(n - 2);
  }
}
```

Förenklat

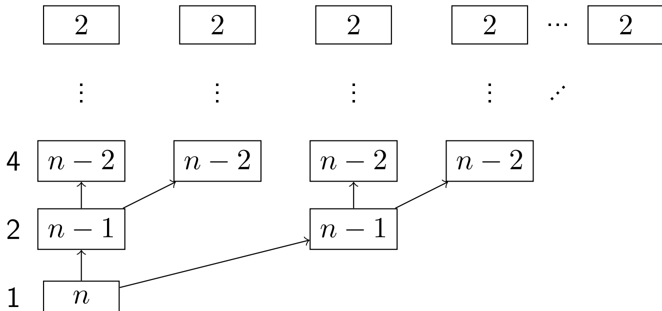
```
int fib1(int n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib1(n - 1)
      + fib1(n - 1);
  }
}
```

Detta är okej, vi gör algoritmen **långsammare** (och felaktig)! Vårt $\mathcal{O}$-uttryck kommer vara lite för högt, men det kommer fortfarande stämma.

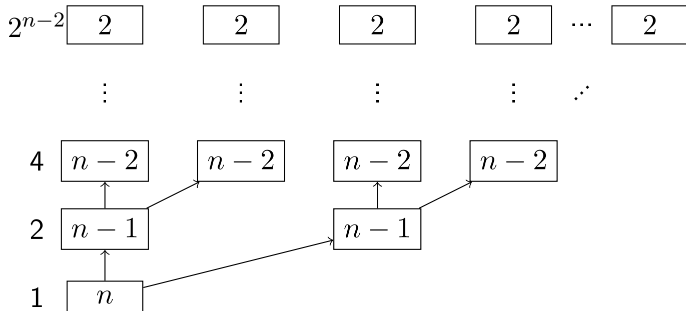
Analys



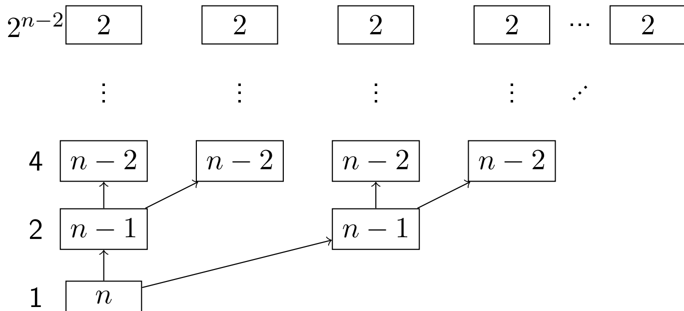
Analys



Analys

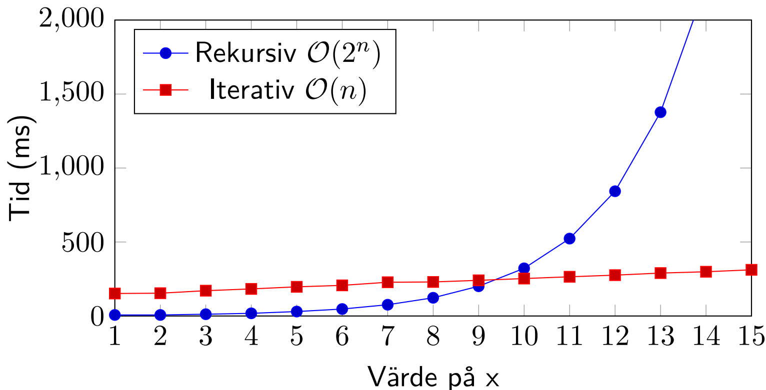


Analys



Totalt: $2^{n-1} - 1 \in \mathcal{O}(2^n)$ anrop

Åter till mätdatan – Stämmer beräkningarna?



1 Kursinformation
2 Varför DALG?
3 Algoritmer
4 Hur körs ett program – Modell
5 Ordonotation - Tillväxthastighet
6 Beräkna tidskomplexitet
7 Nästa gång

LINKÖPING UNIVERSITY

Nästa gång

Nu har vi grunderna för att analysera linjära strukturer!

- Abstrakta datatyper (ADT)
- Array, lista
- Analys av dessa (mer övning på tidskomplexitet, samt nya koncept)

Extrauppgifter på Kattis

- `erase` (enkel)
  Introduktion till Kattis
- `howmanyzeros` (svårare)
  Tänk på tidskomplexiteten på er lösning. Räkna med att ni kan exekevera 1 000 000 000 000 operationer. Vilken tidskomplexitet måste er lösning då ha?

Fördjupningsfrågor

- Hur mycket minne använder båda algoritmerna för fibonacci uttryckt i $\mathcal{O}$-notation?
- Kan vi förbättra den iterativa lösningen på något sätt?
- Kan vi beräkna `fibonacci(n)` på kortare tid än $\mathcal{O}(n)$?

Filip Strömbäck

www.liu.se

LINKÖPING
UNIVERSITY