Uke 5,

Forelesning 1

- Essensen i faget: Effektive algoritmer
- Matematiske forutsetninger
- Introduksjon til metodekall og rekursjon
- Introduksjon til kombinatoriske søk, kombinasjoner og permutasjoner
- Introduksjon til analyse av algoritmer

Uke 1 og Uke 2

- Abstrakte datatyper (ADT'er)
- Lister, stabler og køer

Uke 3 og Uke 4.1

- Generelle trær
- Binære trær
- Introduksjon til binære søketrær

Uke 4.2

OVERSIKT – *Uke 5, Forelesning 1 (W5.L1)*

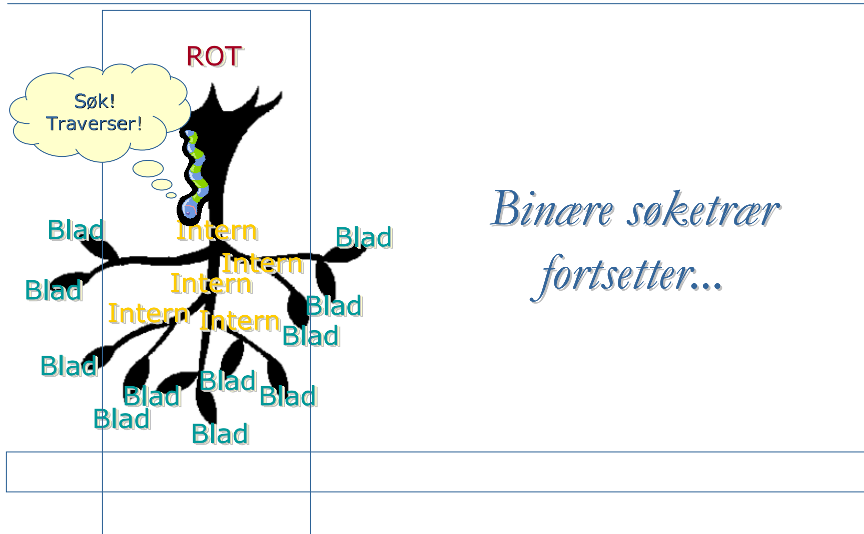
- Vi fortsetter med tema #3:
  BINÆRE SØKETRÆR (MAW kapittel 4.3)
  - Og om ca. 2 UKER (uke 6 forelesning 2 eller W6.L2):
    TEMA #4: B-trær (MAW kapittel 4.7)

- Vi skal "snakke om" balanserte binære trær også (AVL-trær, MAW kapittel 4.4).

  Det er for å gjøre forståelsen av trær mer komplett,
  men DET ER IKKE PENSUM!

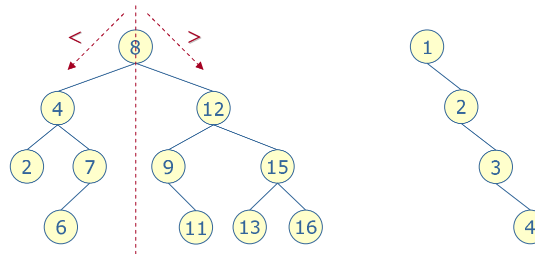
- OBS! Har du husket å levere din Oblig # 1 til din kjære gruppelærer?

TEMA #3: BINÆRE SØKETRÆR



BINÆRE SØKETRÆR – *Introduksjon (fra forrige gang)...*

- Binære søketrær er variant av binære trær hvor følgende holder for hver node i treet:
  - Alle verdiene i nodens **venstre** subtre er **mindre** enn verdien i noden selv.
  - Alle verdiene i nodens **høyre** subtre er **større** enn verdien i noden selv.



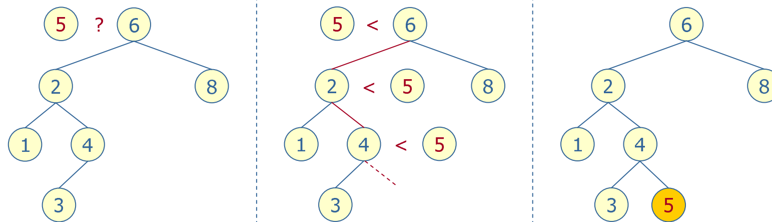
BINÆRE SØKETRÆR – *Søk...*

```
// Rekursiv metode for søk:
public BinNode finn(int x, BinNode t)
{  if (t == null)
   {  return null;
   }
   else if (x < t.tall)
   {  return finn(x, t.venstre);
   }
   else if (x > t.tall)
   {  return finn(x, t.hoyre);
   }
   else
   {  return t;
   }
}
```

```
// Ikke-rekursiv metode søk:
public BinNode finn(int x, BinNode t)
{  BinNode n = t;
   while (n != null && n.tall != x)
   {  if (x < n.tall)
      {  n = n.venstre;
      }
      else
      {  n = n.hoyre;
      }
   }
   return n;
}
```

- Ideen er enkel:
  - Gå nedover i treet på samme måte som ved søking.
  - Hvis tallet finnes i treet allerede, gjør ingenting.
  - Hvis du kommer til en null-peker uten å ha funnet tallet: sett inn en ny node (med tallet) på dette stedet.

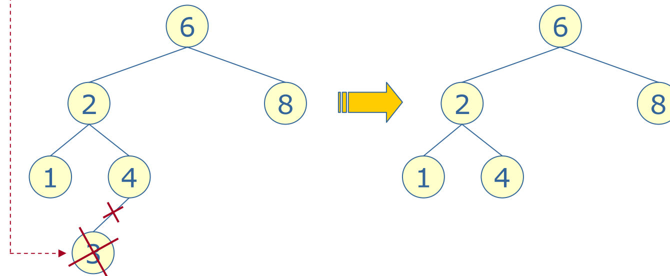
- Eksempel: Sett inn 5 ...



- Sletting er vanskeligere:
  Etter å ha funnet noden som skal fjernes har vi **tre** mulige situasjoner.
  - Noden er en bladnode:
    - Kan fjernes direkte.
  - Noden har bare ett barn:
    - Foreldrenoden kan enkelt hoppe over den som skal fjernes.
  - Noden har to barn:
    - Erstatt verdien i noden med den minste verdien i **høyre** subtreet.
    - Slett noden som denne minste verdien var i.

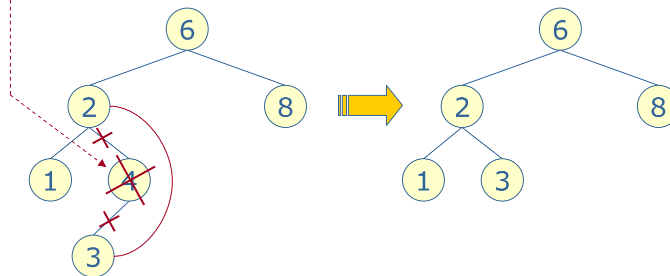
BINÆRE SØKETRÆR – *Slett en bladnode...*

- Noden er en bladnode (for eksempel node 3):
  - Kan fjernes direkte.



BINÆRE SØKETRÆR – *Slett en node med ett barn...*

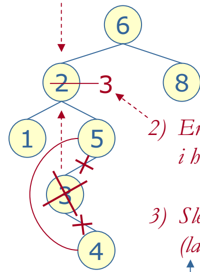
- Noden har bare ett barn (node 4):
  - Foreldrenoden kan enkelt hoppe over den som skal fjernes.



BINÆRE SØKETRÆR – *Slett en node med to barn...*

- Noden har to barn (node 2):
  - Erstatt verdien i noden med den minste verdien i **høyre** subtre.
  - Slett noden som denne minste verdien var i.

1) Traverser og finn noden



2) Erstatt med minste verdien i høyre subtreeet

3) Slett noden (la foreldrenode peke til det ene barnet)

6
3 8
1 5
4

NB! Kun fordi dette er slett med ett barn! Alternativt: Bare fjern (hvis bladnode), eller anvend slett med to barn "rekursivt"

BINÆRE SØKETRÆR – *Rekursiv sletteprogram...*

- Rekursiv metode for å fjerne et tall:

```
public BinNode fjern(int x, BinNode t)
{  if (t == null) return null;          // Fant ikke x, skal ikke gjøre noe
   if (x < t.tall) t.venstre = fjern(x, t.venstre);
   else if (x > t.tall) t.hoyre = fjern(x, t.hoyre);
   else if (t.venstre != null && t.hoyre != null)
   {  t.tall = finnMinste(t.hoyre).tall;
      t.hoyre = fjern(t.tall, t.hoyre);
   }
   else
   {  if (t.venstre != null) t = t.venstre;
      else t = t.hoyre;
   }
   return t;
}
```

- NB! Denne metoden er lite effektiv, siden den leter etter minste tall i høyre subtre to ganger. En lokal variabel kan rette opp det. (Hvordan?)

BINÆRE SØKETRÆR – *Analyse av gjennomsnittlig tidsforbruk...*

- Intuitivt vil vi forvente at alle operasjonene vi utfører på et binært søketre vil ta *O*(*log n*) tid siden vi hele tiden grovt sett halverer størrelsen på treet vi jobber med.
- Det kan bevises at den **gjennomsnittlige dybden** til nodene i treet er *O*(*log n*) når alle innsettingsrekkefølger er like sannsynlige.
- Se MAW kapittel 4.3.5 (detaljene er ikke pensum).
- Se også oppgave 5 til neste uke.

BINÆRE SØKETRÆR – *Vise gjennomsnittlig søketid = O(log n)?*

- Vi skal ikke bevise men vi kan vise at **dybden** i et tre = *O*(*log n*).
- Vi kan i tillegg vise at

  dybden = lengden fra roten til bladnodene
  (hvis søket "i verste fall" ender i en bladnode)
  eller

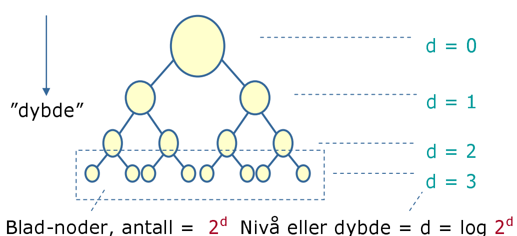
  dybden = lengden fra roten til noden vi trenger
  (hvis søket "i gjennomsnitt" ender i en internnode)

  og er tilsvarende antall "steg", "gjennomganger" eller "rekursive kall" som trenges for å finne en verdi i bladnodene
- Altså er dybden = kjøretid = *O(log n)*.

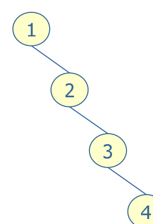
BINÆRE SØKETRÆR – *Analyse av gjennomsnittlig tidsforbruk...*

- Gjennomsnitt = verstefall
  hvis binæere treet er "balansert",
  dvs. hvis hver node har (idéelt sett) to barn.

HUSK (balansert):



IKKE balansert:



BINÆRE SØKETRÆR – *O(log n) er bra!*

- Kjøretid = *O(log n)* er ikke beste men meget bra:
  O(log n) er bedre enn *O(n)*. HUSK:

	f(n)	Navn
1. beste	1	Konstant
2. beste	log n	Logaritmisk
3. beste	n	Lineær
Bra	n log n	?
Levelig	n^2	Kvadratisk
Levelig	n^3	Kubisk
Verstingene	2^n , n!	Eksponensiell

Polynomisk tid (n^2, n^3)

Raskere voksende

n! = 1 x 2 x 3 x ... x n
Vokser meget raskt!

BINÆRE SØKETRÆR – *Et større eksempel #1*

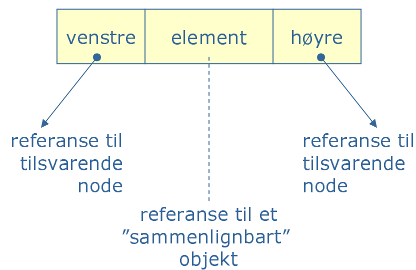
- Problem: Vi ønsker å analysere Henrik Ibsens skuespill, Vildanden.
  - Hvilke ord har Ibsen brukt?
  - Hvor mange forskjellige ord er brukt?
  - Hvilke ord er mest brukt?
- **Implementasjon**
  - Hvilke ord har Ibsen brukt?
    - Ide: Leser inn ordene fra fil og lagrer dem i et binært søketre. Like ord telles opp.
  - Hvilke ord er mest brukt?
    - Ide: Lager et nytt søketre sortert på antall ganger brukt!

BINÆRE SØKETRÆR – *Et større eksempel #2*

```
import inf110.*;                                    // Vi henter inn IbsenTre og litt til her.
public class Ibsen
{   public static void main(String[] args)          // HOVEDPROGRAM
    {   String ord;
        int antallOrd = 0;

        String sep = ". :,;-?!\"";                  // Ordskillere
        Inn innfil = new Inn(args[0]);              // Navnet til innfil er i kommandolinjen
        IbsenTre tre = new IbsenTre();              // IbsenTre er et "søketre"

        innfil.skipSep(sep);                        // Finn ordbegynnelsen etter ordskiller
        while (!innfil.endOfFile())                 // Så lenge det ikke er "end-of-file"...
        {   ord = innfil.inString(sep).toLowerCase(); // Les ord og konverter til småbokstaver
            tre.settInn(new IbsenElem(ord));        // Sett ordet i treet
            antallOrd++;                            // Tell opp ("inkrementer") antallet ord
            innfil.skipSep(sep);                    // Hopp over ordskiller og finn begynnelsen
        }                                           //     av neste ord.

        System.out.println("Totalt antall ord: " + antallOrd);
        System.out.println("Antall ulike ord: " + tre.size());

        IbsenFrekTre frekTre = new IbsenFrekTre();
        frekTre.innsetting(tre.getRot());

        frekTre.skrivInnfiks();
    }
}
```

// Vanlig node i binært søketre: Basis node-element

```
class BinNode
{  Comparable element;

   BinNode venstre;
   BinNode hoyre;

   BinNode(Comparable x)
   {  element = x;
   }
}
```




```
// Vanlig binært søketre der "noe gjøres" ved like elementer ved oppdater(n, x) nederst:
public class BinSokeTre
{   protected BinNode rot;
    protected int antallNoder = 0;

    // Legg inn en ny node med sammenlignbare elementet x i den ved node n
    private void settInn(Comparable x, BinNode n)
    {   int i = sammenlign(x, n.element);   // Sammenlign elementet x med nodens element
        if (i < 0)                          // Hvis det nye elementet er "mindre"...
        {   if (n.venstre == null)          // Sjekk om venstre subtre er tom...
            {   n.venstre = new BinNode(x); // Hvis tom, legg in nye elementet der, og
                antallNoder++;              //     oppdater antallet noder.
            }
            else settInn(x, n.venstre);     // Hvis IKKE tom, legg inn under dens venstre
        }
        else if (i > 0)                     // Hvis det nye elementet er "større"...
        {   if (n.hoyre == null)            // Sjekk om høyre subtre er tom...
            {   n.hoyre = new BinNode(x);   // Hvis tom, legg in nye elementet der, og
                antallNoder++;              //     oppdater antallet noder.
            }
            else settInn(x, n.hoyre);       // Hvis IKKE tom, legg inn under dens høyre
        }
        else oppdater(n, x);                // Hvis det nye elementet ikke er hverken
    }                                       //      "mindre" eller "større", gjør noe (oppdater).
    // Flere metoder her...
}
```

Oppgave: Lag en ikke-rekursiv settInn!

BINÆRE SØKETRÆR – *Et større eksempel #5*

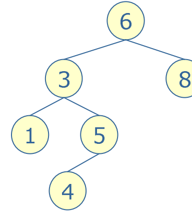
```
public void settInn(Comparable x)    // NB! ≠ settInn(Comparable, BinNode)
{  if (rot == null)                  // Hvis tomt tre...
   {  rot = new BinNode(x);          // Legg new node som rot-noden,
      antallNoder++;                 //      og tell opp antallet noder.
   }
   else settInn(x, rot);             // Hvis IKKE-tomt tre, bruk den andre
}                                    //       settInn(Comparable, BinNode).

protected void oppdater(BinNode n, Comparable x)
{                                    // Default: Ingenting gjøres for like
}                                    //      elementer.

protected int sammenlign(Comparable n1, Comparable n2)
{  return n1.compareTo(n2);          // Bruker metoden "compareTo()" som
}                                    //      er en instans-metode i Comparable
                                     //      klassen (dvs. en metode for
                                     //      klassens objekter (instanser)...
                                     // Returnerer int < 0 for n1 < n2,
                                     //      int > 0 for n1 > n2,
                                     //      og 0 for n1 = n2.
```


BINÆRE SØKETRÆR – *Et større eksempel #6*

```
public void skrivInnfiks()         // Skriver i innfiks rekkefølge.
{  innfiks(rot);                   // Se neste foil!
}

private void innfiks(BinNode n)    // Se neste foil!
{  if (n != null)
   {  innfiks(n.venstre);
      System.out.println(n.element.toString());
      innfiks(n.hoyre);
   }
}

public int size()
{  return antallNoder;
}

public BinNode getRot()
{  return rot;
}
```

BINÆRE SØKETRÆR – *Et større eksempel #7*

```
public void skrivInnfiks()
{   innfiks(rot);
}

private void innfiks(BinNode n)
{   if (n != null)
    {   innfiks(n.venstre);
        System.out.println(n.element.toString());
        innfiks(n.hoyre);
    }
}
```



HÅNDKJØRING (les tallene som noder):

01) innfiks(rot: node-6)
02) node-6 != null; innfiks(6.venstre: 3)
03) 3 != null; innfiks(3.venstre: 1)
04) 1 != null; innfiks(1.venstre: null)
05) null == null; … returnerer til node 1:
06) …println(1.element.toString())
07) innfiks(1.hoyre: null)
08) null == null; … returnerer til node 1:
09) returnerer... til 3
10) …println(3.element.toString())
11) innfiks(3.hoyre: 5)
12) 5 != null; innfiks(5.venstre: 4)
13) 4 != null; innfiks(4.venstre: null)
14) null == null; … returnerer til node 4:
15) …println(4.element.toString())
16) innfiks(4.hoyre: null)
17) null == null; … returnerer… til 4:
18) returnerer... til 5
19) …println(5.element.toString())
20) innfiks(5.hoyre: null)
21) Evt. til 6, skriver den, og så til hoyre
22) Evt. til 8, skriver den og avslutter.

BINÆRE SØKETRÆR – *Et større eksempel #8*

```
// Klasse for å ta vare på ordene og telle opp antall forekomster:
class IbsenElem implements Comparable
{   String ord;
    int antall;

    IbsenElem (String s)
    {   ord = s;
        antall = 1;
    }

    public int compareTo(Object x)
    {   IbsenElem e = (IbsenElem) x;
        return ord.compareTo(e.ord);
    }

    public String toString()
    {   return (ord + " " + antall);
    }
}
```

IbsenElem
ord: String antall: int
compareTo(Object): int toString(): String

```
// Tre sortert på ord:
class IbsenTre extends BinSokeTre
{ protected void oppdater(BinNode n, Comparable e)
   { IbsenElem ie = (IbsenElem) n.element;
     ie.antall++;
   }
}
```

```
// Tre sortert på frekvens:
class IbsenFrekTre extends BinSokeTre
{
    protected int sammenlign(Comparable c1, Comparable c2)
    {   IbsenElem e1 = (IbsenElem) c1;
        IbsenElem e2 = (IbsenElem) c2;
        return e1.antall - e2.antall;
    }

    protected void oppdater(BinNode n, Comparable e)
    {   if (n.venstre == null) n.venstre = new BinNode(e);
        else
        {   BinNode b = new BinNode(e);
            b.venstre = n.venstre;
            n.venstre = b;
        }
        antallNoder++;
    }

    public void innsetting(BinNode n)
    {   if (n != null)
        {   settInn(n.element);
            innsetting(n.venstre);
            innsetting(n.hoyre);
        }
    }
}
```

BINÆRE SØKETRÆR – *Et større eksempel #11, RESULTATER*

De mest brukte ordene (fra UiB):

	Antall	%	Kum.	ORD
1	19506	3.09	3.09	det
2	18241	2.89	5.98	jeg
3	17746	2.81	8.80	og
4	13232	2.10	10.89	i
5	13165	2.09	12.98	er
6	9639	1.53	14.51	at
7	9312	1.48	15.98	du
8	8927	1.42	17.40	ikke
9	8311	1.32	18.72	de
10	7711	1.22	19.94	en
11	7299	1.16	21.10	har
12	7152	1.13	22.23	som
13	6971	1.11	23.34	mig
14	6901	1.09	24.43	for
15	6795	1.08	25.51	til
16	6671	1.06	26.56	så
17	6314	1.00	27.56	med
18	5543	0.88	28.44	han
19	5524	0.88	29.32	den
20	5311	0.84	30.16	på

NESTE GANG – *Uke 5, Forelesning 2 (W5.L2)*

- Neste gang, dvs. uke 5, forelesning 2 (W5.L2):
  Dronningene!
  Spenning, show, alt-i-ett! VÆR DER!
- Ellers skal vi ta en pause fra trær og snakke om invarianter, hashing (W6.L1)
  og utvidbar hashing (W6.L2).
- Så kommer vi tilbake til B-trær (W7.L1)