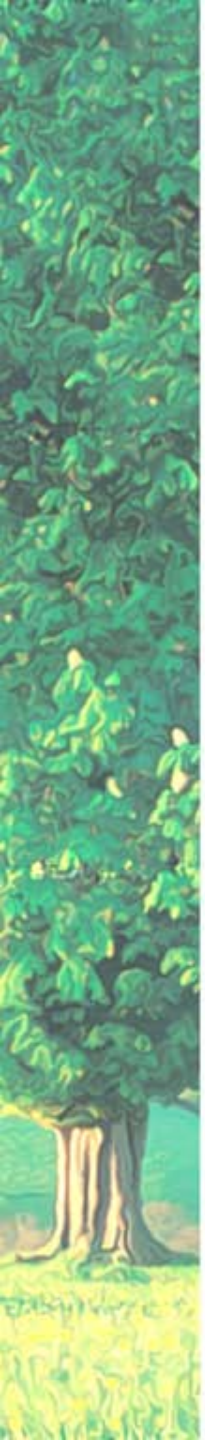
Kap.8 Sortering og søking

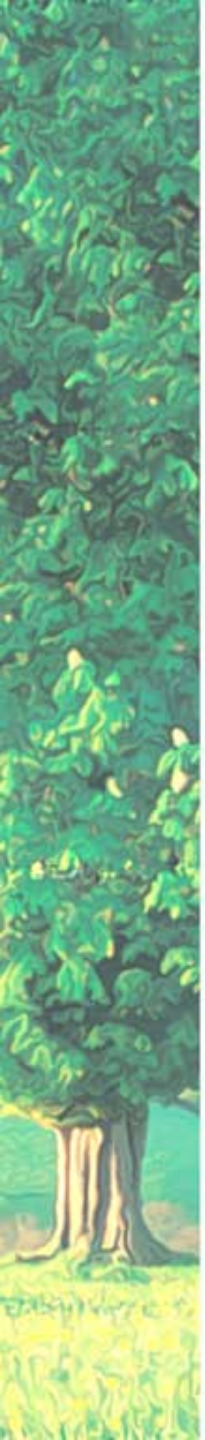
sist oppdatert 16.03

- Del 1 Søking
  - lineær søking m/u sorterte elementer
  - binærsøking
  - analyse
- Del 2 Sortering
  - ”gamle” sorteringsmetoder fra i høst
  - nye
  -analyse



Tema

- Undersøke lineær søking og binær søking.
- Undersøke flere sorteringsalgoritmer 8.2:
  - Utvalgssortering (skal kunne fra i høst)
  - Sortering ved innsetting (skal kunne fra i høst)
  - Bobblesortering (skal kunne fra i høst)
  - Kvikksortering (ny)
  - Flettesortering (ny)
    Kap.8.3: Radix-sort(ny, bruk av køer, sortering uten sml)
  - Kap.11: Haugsortering: (ny)
- Kunne analysere algoritmene



Søking

- *Søking er en prosess der en kan ta stilling til om et målelement fins eller ikke fins blant en samling av elementer.*
- Dette krever gjentatt sammenligning mellom målelementet og elementene i samlingen.
- En effektiv søking utfører ikke flere sammenligninger enn nødvendig.

Interface Comparable

- Vi ønsker å definere algoritmer slik at de kan søke etter et hvilket som helst objekt. Objektene som skal sorteres må være av en klasse som implementerer interface `Comparable`.
- Comparable inneholder metoden, `compareTo`, som er laget for å returnere et heltall som angir relasjonen mellom to objekter : **`obj1.compareTo(obj2)`**
- Dette kallet returnerer et tall mindre, lik eller større enn 0 hvis `obj1` er mindre enn, lik eller større enn `obj2` h.h.v..

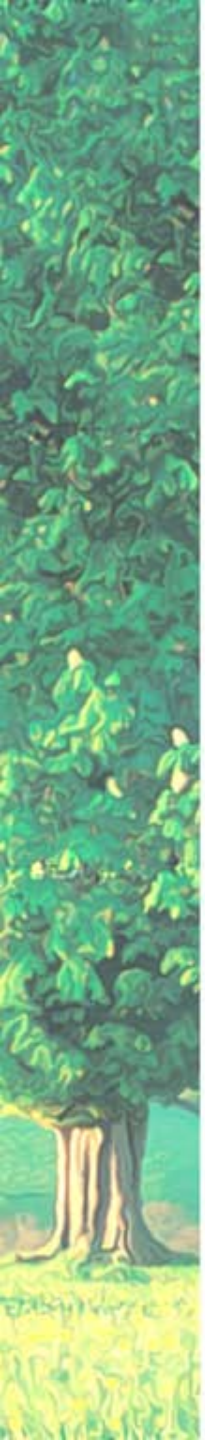
Interface Comparable

- Alle metodene som blir presentert er metoder i klassen SorteringOgSøking.
- Denne klassen gjør bruk av den generiske typen T.
- Vi kan la T arve Comparable

```
public class
SorteringOgSøking<T extends
                  Comparable>
```

Lineær søking

- Ved *lineærsøk* undersøkes hvert element i søkeområdet, ett om gangen, inntil målelementet er funnet eller til vi kan avgjøre at målelementet ikke fins.



Lineær søking

- Vi har foreløpig ikke antatt at elementene er sorterte.
- Vi undersøker hvert element i tur og orden på en lineær måte.
- Enkelt å forstå, men ikke spesielt effektivt.

Lineær søking

Sett funnet til usann

- **Gjenta så lenge flere elementer og ikke funnet**
  - **Hvis elementet er lik søkeelementet sett funnet til sann**

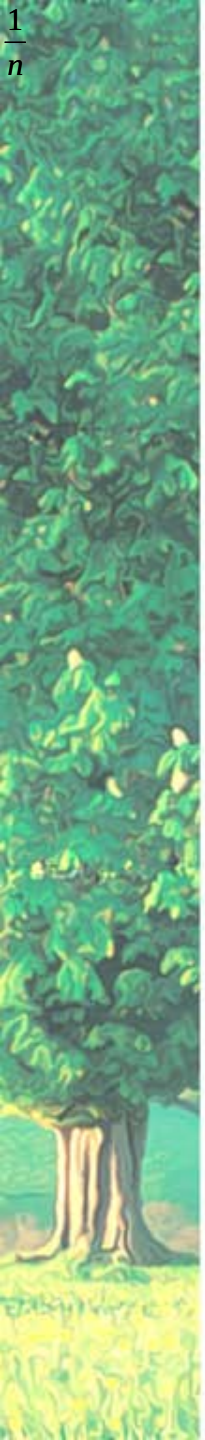
Lineær søking i usortert tabell

```
//Anta tabelllen er fylt med data
public boolean lineærSøkU(T[] data,
     int min, int maks, T element){
  int indeks = min;
  boolean funnet = false;
  while (indeks <= maks &&!funnet){

  if(data[indeks].compareTo(element)
       == 0)
       funnet = true;
    indeks++;
  }//while
  return funnet;}//metode
```

Analyse lineærsøk (1) - usortert, elementet fins

- Listen av n antall elementer er **usortert.** Vi antar elementene er ulike.
  Vi søker etter målelement som **fins** i listen. Vi skal finne antall sml. (antall kall av compareTo()).
- I verste tilfelle må vi søke gjennom hele listen til vi finner det som siste element.
- Antall sml. blir da n, dvs O(n).



Analyse lineærsøk (1) - usortert, elementet fins

Et mål for gj.sn. antall sml blir:

$$(1 + 2 + 3 + \ldots + n)/n = n(n+1)/2n =$$

$$(n+1)/2$$

Dvs O(n)

Analyse lineærsøk (2) - usortert, elementet fins ikke

- Listen av n antall elementer er **usortert.** Alle elementene er ulike. Vi søker etter målelementet som i**kke fins** i listen. Vi skal finne antall sml.
- Vi må alltid søke gjennom hele listen.
- Antall sml blir n både i det verste tilfellet og i gj.sn. Altså O(n).

Lineær søking i sortert tabell

NB! Vi kan avbryte søkingen tidligere i en sortert tabell hvis elementet ikke fins

Sett funnet til usann

Gjenta så lenge flere elementer og
aktuelt element < søkeelementet
hent neste element
Hvis aktuelt element er lik søkeelementet
sett funnet til sann

Lineær søking i sortert tabell

NB! Vi kan avbryte søkingen tidligere i en sortert tabell hvis elementet ikke fins

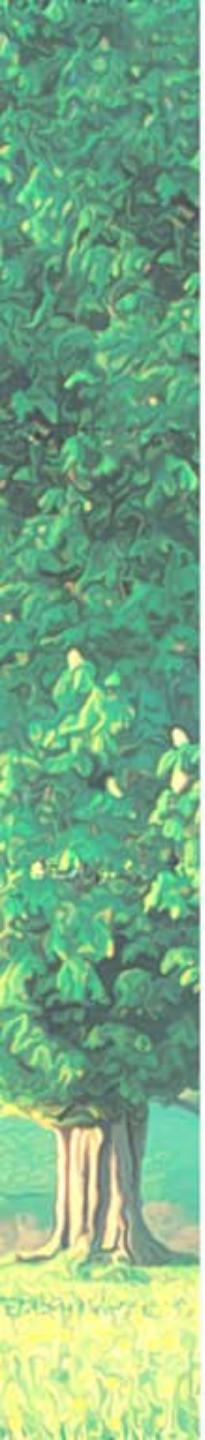
```
//Anta tabelllen er fyllt med data
public boolean lineærSøkS(T[] data, int
  min,int maks,T element)
   int indeks = min;
   boolean funnet = false;
   while(indeks <= maks &&
   data[indeks].compareTo(element)
                      < 0){
           indeks++;
   }//while
   if(data[indeks].compareTo(element))
                                  == 0)
      funnet = true;
   return funnet;
}//metode
```

Analyse lineærsøk (3) – sortert, elementet fins

- Listen av n antall elementer er **sortert**. Alle elementene er ulike. Vi søker etter målelementet som **fins** i listen. Vi skal finne antall sml.
- I verste tilfelle må vi søke gjennom hele listen til vi finner målelementet som det siste elementet. Antall sml. blir da n, dvs O(n).
- Gj.sn. antall sml. blir (n +1)/2, dvs O(n)

Analyse lineærsøk (4) - sortert, elementet fins ikke

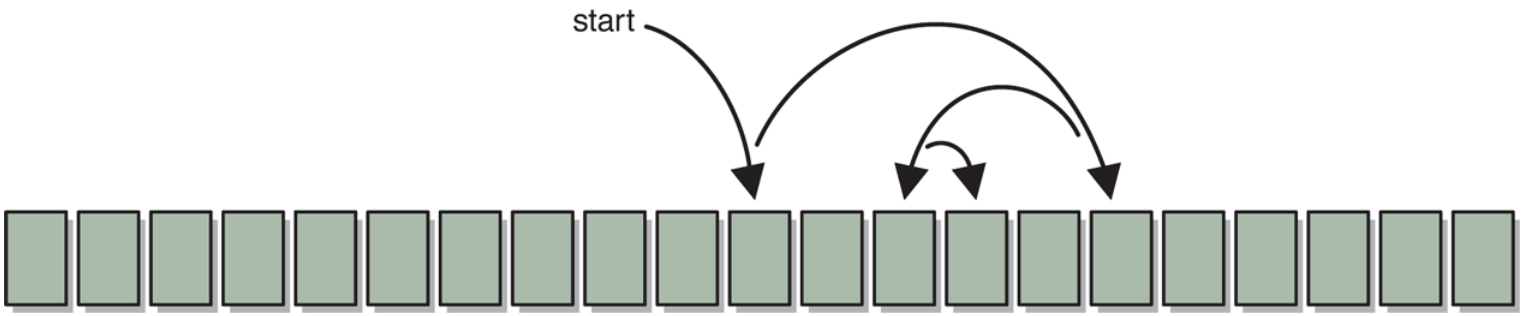
- Listen av n antall elementer er **sortert**.
  Vi søker etter målelement som **ikke fins** i listen.
- Vi må søke gjennom inntil et element er større enn målelementet.
  Antall sml. i verste tilfellet blir n og gj.sn blir ca n/2**,** dvs. O(n) i begge tilfeller.
- **Lineærsøk klarer ikke utnytte sorteringen på en spesielt god måte. O(n).**



Binærsøk

- **Hvis elementene er sorterte** kan vi utnytte det mer effektivt enn ved lineærsøk.
- **En *binær søking eliminerer større deler av søkeområdet ved hver sammenligning.***
- I stedet for å starte søking i en ende, så starter vi søkingen i midten.
- Hvis elementet vi søker etter ikke påtreffes på midtelementet vet vi at hvis målelementet fins, så må det være i den ene av de to halv-delene.
- Vi kan da hoppe til midten av den “riktige” halvdelen og fortsette der på samme måte.

FIGUR 8.2 Binærsøk



Binærsøk

- For eksempel, finn tallet 29 i følgende sorterte liste av tall:

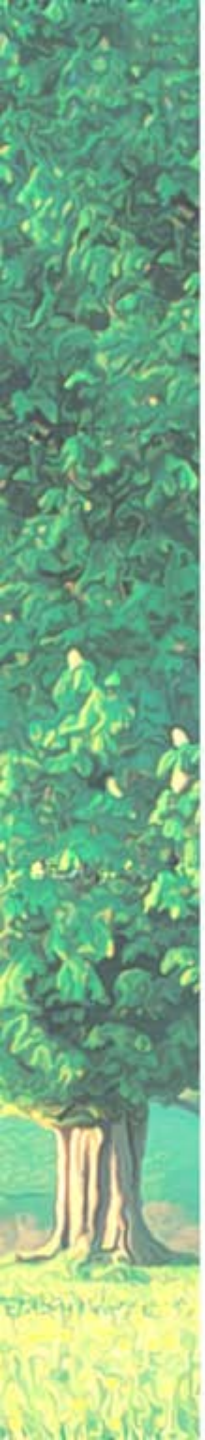
  8 15 22 29 36 54 55 61 70 73 88

- Sammenlign målelementet 29 med midtverdien 54.
- Vi vet nå at hvis 29 fins i listen, så må det være i nedre halvdelen av listen.
- **Med en sammenligning har vi eliminert halvparten av elementene**.

Analyse binærsøk

- Ved starten har vi n elementer.
- Etter første kall av metoden binærSøk har vi n/2 elementer.
- Etter andre kall av metoden binærsøk har vi $n/4 = n/2^2$ elementer…osv.
- Generelt etter i'te kall av metoden binærsøk har vi $n/2^i$ elementer.
- Det maksimale antall (i verste tilfellet) rekursive kall er det minste heltall m slik at $n/2^m < 1$.
  Dvs. $m > \log_2 n$ Slik at vi får: $m = \lfloor \mathbf{\log_2 n} \rfloor \mathbf{+ 1}$.

  Dvs arbeidsmengden i verste tilfelle er $\mathbf{O(\log_2 n)}$.
- Sml arbeidsmengden ved **lineært** søk i en tabell i verste tilfelle: **O(n)**.
- $\lfloor \ \rfloor$ betyr rundet av nedover til nærmeste heltall.



Binærsøking

- En binær søkealgoritme er ofte implementert rekursivt.
- Hvert rekursivt kall søker i mindre og mindre søkeområder (ca halvparten av foregående).

Binærsøking

sett funnet til usann

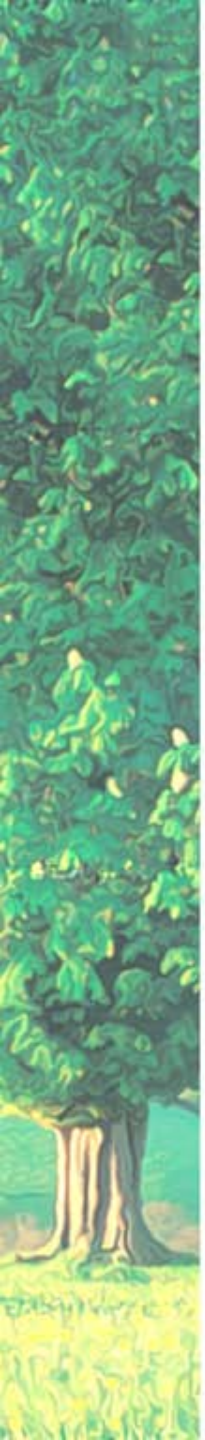
Hvis flere elementer og ikke funnet
finn midtelementet

Hvis midtelementet er lik søkeelementet
så sett funnet til sann
Hvis midtelementet er større enn
søkeelementet så fortsett binær søking til
venstre for midtelementet
Hvis midtelementet er mindre enn
søkeelementet så fortsett binær søking til
høyre for midtelementet

returner funnet

Binærsøk rekursiv

```
public boolean binærSøk(T[] data, int min, int
  maks,
                          int element){
  boolean funnet = false;
  int midtpunkt = (min + maks) / 2;  //
  midtpunktet
  if (data[midtpunkt].compareTo(element) == 0)
      funnet = true;
  else if(data[midtpunkt].compareTo(element) > 0){
       if(min <= midtpunkt - 1)
         funnet = binærSøk(data, min, midtpunkt-1,
                           element);
       }
  else if (midtpunkt + 1 <= maks)
       funnet = binærSøk(data, midtpunkt+1, maks,
                           element);
  return funnet;
  }//
```



Sammenligning av søkealgoritmer

- I gj.sn. vil et lineært søk sammenligne n/2 elementer hvis målelementet fins.
- Derfor er lineærsøk O(n).
- Verste tilfellet for et binærsøk er ($\log_2 n$) sml.
- **Et binærsøk er en *logaritmisk algoritme*.**
- **Den har tidskompleksitet av O($\log_2 n$).**
- **Men husk at for binærsøk må elementene være sorterte.**
- For stor n, vil binærsøk være mye raskere enn lineærsøk. Tegn opp n og logn!