Kapittel 9:
Sortering og søking
Kort versjon

Redigert av:

Khalid Azim Mughal (`khalid@ii.uib.no`)

Kilde:

Java som første programmeringsspråk (3. utgave)

Khalid Azim Mughal, Torill Hamre, Rolf W. Rasmussen
Cappelen Akademisk Forlag, 2006.
ISBN: 82-02-24554-0
`http://www.ii.uib.no/~khalid/jfps3u/`
(NB! Boken dekker opptil Java 6, men notatene er oppdatert til Java 7.)

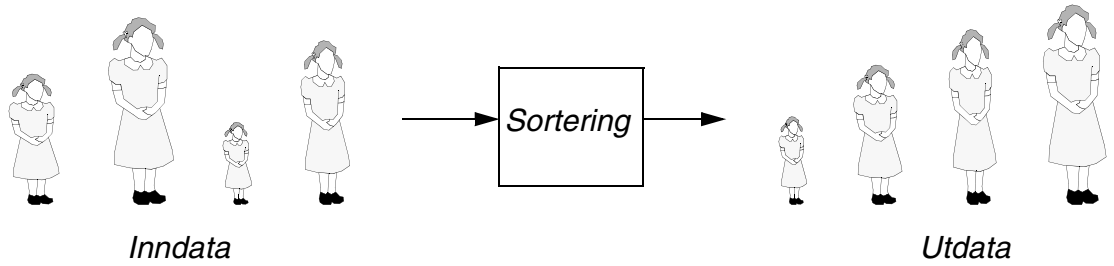
Emneoversikt

Pseudokode

Ordnede datamengder
- Naturlig ordning
- Sammenligningsoperatorer og -metoder

Sorteringsalgoritmer:
- Boblesortering
- Sortering ved utvalg
- Sortering ved innsetting

- Sortering av objekter
  - Kontrakten `Comparable`
- Søking
  - Lineært søk
  - Binært søk
  - Eksempel: sortering og søking med objekter



Pseudokode

- Pseudokode beskriver en algoritme i vanlig språk.
- Kontrollflyt angis vha. spesielle ord, f.eks. **`hvis`** for å innlede en valgsetning.
- Strukturen til handlingene i algoritmen vises vha. innrykk.
- Eksempel: boblesortering

For hvert gjennomløp:
  For hvert nabopar i usortert del av tabellen:
    Hvis verdier i nabopar står i feil rekkefølge:
      Bytt om verdier i nabopar

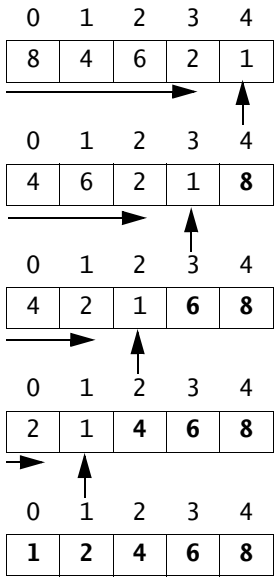
- Pseudokode er skrevet *av* og *for* programmerere.
- Den er et nyttig hjelpemiddel i algoritmeutvikling.
- Kompliserte steg kan forfines videre inntil man forstår problemet tilstrekkelig til å kunne starte kodingen.
- Dessuten kan pseudokode skrives inn i kildekoden, og dermed inngå som en del av dokumentasjonen.

Ordnede datamengder

- Tallverdier og tegn kan sammenlignes vha. operatorene:
  - == (likhet)
  - != (ulikhet)
  - < (mindre enn)
  - > (større enn),
  - <= (mindre eller lik)
  - >= (større eller lik)

Boblesortering

Tabell a:



Gitt n = antall elementer -1, dvs lik 4.

1. pass:
   for k = 0(1)n-1:
   hvis $element_k$ er større enn $element_{k+1}$,
   bytt elementene.
2. pass:
   for k = 0(1)n-2:
   hvis $element_k$ er større enn $element_{k+1}$,
   bytt elementene.
3. pass:
   for k = 0(1)n-3:
   hvis $element_k$ er større enn $element_{k+1}$,
   bytt elementene.
4. pass:
   for k = 0(1)n-4:
   hvis $element_k$ er større enn $element_{k+1}$,
   bytt elementene.

Antall pass er lik n, dvs antall elementer -1.

$a[0] \leq a[1] \leq \ldots \leq a[n]$

```
int a[] = {8,4,6,2,1};

// For hvert gjennomløp:
for (int i = a.length -1;
     i > 0;
     i--) {
    // For hvert nabopar i
    // usortert del av tabellen
    for (int k = 0; k < i; k++)
        // Hvis verdier i
        // nabopar står i feil
        // rekkefølge: Bytt om
        // verdier i nabopar
        if (a[k] > a[k+1]) {
            int temp = a[k];
            a[k] = a[k+1];
            a[k+1] = temp;
        }
}
```

Denne algoritmen kan forbedres.

Sortering ved utvalg

Tabell a:

0	1	2	3	4
8	4	6	2	**1**

- finn minste i hele tabellen.
- bytt a[0] med den minste.

0	1	2	3	4
1	4	6	**2**	8

- finn minste fra a[1] til a[4].
- bytt a[1] med den minste.

0	1	2	3	4
1	**2**	6	4	8

- finn minste fra a[2] til a[4].
- bytt a[2] med den minste.

osv.

0	1	2	3	4
1	**2**	**4**	6	8

0	1	2	3	4
1	**2**	**4**	**6**	**8**

- fortsett inntil hele tabellen er gjennomgått.

```
int a[] = {8,4,6,2,1};

// For hvert gjennomløp:
for (int ind = 0;
     ind < a.length - 1;
     ind++) {

    // Finn minste-verdi i usortert
    // del av tabellen
    int ind_til_minste = ind;
    for (int k = ind + 1; k < a.length; k++)
        if (a[k] < a[ind_til_minste])
            ind_til_minste = k ;
    // Bytt om på minste-verdi og første
    // element i usortert del
    int temp = a[ind_til_minste];
    a[ind_til_minste] = a[ind];
    a[ind] = temp;
}
```


Sortering ved innsetting

Tabell a:

0	1	2	3	4
8	**4**	6	2	1

sett inn verdi til a[1] på riktig plass s.a. a[0]≤a[1].

0	1	2	3	4
4	8	**6**	2	1

sett inn verdi til a[2] på riktig plass s.a. a[0]≤a[1] ≤a[2].

0	1	2	3	4
4	6	8	**2**	1

sett inn verdi til a[3] på riktig plass s.a. a[0]≤a[1] ≤a[2] ≤a[3].

0	1	2	3	4
2	4	6	8	**1**

sett inn verdi til a[4] på riktig plass s.a. a[0]≤a[1] ... ≤a[4].

0	1	2	3	4
1	2	4	6	**8**

for (i = 1; i < a.length; i++)

- sett verdi til *a[i]* inn i riktig posisjon *j* blant elementene *a[0]...a[i].*

for (i = 1; i < a.length; i++)

- finn riktig posisjon *j*, *0 ≤j ≤i*, for verdi til *a[i]* ved å flytte verdiene til *a[j]...a[i-1]* en posisjon til høyre.
- plasser verdi til *a[i]* i posisjonen *j*.

```
int a[] = {8,4,6,2,1};
// For hvert gjennomløp:
for (int i = 1; i < a.length; i++) {
    // Ta vare på neste-verdi som innsettes
    int verdi = a[i];
    // Forskyv alle verdier i sortert
    // deltabell som er større enn denne
    // ett hakk bakover i tabellen
    int j;
    for (j = i;
         (j > 0 && verdi < a[j-1]);
         j--)
        a[j] = a[j-1];  // flytt bakover!
    a[j] = verdi; // inn på riktig plass!
}
```

Sortering av objekter

- *Naturlig ordning* for verdier av type `T`:
  - To verdier `a` og `b` av en type `T` kan sammenlignes for å avgjøre om a er *mindre enn*, *lik* eller *større enn* `b`.
- Objekter *sammenlignes* for likhet vha. == (referanselikhet) og `equals()` (objektlikhet).
  - Metoden `equals()` må *overkjøres* hvis superklassens metode ikke har en passende implementasjon.
  - Pass på at du ikke *overlaster* `equals()`-metoden!
- Objekters *innbyrdes ordning* avgjøres vha. metoden `compareTo()`, som er spesifisert i kontrakten `Comparable`.

Kontrakten `Comparable`

- Objekters *innbyrdes ordning* avgjøres vha. metoden `compareTo()`, som er spesifisert i kontrakten `Comparable`.

```
interface Comparable {
    int compareTo(Object objRef);
}
```

- Kallet `obj1.compareTo(obj2)` skal returnere et heltall som er:
  - == 0 hvis `obj1` og `obj2` er like
  - < 0 hvis `obj1` kommer før `obj2`
  - > 0 hvis `obj1` kommer etter `obj2` i den naturlige ordningen.
- Alle klasser der objektene skal ha en *naturlig ordning* må implementere denne kontrakten.
- Implementasjonen av `compareTo()`-metoden må være i samsvar med klassens implementasjon av `equals()`-metoden.

Sortering av objekter

- En klasse må implementere kontrakten `Comparable`, dersom objekter skal ha en naturlig ordning.
- Klassen `String` implementerer kontrakten `Comparable`, og det gjør også wrapper-klassene for primitive verdier.

```
public static void sortVedUtvalg(Comparable[] tab) {
  // For hvert gjennomløp:
  for (int ind = 0; ind < tab.length - 1; ++ind) {
    // Finn minste-verdi i usortert del av tabellen
    int indTilMinste = ind;
    for (int k = ind + 1; k < tab.length; ++k) {
      if (tab[k].compareTo(tab[indTilMinste]) < 0) {   // (1)
        indTilMinste = k ;
      }
    }
    // Bytt om minste-verdi og første element i usortert del
    Comparable temp = tab[ind];                        // (2)
    tab[ind] = tab[indTilMinste];
    tab[indTilMinste] = temp;
  }
}
```



```
public class SortUtvalgKlient {
  public static void main(String args[]) {
    Integer iTab[] = {8,4,6,2,1};

    HjelpeKlasse.skrivTabell(iTab, "Før sortering: ");

    HjelpeKlasse.sortVedUtvalg(iTab);

    HjelpeKlasse.skrivTabell(iTab, "Etter sortering: ");
  }
}
```

Utskrift fra kjøring:

```
Før sortering: 8 4 6 2 1
Etter sortering: 1 2 4 6 8
```

Lineært søk

Problem: Finn en gitt *verdi* (om den finnes!) i en tabell. Verdien vi søker etter kalles for en *nøkkel*.

```
int a[] = {8,4,6,2,1};
int nøkkel = 6;
boolean funnet = false;
int teller = 0;
while (teller < a.length && !funnet)
    if (a[teller] == nøkkel)
        funnet = true; // verdi er funnet
    else
        teller++;      // ellers fortsett i tabellen
// !(teller < a.length && !funnet) <=> (teller >= a.length || funnet)
assert (teller >= a.length || funnet) : "Feil i lineært søk";
if (funnet)
    System.out.println("Verdi " + nøkkel +
                       " funnet i element med indeks " + teller);
else
    System.out.println("Verdi " + nøkkel + " finnes ikke!");
```

- Nøkkelen kan være hvilket som helst element siden tabellen er usortert; dette krever (`N_ELEMENT/2`) sammenligninger i gjennomsnitt.

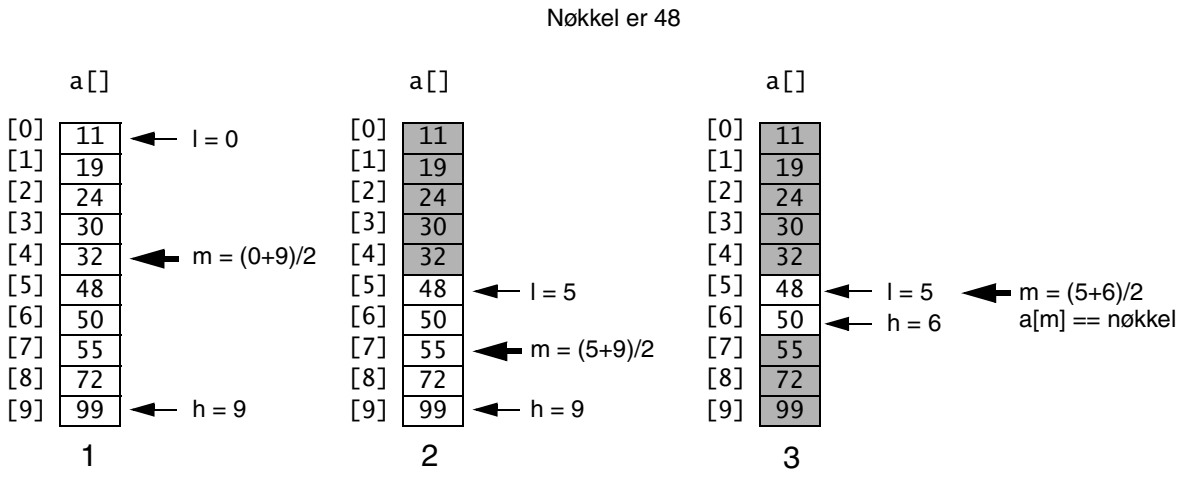
Binært søk

Problem: Finn en gitt verdi (om den finnes!) i en *sortert* tabell

```
int a[] = {11,19,24,30,32,48,50,55,72,99}; // verdier i stigende rekkefølge.
int nøkkel = 48;

int l = 0, h = a.length - 1, m = -1;
boolean funnet = false, finnes = true;
while (finnes && !funnet) {
    m = (l + h) / 2;                          // midten av tabellen
    if (l > h) finnes = false;                // nøkkel finnes ikke
    else if (a[m] == nøkkel) funnet = true;   // nøkkel er i a[m]
    else if (a[m] < nøkkel) l = m + 1; // finnes nøkkel, så er den i øvre halvparten
    else h = m - 1;                    // finnes nøkkel, så er den i nedre halvparten
}
// !(finnes && !funnet) <=> (!finnes || funnet)
assert (!finnes || funnet) : "Feil i binært søk";
if (funnet)
    System.out.println("Verdi " + nøkkel + " funnet i element med indeks " + m);
else
    System.out.println("Verdi " + nøkkel + " finnes ikke!");
```

Binært søk (forts.)



Maks. antall sammenligninger angitt ved $2^{\texttt{sammenligninger}} = \texttt{N\_ELEMENT}$, dvs. $\texttt{sammenligninger} = \lceil \log_2 \texttt{N\_ELEMENT} \rceil$,
f.eks. for `N_ELEMENT` lik 9, er `sammenligninger` lik 3.

Lineærsøk for objekter

- Objekter sammenlignes for *objektlikhet* med `equals()`-metoden.
- Klassen `String` implementerer `equals()`-metoden, og det gjør også wrapper-klassene for primitive verdier.
- Lineærsøk-metoden fra `HjelpeKlasse`.

```
public static int linaersoek(Object[] tab, Object nøkkel) {
  boolean funnet = false;
  int teller = 0;
  while (teller < tab.length && !funnet)
    if (nøkkel.equals(tab[teller]))
      funnet = true; // verdi er funnet
    else
      ++teller;      // ellers fortsett nedover
  // !(teller < tab.length && !funnet) <=> (teller >= tab.length || funnet)
  assert (teller >= tab.length || funnet) : "Feil i lineært søk";
  if (funnet)
    System.out.println("Verdi " + nøkkel +
        " funnet i element med indeks " + teller);
  else
    System.out.println("Verdi " + nøkkel + " finnes ikke!");
  return teller;  // Returnerer indeksen der nøkkel er eller burde vært.
}
```

Binært søk for objekter

- Binært søk-metoden fra `HjelpeKlasse`.

```
public static int binsøk(Comparable[] tab, Comparable nøkkel) {
    int l = 0, h = tab.length - 1, m = -1;
    boolean funnet = false, finnes = true;
    while (finnes && !funnet) {
        m = (l + h) / 2;                          // midten av tabellen
        if (l > h) finnes = false;                // nøkkel finnes ikke
        else {
            int status = tab[m].compareTo(nøkkel);  // Sammenlign
            if (status == 0) funnet = true; // nøkkel er i tab[m]
            else if (status < 0) l = m + 1; // finnes nøkkel,
                                            // så er den i øvre halvparten
            else h = m - 1; // finnes nøkkel, så er den i nedre halvparten
        }
    }
    // !(finnes && !funnet) <=> (!finnes || funnet)
    assert (!finnes || funnet) : "Feil i binært søk";
    if (funnet)
        System.out.println(nøkkel + " funnet i element med indeks " + m);
    else
        System.out.println(nøkkel + " finnes ikke!");
    return m;  // Returnerer indeksen der nøkkel er eller burde vært.
}
```


Eksempel: sortering og søking med objekter

```
public class Vare implements Comparable {
    String varenavn;   // navn på det som kan kjøpes
    int beholdning;    // antallet som er ikke solgt

    Vare(String varenavn, int beholdning) {
        this.varenavn = varenavn;
        this.beholdning = beholdning;
    }

    public String hentVarenavn() { return varenavn; }
    public int hentBeholdning() { return beholdning; }
    public void settVarenavn(String varenavn) {
        this.varenavn = varenavn;
    }

    public void settBeholdning(int beholdning) {
        this.beholdning = beholdning;
    }
```

```
  @Override
  public int compareTo(Object obj) {
    if (this == obj) return 0;
    Vare vare = (Vare) obj;
    int status = this.varenavn.compareTo(vare.varenavn);
    if (status == 0) { // samme varenavn?
      status = this.beholdning - vare.beholdning;
    }
    return status;
  }

  @Override
  public boolean equals(Object obj) {  // Samsvar med compareTo()
    if (!(obj instanceof Vare)) return false;
      return this.compareTo(obj) == 0;
  }
  // ...
}
```


Eksempel: sortering og søking med objekter (forts.)

- Bruker sorteringsmetoden fra `HjelpeKlasse`.

```
public class VareKatalog {
  public static void main(String[] args) {
    Vare[] varekatalog = {
        new Vare("Cola 0.5l", 30),
        new Vare("Cola 0.33l", 20),
        new Vare("Pepsi 0.5l", 20),
        new Vare("Solo 0.5l", 10),
        new Vare("Cola 0.33l", 25)
    };
    HjelpeKlasse.skrivTabell(varekatalog, "Usortert varekatalog:");
    HjelpeKlasse.sortVedUtvalg(varekatalog);
    HjelpeKlasse.skrivTabell(varekatalog, "Sortert varekatalog:");
    HjelpeKlasse.linaersoek(varekatalog, new Vare("Solo 0.5l", 10));
    HjelpeKlasse.linaersoek(varekatalog, new Vare("Pepsi 0.5l", 10));
  }
}
```

Utskrift fra kjøring:

```
Usortert varekatalog:
Varenavn: Cola 0.5l Beholdning: 30
Varenavn: Cola 0.33l  Beholdning: 20
Varenavn: Pepsi 0.5l  Beholdning: 20
Varenavn: Solo 0.5l Beholdning: 10
Varenavn: Cola 0.33l  Beholdning: 25
Sortert varekatalog:
Varenavn: Cola 0.33l  Beholdning: 20
Varenavn: Cola 0.33l  Beholdning: 25
Varenavn: Cola 0.5l Beholdning: 30
Varenavn: Pepsi 0.5l  Beholdning: 20
Varenavn: Solo 0.5l Beholdning: 10

Varenavn: Solo 0.5l Beholdning: 10 funnet i element med indeks 4

Varenavn: Pepsi 0.5l  Beholdning: 10 finnes ikke!
```