

Uke 3, Forelesning 2



HUSK – Hittil...

- Essensen i faget: Effektive algoritmer
 - Matematiske forutsetninger
 - Introduksjon til metodekall og rekursjon
 - Introduksjon til kombinatoriske søk, kombinasjoner og permutasjoner
 - Introduksjon til analyse av algoritmer
- Uker 1 + 2
-
- Introduksjon til ADT'er
 - Lister
- Uke 3.1
-
- Mer om lister...
 - Stabler
 - Køer
- Uke 3.2
...
Uke 4.1

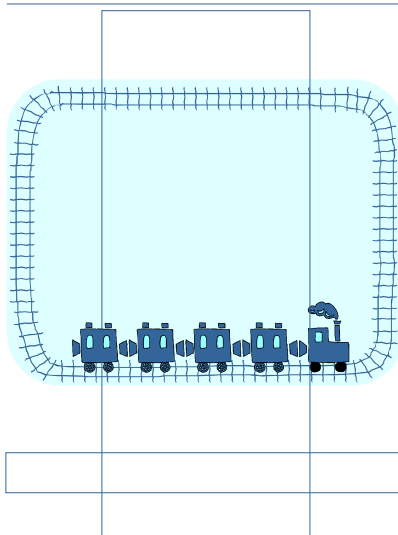


OVERSIKT – Uke 3, Forelesning 2 (W3.L2)

- Denne uken: **LISTER**, **STABLER** og **KØER**.
- **FORRIGE FORELESNING (W3.L1):**
 - TEMA #1: Introduksjon til abstrakte datatyper (ADT'er)
 - TEMA #2: **Lister**
- **I DAG (W3.L2):**
 - Litt mer om TEMA #2: **Lister**
 - TEMA #3: **Stabler** (Engelsk: Stacks, Norsgelsk: Stakker)
 - TEMA #4: **Køer**
- Vil avsluttes i uke 4, forelesning 1 (W4.L1)



TEMA #2 – Fortsetter...



Mer om Lister...



LISTER – Pekerkjedeliste med hode og siste-peker #1

```
public class IntNode
{
    int element;
    IntNode neste;

    IntNode (int i)
    {
        element = i;
    }
}

class Liste
{
    IntNode liste = new IntNode(-1);
    IntNode siste = liste;

    IntNode forste()
    {
        return liste.neste;
    }

    // print, insert og remove: neste foiler...
    // skal settes inn her.

}
```



LISTER – Pekerkjedeliste med hode og siste-peker #2

```
class Liste
{
    //Flere metoder. Fortsetter...

    void print()
    {
        IntNode n = liste.neste;
        while (n != null)
        {
            System.out.print(n.element + " ");
            n = n.neste;
        }
        System.out.print("\n");
    }

    // insert og remove: neste foiler...
    // skal settes inn her.

}
```



LISTER – Pøkerkjedeliste med hode og siste-pøker #3

```
class Liste
{ // Flere metoder. Fortsetter...

    void insert(int i)
    { siste.neste = new IntNode(i);
      siste = siste.neste;
    }

    void remove(int i)
    { IntNode n = liste;
      while (n.neste.element != i)
      { n = n.neste;
      }
      n.neste = n.neste.neste;
      if (n.neste == null)
      { siste = n;
      }
    }

    //Avslutter klasse "Liste"
}
}
```



LISTER – Andre alternativer...

- **Multilister:**
Som en "matrise",
Lister på kolonner og rader (enkle eller doble kjeder,
array'er og lignende)
- **Cursor-implementasjon av lenkede lister:**
Når pøkere eller referanser til objekter ikke finnes,
simulerer vi disse mekanismene ved hjelp av records og
pointer-to-records osv...



HUSK at sortert liste er en mulighet (for implementasjon):

- **Find** blir fortsatt $O(n)$ hvis vi bruker pekerkjede, men $O(\log n)$ hvis vi bruker array og binærsøk.

Eksempel: Radix-sortering (kort-sortering i gamle dager)

Er en generalisering av ...

Bøtte-sortering

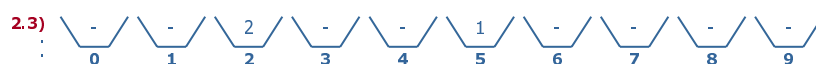
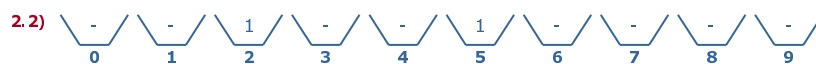
- Hvis vi har N heltall i området 0 til $M-1$, kan disse sorteres som følger:
 1. Opprett en heltallsarray `antall[M]`, initialisert til 0 (**bøttene**).
 2. For hver input med verdi a_i , øk `antall[ai]` med én.
 3. Etter at all inputen er lest, gå gjennom arrayen og skriv ut en representasjon av den sorterte listen.
- Denne algoritmen tar $O(M + N)$ tid, der M er antall bøtter og N er antallet tall som skal sorteres.
- Hvis M og N er omtrent like store, blir bøttesortering $O(N)$.



- Eksempel rekke: 5, 2, 2, 6, 1, 9, 0 i området 0 til 9 (10 bøtter):

Bøtte-sortering var: Hvis vi har N heltall i området 0 til $M-1$:

- 1) Opprett en heltallsarray `antall[M]`, initialisert til 0 (**bøttene**).
- 2) For hver input med verdi a_i , øk `antall[ai]` med én.
- 3) Når all input er lest, gå gjennom array'en og skriv ut en representasjon av den sorterte listen.



3) 0, 1, 2, 2, 5, 6, 9

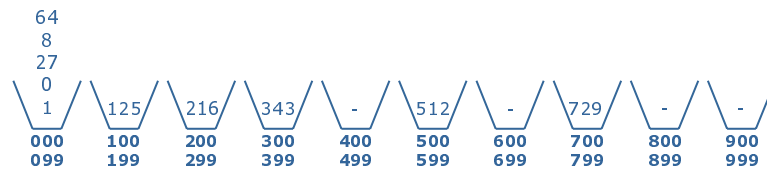


Et annet eksempel: 10 heltall i intervallet 0 til 999...
64, 8, 216, 512, 27, 729, 0, 1, 343, 125

MEN... BØR vi opprette 1000 bøtter for 10 tall??? Nei, helst ikke!

Idé:

- La bøttene representere områdene 0-99, 100-199 osv. til 900-999
- Les tallene og legg de i sine respektive bøtter



- Når ferdig, sorter bøttene: **Færre å sortere!**
- Eksempel: Kun 5 å sortere i bøtte for 000-099.

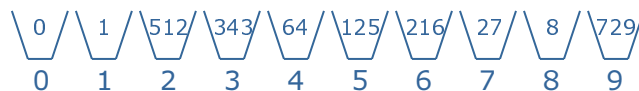


Samme eksempel med Radix-sort:

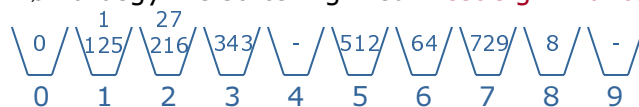
10 heltall i intervallet 0 til 999:
64, 8, 216, 512, 27, 729, 0, 1, 343, 125

Idé:

Sorter etter **minst signifikante siffer**, deretter det nest minste signifikante siffer og så videre.



Prøv å begynne sortering med **mest signifikante siffer** også.

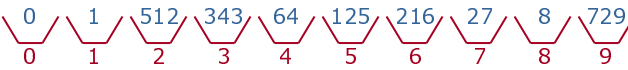


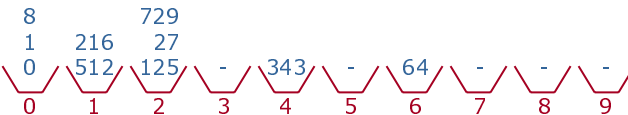
Mer naturlig?
Won't Work!

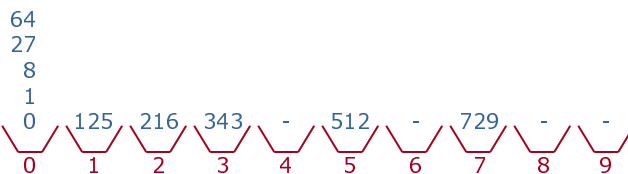


LISTER – Radix-sort #2

- Sorter etter **minst signifikante siffer**, deretter det nest minste signifikante siffer og så videre...

Pass 1. 

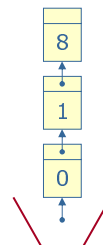
Pass 2. 

Pass 3. 



LISTER – Radix-sort #3

- Hvis vi har N elementer som skal sorteres,
... B antall bølter,
... og P "pass" ...
- Da vil kjøretiden være = $O(P \times (N + B))$
- NB! Bøttestørrelse og antall "pass" er så-kalte "design issues"!
- NB!
Bølter er ofte implementert som lister...



LISTER – Radix i bøtter og spann #1

```
public class Radix
{ // Bøtter implementert som "array av Liste"...
  static Liste[] botter;

  public static void main (String[] args)
  { int[] input = {64, 8, 216, 512, 27, 729, 0, 1,343, 125};
    // Initialiserer bøtte-listene
    botter = new Liste[10];
    for (int i = 0; i < 10; i++)
    { botter[i] = new Liste();
    }

    // Plasserer input i bøttene etter siste siffer
    for (int i = 0; i < input.length; i++)
    { int tall = input[i];
      int indeks = tall%10;
      botter[indeks].insert(tall);
    }

    // Fortsetter... (klassen avslutter på neste side).
```



LISTER – Radix i bøtter og spann #2

```
public class Radix
{ //Fortsetter fra forrige foil...

  public static void main (String[] args)
  { // Fortsetter fra forrige foil...

    // *****
    // Her skal selve sorteringen inn - se neste foil
    // *****

    for (int i = 0; i < 10; i++)
    { System.out.print("Bøtte " + i + ": ");
      botter[i].print();
    }
  }
}
```

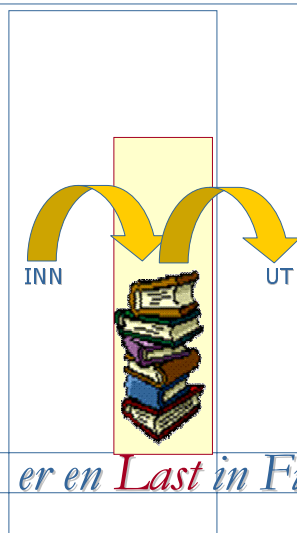


LISTER – Radix i botter og spann #3

```
// Selve sorteringen... 3 Pass... Skal inn i klassen Radix
for (int pass = 2; pass <= 3; pass++)
{ //Går gjennom hver bøtte og sorterer i henhold til sifferet
  //på plassen pass bakfra.
  for (int i = 0; i < 10; i++)
  { IntNode node = botter[i].forste();
    while (node != null)
    { int tall = node.element;
      // Finner de pass bakerste sifferne
      int siffer = tall%(int)Math.pow(10,pass);
      // Plukker ut det første av disse
      siffer = siffer/(int)Math.pow(10, pass-1);
      if (siffer != i)
      { botter[i].remove(tall);
        botter[siffer].insert(tall);
      }
      node = node.neste;
    }
  }
}
```



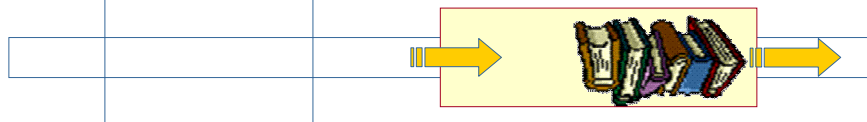
TEMA #3



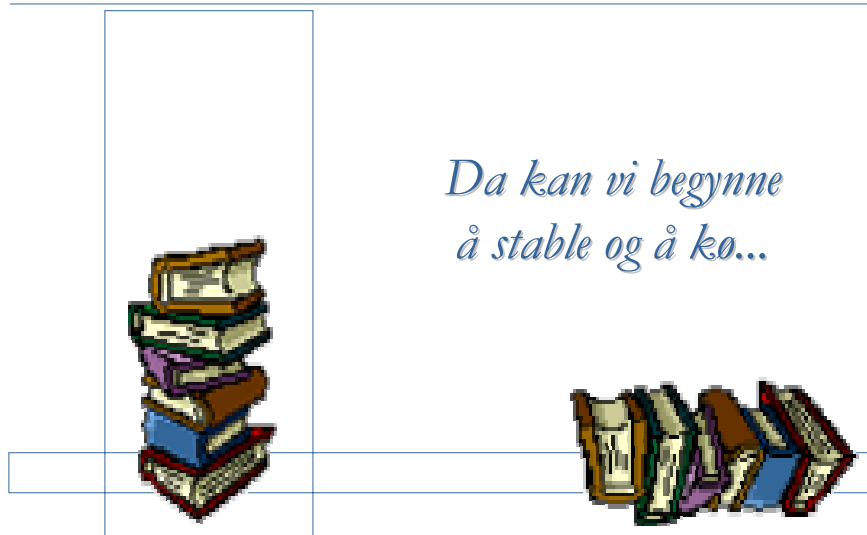
*En stabel...
er en **Last in First Out (LIFO)** struktur.*



*En kø derimot...
er en **First in First Out (FIFO)** struktur!*



*Da kan vi begynne
å stable og å kø...*



OVERSIKT – Uke 3, Forelesning 2 (W3.L2)

- Neste gang, dvs. uke 4, forelesning 1 (W4.L1):
STABLER og **KØER** avsluttes.
- Tenk litt på operasjoner som er "naturlige" for disse ADT'er
- Tenk også litt på intuitive og kloke implementasjoner av disse ADT'er (dvs. mulige datastrukturer med sine operasjoner)
- Og ikke glem å tenke på hvor effektive de respektive implementasjonene er... Prøve å regne ut den store O'en!
- Disse finner du i del 3.3 utover i MAW kapittel 3.

