Uke 2,
Forelesning 2

- Essensen i faget: Effektive algoritmer
- Matematiske forutsetninger
  - Logaritmer, regneregler for logaritmer
  - Eksponenter, regneregler for eksponenter
  - Rekker og summer
  - Bevis
    - Induksjonsbevis
    - Motbevis ved hjelp av moteksempel
    - Bevis ved hjelp av selvmotsigelse
- Begreper/teknikker: Pseudo-kode; Måle reel tid eller beregne/estimere (teoretisk) tid

Uke 1.1

- Introduksjon til metodekall og rekursjon
- Introduksjon til kombinatoriske søk, kombinasjoner og permutasjoner
- Introduksjon til dronning oppgaven

Uke 1.2

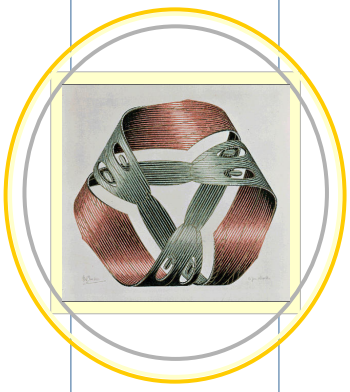
- Hvordan en kan GENERERE PERMUTASJONER
- Introduksjon til ANALYSE av ALGORITMER

Uke 2.1

OVERSIKT – *Uke 2, Forelesning 2 (W2.L2)*

- AVRUNDE
  - Matematiske forkunnskaper
  - Rekursjon
  - Kombinasjoner og permutasjoner
- Litt mer om DRONNING OPPGAVEN

TEMA #1



Avrunde
Uke 1 og 2

LOGARITMER – *En gjentakelse*

- Vi skal snakke kun om $\log_2$ (dvs. basis-2 logaritmer)
- Vi har sett basisregelen, dvs. at $\log 2^x = x$

- Vi har sett bl.a. følgende regler:
  - $\log x \cdot y = \log x + \log y$
  - $\log x / y = \log x - \log y$
  - $\log x^a = a \cdot \log x$

- Vi kan øve litt ved å prøve å bevise reglene ut i fra basisregelen

LOGARITMER – *En øvelse*

- Husk basisregelen, dvs. at $\log 2^x = x$
- Vi skal prøve å vise at
  $\log x \cdot y = \log x + \log y$

- Vi kan alltid skrive x og y slik at
  $x = 2^a$ og $y = 2^b$

- Da er venstre side (vs), dvs. $\log x \cdot y$, det samme som
  $= \log (2^a \cdot 2^b)$
  $= \log 2^{a+b}$ .....fordi $(2^a \cdot 2^b) = 2^{a+b}$
  $= a + b$
  .....siden $\log 2^{a+b} = \log 2^x$ der $x = a + b$,
  .....og ved bruk av basisregel er $\log 2^x = x = a + b$
- Høyre side (hs), dvs. $\log x + \log y$, kan skrives som
  $= \log 2^a + \log 2^b$
  $= a + b$ .....fordi $\log 2^a = a$ og $\log 2^b = b$ ved bruk av basisregel.
- **Q.E.D.** .....Quod Erat Demonstrandum (som skulle demonstreres)

LOGARITMER – *Enda en øvelse*

- Husk igjen basisregelen, dvs. at $\log 2^x = x$
- Vi skal prøve å vise at
  $\log x / y = \log x - \log y$
- Igjen kan vi skrive x og y slik at
  $x = 2^a$ og $y = 2^b$
- Da er venstre side (vs), dvs. $\log x / y$, det samme som
  $= \log (2^a / 2^b)$
  $= \log 2^{a-b}$ .....fordi $(2^a / 2^b) = (2^a \cdot 2^{-b}) = 2^{a-b}$
  $= a - b$
  .....siden $\log 2^{a-b} = \log 2^x$ der $x = a - b$,
  .....og ved bruk av basisregel er $\log 2^x = x = a - b$
- Høyre side (hs), dvs. $\log x - \log y$, kan skrives som
  $= \log 2^a - \log 2^b$
  $= a - b$ .....fordi $\log 2^a = a$ og $\log 2^b = b$ ved bruk av basisregel.
- **Q.E.D.**

LOGARITMER – *En tredje øvelse*

- Husk igjen basisregelen, dvs. at $\log 2^x = x$
- Vi skal prøve å vise at
  $\log x^a = a \cdot \log x$
- Vi kan skrive x slik at
  $x = 2^k$
- Da er venstre side (vs), dvs. $\log x^a$, det samme som
  $= \log (2^k)^a$
  $= \log 2^{k.a}$ .....fordi $(2^k)^a = 2^{k.a}$
  $= k \cdot a$
  .....siden $\log 2^{a.k} = \log 2^x$ der $x = k \cdot a$,
  .....og ved bruk av basisregel er $\log 2^x = x = k \cdot a$
- Høyre side (hs), dvs. $a. \log x$, kan skrives som
  $= a \cdot \log 2^k$
  $= a \cdot k$ .....fordi $\log 2^k = k$ ved bruk av basisregel.
- **Q.E.D.**

LOGARITMER – *Andre øvelsen andre gang*

- Husk igjen basisregelen, dvs. at $\log 2^x = x$
- Vi skal prøve, for andre gang, å vise at
  $\log x / y = \log x - \log y$

- Merk at vi kan skrive $x / y = x \cdot y^{-1}$
- Det betyr at vi skal se på venstre side
  $= \log x \cdot y^{-1}$

- Så kan vi enkelt bruke første bevis,
  dvs. at $\log x \cdot (y^{-1}) = \log x + \log (y^{-1})$
- Og så bruker vi siste bevis, dvs. at $\log x^a = a \cdot \log x$, som betyr at
  $= \log (y^{-1}) = -1 \cdot \log y = - \log y$, som igjen betyr at
  $= \log x + \log (y^{-1}) = \log x - \log y$
- Altså har vi avledet (Engelsk: "derived") høyre side fra venstre side direkte!

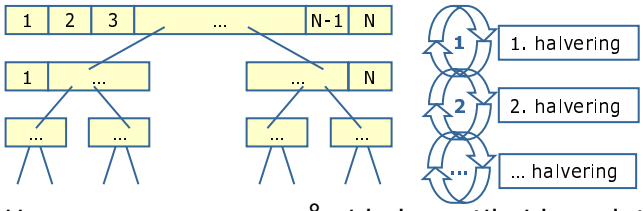
- **Q.E.D.**

LOGARITMER – *Et brukseksempel*

- Gitt at jeg har N elementer i en liste eller array:

1	2	3	…	N-1	N

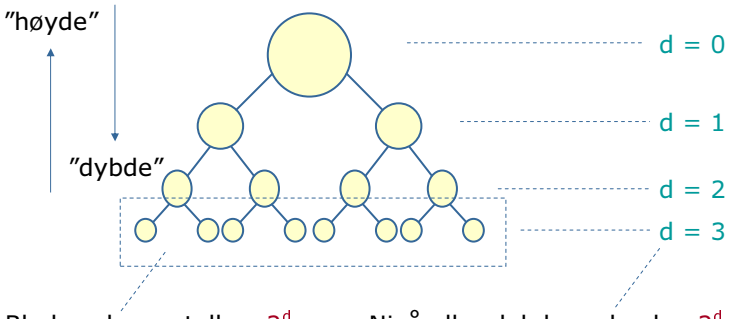
- Vi har en løkke (eller rekursivt kall) som splitter listen opp i to ved hver gjennomgang:



- Hvor mange ganger må vi halvere til vi kan slutte ved en liste eller array med kun 1 element igjen, som betyr at det ikke lenger kan halveres?
- SVAR: log N

LOGARITMER og TRÆR – *Hvorfor log N?*

- Et binært tre er en hierarki der en node deles opp i to noder:



Blad-noder, antall = 2^d

Nivå eller dybde = d = log 2^d

Antall noder = $\sum_{i=0}^{d} 2^i$

SUM av SEKVENSER – *Noen eksempler #2*

- En polynom, som for eksempel $4x^3 + 2x + 1$, kan representeres i sin generelle form ved hjelp av en sum:

$$\sum_{i=0}^{n} a_i x^i = a_0x^0 + a_1x^1 + a_2x^2 + \ldots + a_nx^n$$

"koeffisientene"
dvs.
multiplikasjonsfaktorene

- Null koeffisient betyr at termen ikke eksisterer. Eksempel:
  For $4x^3 + 2x + 1$
  er koeffisientene $a_0 = 1$, $a_1 = 2$, $a_2 = 0$ og $a_3 = 4$

SUM av SEKVENSER – *Husk kjente summer*

- Noen summer kjenner vi fra før, og vi har formulert de:

$$\sum_{i=0}^{n} 2^i = 1+2+4+8+\ldots+2^n = 2^{n+1}-1 \qquad \text{Sum formel \#1}$$

$$\sum_{i=1}^{n} i = 1+2+3+4+\ldots+n = \frac{n(n+1)}{2} \qquad \text{Sum formel \#2}$$

$$\sum_{i=0}^{n} a^i = 1+a+a^2+\ldots+a^n = \frac{1-a^{n+1}}{1-a} \qquad \text{Sum formel \#3}$$

- Husk at "sum-av-sekvenser" antyder løkker, som krever N gjennomganger for N elementer, dvs. tar O(N) tid…
- Men beregning av formlene tar O(1) tid (konstant tid). BRUK FORMLER NÅR DU KAN!

SUM av SEKVENSER – *Bruk kjente summer!*

- Eksempel:
  Skriv et program som beregner følgende sekvens:

  $S = 2 + 2^2 + 2^3 + 2^4 + \ldots + 2^n$

- HUSK FØRST:

$$\sum_{i=0}^{m} 2^i = 1+2+4+8+\ldots+2^m = 2^{m+1}-1 \qquad \text{Sum formel \#1}$$

- Merk at $S = 2 \cdot (1 + 2 + 2^2 + 2^3 + \ldots + 2^{n-1})$ og la $m = n-1$
- Altså er $S = 2 \cdot (2^{m+1} - 1) = 2 \cdot (2^{n-1+1} - 1)$
  $= 2 \cdot (2^n - 1)$

- Nå kan du programmere det i O(1) tid!

- HUSK:
  Rekursjon er at en funksjon kaller seg selv, gjerne med en "ny" verdi som helst er nærmere en "slutt" verdi.
- Rekursjon er ikke bare et programmeringsbegrep.
  Det er faktisk først og fremst et matematisk begrep!

- Her er en veldig enkel rekursiv funksjon (ligning):
  $f_n = f_{n-1} + 1$, der $n > 0$, og $f_0 = 1$
  Regn ut (formuler) hva f_n er UTEN REKURSJON!

- Merk at $f_n = f_{n-1} + 1 = (f_{n-2} + 1) + 1 = ((f_{n-3} + 1) + 1) + 1$
  og helt til … $= (((f_0 + 1) + 1) \ldots) + 1$
  $= (((1 + 1) + 1) \ldots) + 1$
- f_n er altså = 0 til n-1 stykker 1'ere (n av dem) + 1 , dvs.:
  $f_n = n+1$

- Et annet eksempel:
  $f_n = 2.f_{n-1} + 1$, der $n > 0$, og $f_0 = 1$
  Regn ut (formuler) hva f_n er UTEN REKURSJON!
- NB! Hanoi løses med tilsvarende rekursiv ligning…
- $f_n = 2. f_{n-1} + 1$
  $= 2.(2.f_{n-2} + 1) + 1$
  $= 2.(2.(2.f_{n-3} + 1) +1) + 1$
  $= 2.(2^2.f_{n-3} + 2) +1) + 1$
  $= 2^3.f_{n-3} + 2^2 +2 + 1$
- Se mønsteret: $f_n = 2^k f_{n-k} + 2^{k-1} + 2^{k-2} + \ldots + 2^2 + 2 + 1$
- Når k = n vil f_n være
  $= 2^n f_0 + 2^{n-1} + 2^{n-2} + \ldots + 2^2 + 2 + 1$, dvs.
  $= 2^n + 2^{n-1} + 2^{n-2} + \ldots + 2^2 + 2 + 1$, siden $f_0 = 1$.
- Men dette er jo en kjent sekvens sum (formel #1):

$$\sum_{i=0}^{n} 2^i = 1 + 2 + 4 + 8 + \ldots + 2^n = \boxed{2^{n+1} + 1}$$

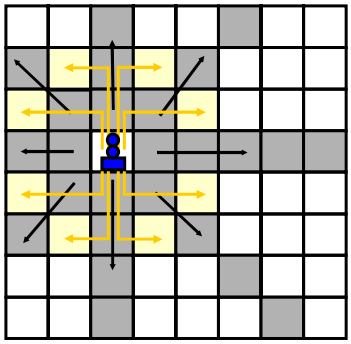
HUSK: OBLIG 1 – *Introduksjon*

- En Dronning i sjakk kan slå alle brikker som står på…
  samme rad,
  samme kolonne,
  eller samme diagonaler (dvs. rette skrålinjer)…

- MEN merk symmetriene, reglene:

  Merk at ”safe” områder er en ”springer” avstand fra dronningen!

  Kan du bruke denne observasjonen?

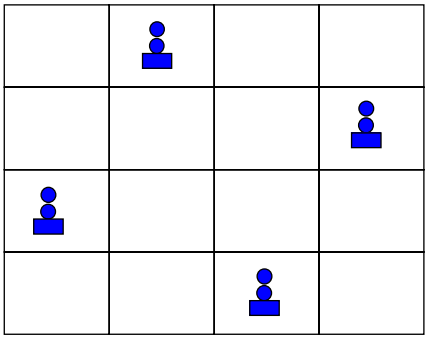


OBLIG 1 – *Selve oppgaven (gjentatt)*

- Bruk permutasjoner til å skrive ut alle interessante løsninger av ’n’ dronninger på et N x N sjakkbrett hvor ingen dronning kan slå en annen.
- Ser at N=1 har 1 løsning, mens N = 2 og 3 ikke har løsning. Hva med N=4 ?
  Løs oppgaven generelt, og test med N=8.
- Hvorfor permutasjoner:
  - La p[i] si hvilken kolonnenummer. Vi vil plassere en dronning i rad nr ’i’. Da slipper vi å generere mange håpløse kombinasjoner…
- Vi er ikke interessert i ’nye’ løsninger som kan lages fra en allerede presentert løsning ved:
  - å vippe sjakkbrettet rundt om en midtlinje eller diagonal
  - rotere brettet 90, 180 eller 270 grader

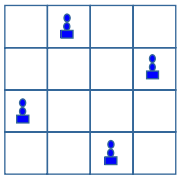
OBLIG 1 – *En løsning for $N = 4$*

- Her er en løsning for N = 4:

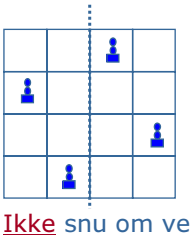


OBLIG 1 – *Krav til løsningen #1*

- Husk å fjerne de løsninger som bare er en snuing av brettet om horisontal/vertikal midtlinje eller rotering av brettet som i eksempelet nedenfor for N = 4…



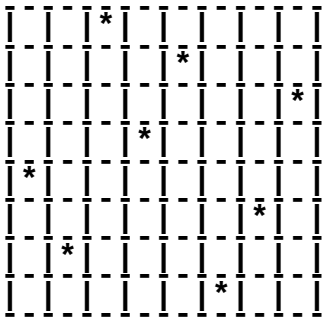
Utgangsløsning



Ikke snu om vertikal midtlinje!

OBLIG 1 – *Krav til løsningen #2*

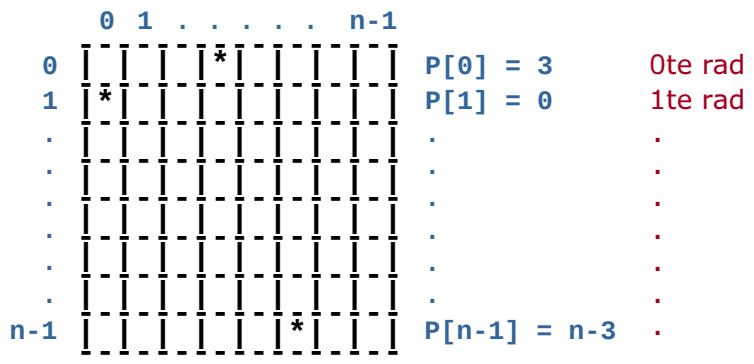
- Tell opp antall ”nye” løsninger
- La programmet skrive ut brettet med dronningene for hver løsning på en enkel måte:



OBLIG 1 – *Tips'n tricks*

ALGORITMESKISSE

- Lag en permutasjon som sørger for at alle dronningene står på ulike rader **og** kolonner – som nedenfor



ALGORITMESKISSE (fortsetter)

- Lag en permutasjon som sørger for at alle dronningene står på ulike rader **og** kolonner – som ovenfor
- Test om den er en løsning – dvs. om ingen av dronningene står på samme diagonal (opp- og ned-diagonal)
- Test om dette er en ny løsning, eller bare er en snuing av brettet (om diagonaler eller midtlinjer) eller rotering av brettet (90º, 180º eller 270º).

  Tips her: Hvis man etter å ha snudd eller rotert brettet ender opp med en permutasjon som er mindre (sett som et n-sifret tall), **har vi laget den før** og skrevet den ut - hopp over. Hvis alle slike snuinger/rotasjoner er større (eller lik) er den ny !

- Hva gjør **brukPerm() ?**
- Hvordan teste om det står noe på en diagonal ?
- Hvordan skrive ut en permutasjon ?
- Hvordan foreta en snuing og rotasjon og så sammenligne med den du har i p[ ] ?

- Og så tilsist (ikke krav):

  Hvordan få en virkende program til å gå minst 100x fortere for n > 10

- Det er oppgaven - lykke til !

OBLIG 1 – *Konkurransen…*

- **NB!** Konkurranse…
- **Tre raskeste riktige og velstrukturerte** løsninger for N = 10, som bare skriver ut sjakkbrettet pr. løsning + totale tiden (for alle)!
  - Alle tre får ”anerkjennelse” … skriftlig!
  - Første får i tillegg … noe ganske godt ☺
  - Nummer to og tre får i tillegg en pose ”Twist” hver…
  - Uke 5, mandag 15. september 2003, skal de tre presentere (forklare) sine løsninger og kjøre de på auditoriet…