Welkommen til INF110, *Algoritmer & datastrukturer, det sannelig mest gøyale faget i hele universet!*

REFERANSE:
Disse foiler er basert på arbeid og foiler av
Almira Karabeg, Dag Belsnes, Arne Maus og Ragnar Normann fra tidligere år.

- Introduksjon til kurset og litt praktisk informasjon
- Introduksjon til faget og kort gjennomgang av matematiske forutsetninger

- Introduksjon til rekursjon
- Introduksjon til søking. Kombinatoriske søk

- Hvor effektiv er en algoritme? Kan alle problemer løses effektivt?

- Abstrakte datatyper (ADT): Skille definisjon & implementasjon
- Viktige datastrukturer:
  - Stabler (stacks), køer, lister hash-tabeller
  - Treer
  - Grafer
  - Prioritetskøer (heaps)
  - Kataloger og andre datastrukturer
- Viktige algoritmer knyttet til datastrukturene

- Sortering

INF110 – *Praktisk informasjon #1*

- Obligatorisk oppmøte: Du **må** krysse av for oppmøte i dag!
- Etteranmelding kan være mulig etter 1. september. Hør med administrasjonen.

- Obligatorisk pensum er:
  - Java-versjon av 'the MAW-book' fra Addison-Wesley:
    Data Structures & Algorithm Analysis av Mark Allen Weiss (or MAW)
  - NB! Kapitler 11 & 12 er IKKE pensum, og det er flere seksjoner som IKKE er pensum i de første 10 kapitlene. Se detaljene på Web'en:
    http://www.uio.no/studier/emner/matnat/ifi/INF110/h03/pensumliste.xml
  - Forelesninger og forelesningsnotater (foiler) er pensum!

INF110 – *Praktisk informasjon #2*

- Foiler er markert med uke og forelesning foilene hører til (nederst til venstre).
  Også på Web'en.

  W1.L1 betyr "week 1, lecture 1",
  og det er 2 forelesningsøkter á 2 timer hver (4 timer/uke totalt)

- Det er 14 ukers moro fra mandag 18. august 2003 (og vennligst ta moroa på alvor)
  - 4 obligatoriske oppgaver (OOPs)
  - Siste 2 uker er for gjennomgang av gamle eksamensoppgaver, oppsummering og eventuelle spørsmål
  - Uke 5 (dvs. ved OOP #1 innlevering) er "The Queens Competition Day"
- Moroa tar slutt etter siste forelesning på fredag 21. November 2003.

- Ja. Java er nødvendig. Snakk med gruppelæreren hvis du trenger hjelp

- All info/nyheter legges ut på Web'en:
  http://www.uio.no/studier/emner/matnat/ifi/INF110/h03/

- NB! Vi er ~350 studenter, 1 foreleser og noen gruppelærere

INF110 – *Essensen i faget*

- Lære å lage effektive og velstrukturerte programsystemer og biblioteker, løse vanskelige problemer mer effektivt... **Lære bedre programmering!**

- "Programmering er en kunst" (Ole Johan Dahl)
- "Programmering er forståelse" (Kristen Nygaard)

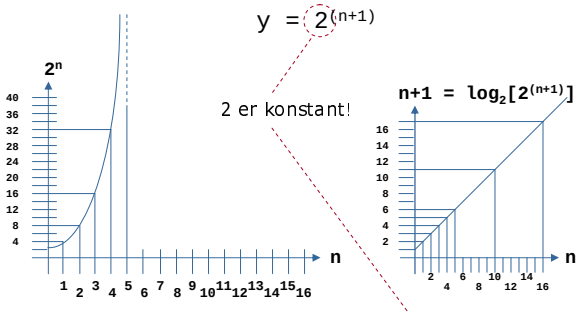
- Eller kanskje:
  Programmering er en kunst som krever forståelse og kjennskap til teknikker!

- I dette faget er programeffektivitet ofte tilknyttet TID. Med andre ord, 'et effektivt program' i dette faget vil ofte bety at 'programmet kjører raskest mulig'.
- Altså skal vi lære teknikker fro å kunne mestre kunsten å lage effektive programmer!

- Eksempel problem: Hvor lang tid tar det å sortere 1 million tall?
- Tre alternative løsninger:
  - **16:10:00.00** (16 timer, 10 minutter og 0,0 sekunder) med optimalisert boble-sortering
  - **00:00:01.40** (1,4 sekunder) med quick-sort
  - **00:00:00.80** (0,8 sekunder) med radix-sort

MATTE – *Logaritmer #1*

- Imagine...
  at du trenger å tegne en graf av
  $y = 2^{(n+1)}$
  for $n = 0, 1, 2, 3, \ldots$



a) Hvis du tegner y mot n!

b) Du kan følge eksponenten!

n	$2^{(n+1)}$
0	2
1	4
2	8
3	16
4	32
5	64
6	128
7	256
8	512
9	1 024
10	2 048
11	4 096
12	8 192
13	16 384
14	32 768
15	65 536
16	131 072

- For $y = b^x$
  vil $\log_b y = \log_b b^x = x$
- For et vilkårlig tall a
  $\log_b a$ betyr at vi er på jakt etter tallet x der $a = b^x$ der
- Talle 'b' er logaritmens basis-tall,
  (som egentlig er basis tall for uttrykket b^x)
- Eksempel:
  - $\log_3 9$ er 2, siden 9 er 3^2
  - $\log_2(3,249009585)$ er $1,7$ siden siden $3,249009585$ er $2^{(1,7)}$
- Mer formelt:
  Logaritmen med basis b til et tall y er det tallet x (eksponenten) vi må opphøye basistallet b i for å få y.
- De vanligste basistall er 2, e (ref. ln, dvs. 'naturlig logaritme') og 10
- VI BRUKER ALLTID BASIS 2 i algoritme-analyse.
  - Dermed skal vi droppe basis-tallet i INF100 og alltid skrive $\log a$ siden vi vet 'by convention' at basistallet er alltid 2 for oss.

Regneregler for algoritmer:

$$\log_b xy = \log_b x + \log_b y$$

$$\log_b x/y = \log_b x - \log_b y$$

$$\log_b x^a = a \cdot \log_b x$$

$$\log_b x = \frac{\log_a x}{\log_a b}$$

Regneregler for eksponenter:

$$a^{(b+c)} = a^b a^c$$

$$a^{bc} = (a^b)^c$$

$$\frac{a^b}{a^c} = a^{(b-c)}$$

$$b = a^{\log_a b}$$

$$b^c = a^{c \times \log_a b}$$

Summen av en vilkårlig rekke :

$$\sum_{i=l}^{k} a_i = a_l + a_{l+1} + ... + a_k$$

En kjent sum:

$$\sum_{i=0}^{n} 2^i = 1 + 2 + 4 + 8 + ... + 2^n \; = \; 2^{n+1} - 1$$

Enda en kjent sum:

$$\sum_{i=1}^{n} i = 1+2+3+4+\ldots+n = \frac{n(n+1)}{2}$$

Bevis:

Kall summen S:

$1+\quad 2\quad +\ldots..+n=S$

$n+(n-1)+\ldots...+1=S$

Summer disse to:

$(n+1)+(n+1)+....+(n+1)=2S$

$$S=\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

En tredje kjent sum – geometrisk rekke:

$$\sum_{i=0}^{n} a^i = 1+a+a^2+\ldots+a^n = \frac{1-a^{n+1}}{1-a}$$

Bevis:

$S=1+a+a^2+\ldots+a^n$ (*) (kaller $\sum_{i=0}^{n} a^i$ for S)

multipliser (*) med a:

$a \cdot S = a+a^2+\ldots+a^n+a^{n+1}$ (**)

(*) - (**):

$S-aS=1-a^{n+1}$

$S(1-a)=1-a^{n+1}$

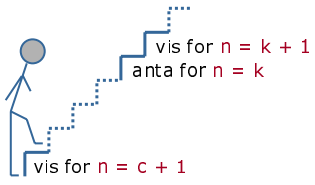
BEVIS – *Tre måter*

- Hvordan bevise/motbevise et matematisk utsagn (teorem)?
- Tre måter:
  - Induksjonsbevis
  - Motbevise ved hjelp av moteksempel
  - Bevis ved hjelp av selvmotsigelse

BEVIS – *Induksjonsbevis #1*

- Gitt en 'matematisk' sats vi skal vise for alle heltall n > c (c konstant, typiske verdier : 0,1,2,3,..)
- Viser først satsen for minste verdi (c + 1)
- Antar så at satsen er riktig for n=k, (k et vilkårlig heltall > c)
- Bruker så antagelsen i (3) for å vise at satsen også er riktig for n=k+1.
- Da har vi vist satsen generelt!

Stige sammenligning: Hvis du kan vise at du kan gå opp på nederste trinn, og at du kan gå et trinn til fra et vilkårlig trinn, har du i prinsipp vist at du kan gå så høyt du vil!



Induksjonseksempel: Vi skal viset at $2^n > n^2 \quad \forall n > 4$

- BASIS: Vi viser at det er riktig for $n = 5$, dvs. at $2^5 > 5^2$
- ANTAGELSE: Vi antar at det holder for k,
  dvs. at $2^k > k^2$ for vilkårlig $k > 4$
- INDUKSJONSSKRITT:
  Vi skal nå vise at det holder også for $n = k + 1$,
  dvs. at $2^{k+1} > (k+1)^2$.
  ... Venstre side: $2^{k+1} = 2^k.2 > k^2.2$ siden vi antar at $2^k > k^2$.
  ... Det er det samme som $k^2 + k^2$. Altså er $2^{k+1} > k^2 + k^2$.
  ... $2^{k+1} > k^2 + k^2 >$ Høyre side $= (k+1)^2 = k^2 + 2k + 1$
  ... $k^2 + k^2 > k^2 + 2k + 1 \Rightarrow k^2 > 2k + 1 \quad \forall k > 4$
  ... at $k^2 > 2k + 1 \quad \forall k > 4$ kan vises ved hjelp av induksjon
  ... (gjør det som oppgave)

- Bevist!

Eksempel på motbevis med moteksempel:
Vi skal vise at $k^2 < 2k + 1$ ikke holder $\forall k > 4$

- PÅSTAND: $k^2 < 2k + 1, \forall k > 0$
- Kun ett moteksempel er nok!

  ... La $k = 3$.
  ... $k^2 = 9$, $2k + 1 = 2 \cdot 3 + 1 = 7$.
  ... Altså er $k^2 > 2k + 1$. Motsigelse.

- Motbevist ved hjelp av moteksempel!

Eksempel på bevis ved hjelp av selvmotsigelse:
Vi skal vise at det finnes et uendelig antall primtall (Euclid).
Primtal er tall ≥ 2, som bare lar seg dele med 1 og seg selv.

- Anta det motsatte, dvs. at det finnes et største primtall. La det største primtallet hete P.
- Vi lager et nytt tall S som følger:
  ... La $S = 2 \times 3 \times 5 \times 7 \times \ldots \times P + 1$,
  ... dvs. alle primtall ganget + 1.
- Men S er et primtall, siden ingen av primtallene går opp i P (det blir 1 til overs), og P er i tillegg større enn P!
  Dette er en selvmotsigelse i forhold til antagelsen!
- Motbevist ved hjelp av selvmotsigelse (contradiction).

- DEFINISJON:
  Pseudo-kode er kode som beskriver en algoritme uten unødige detaljer (klasser, deklarasjoner,...). Blanding av 'kode' og naturlig språk .
- Brukes mye i litteratur. Brukes en del i INF110.
- Eksempel algoritme med pseudo-kode:

```
arrayMax(A,n);
    Input:  An array A with n integers
    Output: The max element in A
        currentMax = A[0];
        for i = 1 to  n-1 do
            if currentMax < A[i] then
                currentMax = A[i];
        return currentMax;
```

TERMINOLOGI – *Tid i kode*

- Hvor lang tid bruker en enkel for-løkke?

```
for (int i = 0; i < n; i++)
    a[i] = a[n-i-1];
```

- Hvor lang tid bruker en dobbel for-løkke?

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        a[i] = a[n-j-1];
```

(Tid i millisekunder)

n	Enkel-løkke	Dobbel-løkke
10	1	1
100	1	1
1 000	1	56
10 000	2	5 856
100 000	13	640 110
1 000 000	134	?

TERMINOLOGI – *Tidsmålinger*

- For å kunne måle effektiviteten av en algoritme, trenger vi å kunne måle tiden. Flere måter:
- Hvordan skal vi ta tiden?

```
long tid = System.currentTimeMillis();
  bruk(n);
  tid = System.currentTimeMillis() - tid;
  System.out.println("Tid brukt: " + tid + "millisekunder.");
```

- Hvordan skal vi beregne/estimere tiden?
  Teoretiske beregninger og estimeringsteknikker kommer neste gang!

  Dette avslutter W1.L1!