Implementasjon

EN ADT AV EN ANNEN DT

Data representasjon

Data Invariant

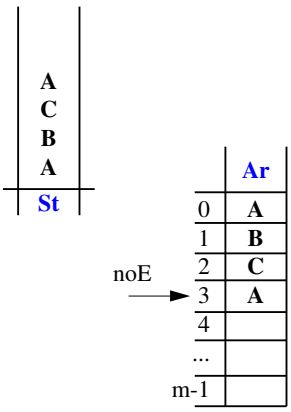
ALGORITMEANALYSE

Asymptotisk notasjon

Kap. 1-2

Implementasjon (*Stack* med array)

```
public interface Stack {
   void push(Object o);
   Object pop() throws
      EmptyStackException;
   Object peek() throws
      EmptyStackException;
   boolean empty();
...}
```



I DATA REPRESENTASJON

Hvordan skal en stabel representeres ?

- **en array med en peker tilsvarende peek()**
- **rekkefølgen er omvend rekkefølgen i Stabel**

II DATA STRUKTUR

Hvilke konkrete data trenger jeg å holde rede på ?

class Stab

– Object[] Ar

– int noE

– int max

III DATA INVARIANT

DS kan holde forskjellige "inkonsistente" verdier – de må bindes på en måte som tilsvarer Stabel og denne "binding" må opprettholdes

noE – Ar[noE] er toppen av stabel – hvis

noE+1 = antall elementer > 0

Må passe på at **noE** *ikke går forbi* **Ar.length** *!*

max = Ar.length > noE >= –1

Opprettholdelse av DataInvarianten

```
public class Stab implements Stack {
   private Object[] Ar;
   private int noE, max=10;

   public Stab() {
      Ar= new Object[max]; noE= –1;
   }                                  // er DI ok etter initialisering ?

   public void push(Object o) {       // hvis: DI ok ved kall
      if (noE=max-1) {
         Object[] temp= new Object[max];
         Copy(Ar, tab);
         max= 2*max; Ar= new Object[max];
         Copy(tab,Ar);  }
      noE++;
      Ar[noE]= o;
   }                                  //  er DI ok ved retur ?

   public Object pop()                // hvis: DI ok ved kall
      throws EmptyStackException {
      if (empty())
         throw new EmptyStackException("Tom");
      else { noE--; return Ar[noE+1]; }
   }                                  //  er DI ok ved retur ?
```

noE – Ar[noE] er toppen av stabel – hvis
noE+1 = antall elementer > 0
max = Ar.length > noE >= –1

```
   public Object peek()               // hvis: DI ok ved kall
      throws EmptyStackException {
      if (empty())
         throw new EmptyStackException("Tom");
      else return Ar[noE];
   }                                  // er resultatet ok ?

   public boolean empty() {           // hvis: DI ok ved kall
      return noE < 0;
   }                                  // er resultatet ok ?

   private void Copy(Object[] fra, Object[] til) {
                                      // hvis: DI ok ved kall
      for (int k=0; k<fra.length; k++)  til[k]= fra[k];
   }                                  // er DI ok ved retur ?
}
```


DataInvarianten kan – og bør – implementeres

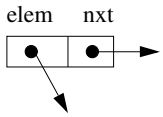
noE – Ar[noE] er toppen av stabel – hvis
noE+1 = antall elementer > 0
max = Ar.length > noE >= –1

```
public class Stab implements Stack {
   private Object[] Ar; private int noE, max=10;
   public Stab() throws DIExc {  Ar= new Object[max]; noE= –1;
    if (!DI()) throw new DIExc("Init feil");
   }
   public void push(Object o) throws DIExc {
    if (!DI()) throw new DIExc("DI feil før push");
    if (noE=max-1) {
      Object[] temp= new Object[max];
      Copy(Ar, tab);
      max= 2*max; Ar= new Object[max];
      Copy(tab,Ar);  }
    noE++;
    Ar[noE]= o;
    if (!DI()) throw new DIExc("DI feil etter push");
   }
   public Object pop()  throws DIExc,
      EmptyStackException {
    if (!DI()) throw new DIExc("DI feil før pop");
    if (empty())
      throw new EmptyStackException("Tom");
    else { noE--;
      if (!DI()) throw new DIExc("DI feil etter pop");
      else return Ar[noE+1]; }
   }
```

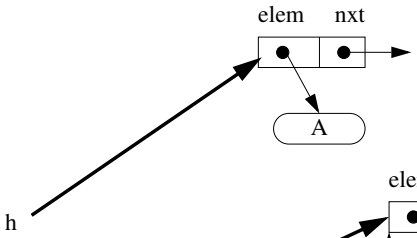
```
   protected boolean DI() {
     if (noE >= –1 && noE < max && max==Ar.length)
        return true;
     else return false;
   }
   public Object peek() throws DIExc,
        EmptyStackException {
     if (!DI()) throw new DIExc("DI feil før peek");
     if (empty())
        throw new EmptyStackException("Tom");
     else return Ar[noE];
   }
   public boolean empty() throws DIExc {
     if (!DI()) throw new DIExc("DI feil ved empty");
     return noE < 0;
   }
   private void Copy(Object[] fr, Object[] ti) throws DIExc {
     if (!DI()) throw new DIExc("DI feil");
     for (int k=0; k<fr.length; k++)  ti[k]= fr[k];
     if (!DI()) throw new DIExc("DI feil etter kopi");
   }
```

En-veis Lenket Liste - datastruktur

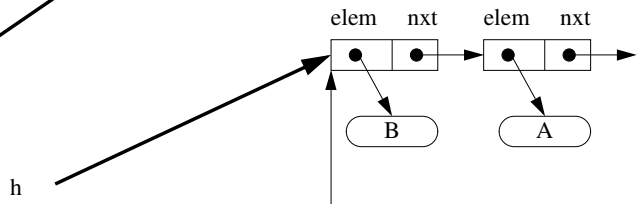
```
public class Node {
   private Object elem;
   private Node next;
   public Node(Object o, Node n) {
      elem= o; next= n;}
   public Node() { this(null, null); }
   public Object getElem() { return elem; }
   public Node getNext() { return next; }
   public void setElem(Object o) { elem= o; }
   public void setNext(Node n) { next= n; }
}
```



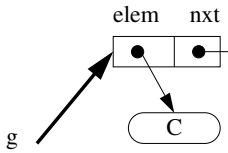
Node h= new Node(A,null);



h= new Node(B,h);

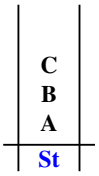


Node g= new Node(C,h)



Implementasjon (*Stack* med LenketListe)

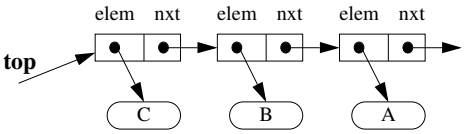
```
public interface Stack {
   void push(Object o);
   Object pop() throws
      EmptyStackException;
   Object peek() throws
      EmptyStackException;
   boolean empty();
...}
```



I Data Representasjon

Hvordan skal en stabel representeres ?

- **en klasse med en peker til top-node = peek()**
- **innsetting skjer i starten av listen**



II Data Struktur

Hvilke konkrete data trenger jeg å holde rede på ?

class StackLL

– Node top

III Data Invariant

DS kan holde forskjellige "inkonsistente" verdier – de må bindes på en måte som tilsvarer Stabel og denne "binding" må opprettholdes

top == peker alltid på toppen av stabel

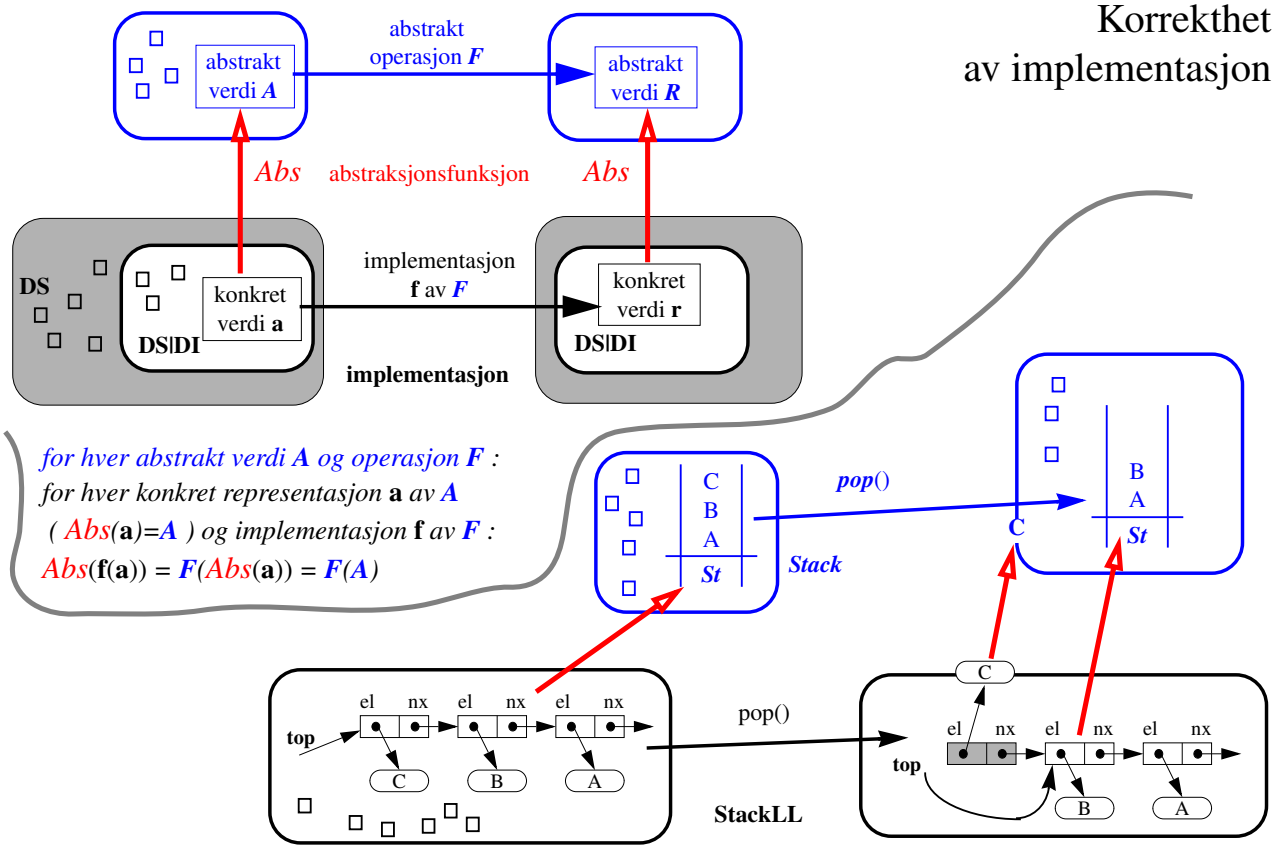
== null hviss stabel er tom

Opprettholdelse av DataInvarianten

```
public class StackLL implements Stack {
  private Node top;

  public StackLL() { top= null; }
                                  // er DI ok etter initialisering ?

  public void push(Object o) {          //hvis: DI ok ved kall
    top= new Node(o,top);
  }                                     // er DI ok ved retur ?

  public Object pop()                   // hvis: DI ok ved kall
    throws EmptyStackException{
    if (empty()) throw
      new EmptyStackException("Tom ved pop");
    else {
      Object u= top.getElem();
      top= top.getNext();
      return u;
  } }                                   // er DI ok ved retur ?

  public Object peek()  // hvis: DI ok ved kall
   throws EmptyStackException {
      if (empty()) throw
                 new EmptyStackException("Tom");
      else return top.getElem();
  }                                     // er resultatet ok ?

  public boolean empty()  {
                                  // hvis: DI ok ved kall
      return (top==null);
  }                                     // er resultatet ok ?
}
```

top == peker på toppen av stabel
== null hviss stabel er tom

Korrekthet av implementasjon



Oppsummering: implementasjon (av en (A)DT med en DT)

1. ***DataRepresentasjon:*** *hvordan kan **ADT**-elementene representeres v.hj.a. **DT**-elementene*

   – *det kan godt hende at du trenger flere konkrete DT'er, **I1**, **I2**...**In**, for å holde all nødvendig informasjon*

2. ***DataStruktur:*** *velg en måte å binde sammen de relevante aspektene fra **I1**...**In***

   – *beskriv abstraksjonsfunksjonen **Abs***
   *(denne avbilder kun de instansene av **DT** som oppfyller **DataInvarianten**)*

3. ***DataInvarianten:*** *for din DataStruktur; spesifiser*

   – *hvilke instanser av DataStruktur faktisk representerer abstrakte verdier fra **ADT***

   – *hvordan karakteriseres disse med relasjoner mellom attributter av forskjellige **I1**...**In***

4. *Implementer operasjonene fra **ADT** på den valgte DataStrukturen*

   – *implementasjonen skal oppfylle spesifikasjonen (gitt i interface ; for- og bakbetingelser)*

   – *verifiser, evt. implementer **DataInvarianten**, forsikre deg om at*

   • ***DI** holder etter initialisering*

   *og for hver implementert/konkret operasjon **f** i din **DT**:*

   • *dersom **DI** holder før **f** kalles, så holder den også etter at **f** returnerer*

5. *Se hvor **effektiv** din implementasjon er*

Effektivitet = tidskompleksitet

Tidsforbruk = *antall primitive (atomære) operasjoner,* P(_)

Primitive operasjoner:
– *sammenlikning*
– *aritmetikk (+, /, ...)*
– *aksess (A[x], P.k)*
– *tilordning,*
– *metodekall,*
– *return*
. . .

	P(op)
Stab(){ Ar= new Object[max]; noE= –1;}	**3**
void ***push***(Object o) { if (noE==max-1) {	2
Object[]temp = new Object[max];	2
Copy(Ar,tab); max= 2*max;	max+2
Ar = new Object[max];	2
Copy(tab,Ar); }	max
noE++; Ar[noE] = o;	3
}	**5 / 11+ 2*max**
Object ***pop***() { if (empty()) return null;	1+1
else { noE--; return Ar[noE+1];}	3
}	**2/4**
Object ***peek***() { if (empty()) return null;	1+1
else return Ar[noE];	2
}	**2/3**
boolean ***empty***() { return (noE < 0); }	**2**

	P(op)
StackLL() { top= null; }	**1**
void ***push***(Object o) {	
top= new Node(o,top); }	**2**
Object ***pop***() {	
if (empty()) return null;	1+1
else { Object u= top.getElem();	3
top= top.getNext(); return u;	4
} }	**2/8**
Object ***peek***() { if (empty()) return null;	1+1
else return top.getElem();	3
}	**2/4**
boolean ***empty***() {return (top==null);}	**2**

- **P**(Stab.push(**n**)) = 11+2*max > 2 = **P**(LL.push(**n**))
- *Alt i LL tar konstant – uavhengig av* **n** *– tid*
- Stab.push *avhenger av* **max** *– ikke av* **n**

Tidskompleksitet = tidsforbruk *som funksjon av inputstørrelse* **n**

En operasjon *op* er *av orden 1*, *op* = **O**(1),
hviss det finnes to konstanter *k*, *n0* s.a.
$\forall n\ (n > n0 \rightarrow \mathbf{P}(op(n)) \leq k*1)$

O(1) = **O**(5) = **O**(k)

Algoritmeanalyse

```
/*   SS - sorterer input array (SeleksjonSort)
 *      @param  - int tab[0...n]
 *      @return - sortert tab
 *
 *   for (k = 0,1,2...n) {
 *      i = k
 *      for ( j = k+1...n)
 *         if (tab[j] < tab[i]) i = j;
 *      bytt elementene ved indeks k og i
 *      }
 */
```

Hvor mange ganger må jeg utføre basis operasjoner ?
sammnlikn. + flytting

k= 0	2	4	1	3	5	4 + 1
k= 1	1	4	2	3	5	3 + 1
k= 2	1	2	4	3	5	2 + 1
k= 3	1	2	3	4	5	1 + 1
k= 4	1	2	3	4	5	0 + 0
						10 + 4 = 14

for en vlikårlig input tabell med lengde **n**:

- *utfører* **n** *iterasjoner (for k=1,2...***n***) og*
- *i hver iterasjon går gjennom sluttsegment* **[k...n]**, *(for j=k+1...n), dvs.*

tidskompleksistet $SS(n) = \left(\sum_{k=1}^{n} k = 1 + 2 + \ldots + (n-1) + n \right) = (n + n^2)/2 = \mathbf{O}(n^2)$

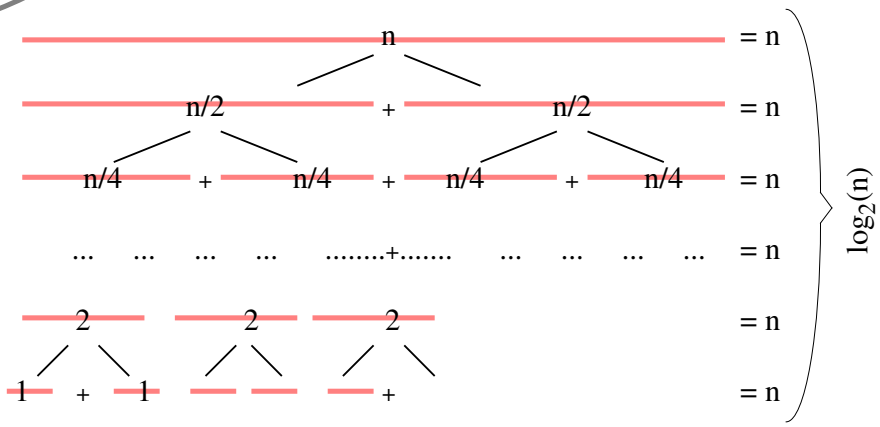
MergeSort

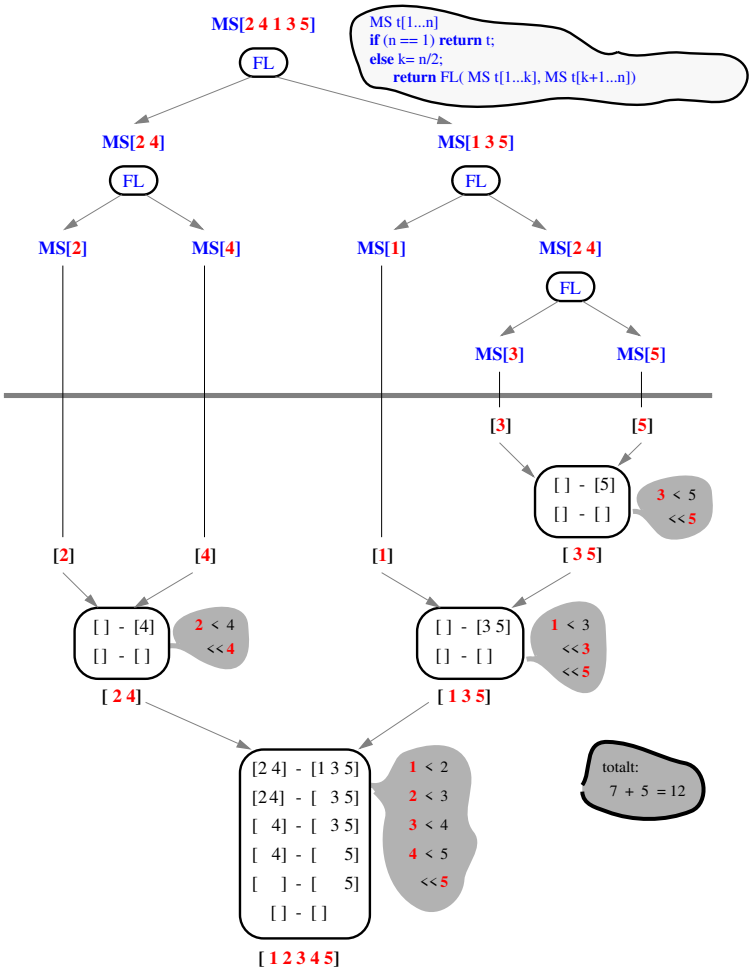
```
/*   FL - fletter to sorterte array:
 *      @param  - int t1[0...n1], t2[0...n2] - sorterte
 *      @return - sortert t[0.....n1+n2]
 *   gå (samtidig) gjennom t1 og t2 (med i1 og i2)
 *   if      t1[i1] < t2[i2] plasser t1[i1] i t og øk i1, i
 *   else                    plasser t2[i2] i t og øk i2, i
 *   hvis noe igjen i t1 eller t2, flytt det til t
 *   return t;
 */
```

$FL(n1,n2) = O\ (n1+n2)$

```
/*   MS - sorterer input array:
 *      @param  - int tab[0...n-1]
 *      @return - sortert tab
 *   if (n == 1) return tab
 *   else { k= n/2;
 *      return FL ( MS(tab[0...k]), MS(tab[k+1..n-1]) ); }
 */
```

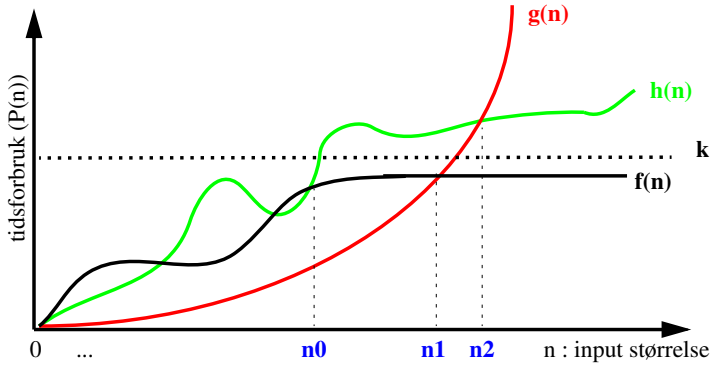
$MS(n) = O\ (n * \log_2(n)\ )$





Asymptotisk notasjon: f(n) = **O**(g(n))

En funksjon f(n) *er av orden* g(n), f(n)= O(g(n)), *hviss*
det finnes konstanter **c** *og* **n0** s.a. ∀n (n > **n0** → f(n) ≤ **c** * g(n))



for alle n > **n1** : **f(n)** ≤ **1** * **g(n)** :
f(n) = **O**(g(n))

for alle n > **n0** : **f(n)** ≤ **1** * **h(n)** :
f(n) = **O**(h(n))

for alle n > **0** : **f(n)** ≤ **k** * **h(n)** :
f(n) = **O**(h(n))

for alle n > **0** : **f(n)** ≤ **1** * **k** :
f(n) = **O**(k)

for enhver konstant **c** >= 1:

for alle n > **0** : f(n) ≤ **k** * **c** – f(n) = **O**(**c**)

for alle n > **0** : f(n) ≤ **k** * **1** – f(n) = **O**(**1**) **!!!**

for alle n > **n2** : h(n) ≤ **1** * **g(n)** – h(n) = **O**(**g(n)**)

O-notasjon

Gitt heltallsfunksjoner f, g, h : N → N

- $f(n) = \boldsymbol{O}(\, g(n)\,)$ det finnes c, n0: n > n0 ! $f(n) \le c * g(n)$ $f \le_{\boldsymbol{O}} g$
  - $\boldsymbol{o}(\, g(n)\,)$ $f <_{\boldsymbol{O}} g$
- $f(n) = \Omega(\, g(n)\,)$ hvis $g(n) = \boldsymbol{O}(\, f(n)\,)$ $f \ge_{\boldsymbol{O}} g$
  - $\omega(\, g(n)\,)$ $f >_{\boldsymbol{O}} g$
- $f(n) = \Theta(\, g(n)\,)$ hvis $f(n) = \boldsymbol{O}(\, g(n)\,)$ og $g(n) = \boldsymbol{O}(\, f(n)\,)$ $f =_{\boldsymbol{O}} g$

$\boldsymbol{O}$ forekommer ikke på venstre side av =

- Hvis $f(n) = \boldsymbol{O}(\, h(n)\,)$ og $h(n) = \boldsymbol{O}(\, g(n)\,)$ så også $f(n) = \boldsymbol{O}(\, g(n)\,)$

  $f(n) = \boldsymbol{O}(\, h(n)\,)$

  $h(n) = \boldsymbol{O}(\, g(n)\,)$

  $f(n) = \boldsymbol{O}(\, g(n)\,)$
- $f(n) + g(n) = \boldsymbol{O}(\, \max\{\, f(n), g(n)\,\}\,)$
- Hvis $f(n) = \boldsymbol{O}(\, h(n)\,)$ så $f(n) + g(n) = \boldsymbol{O}(\, h(n) + g(n)\,)$
- Hvis $f(n) = \boldsymbol{O}(\, h(n)\,)$ så $f(n) * g(n) = \boldsymbol{O}(\, h(n) * g(n)\,)$
- for en gitt c: $\log(n^c) = \boldsymbol{O}(\, \log(n)\,)$

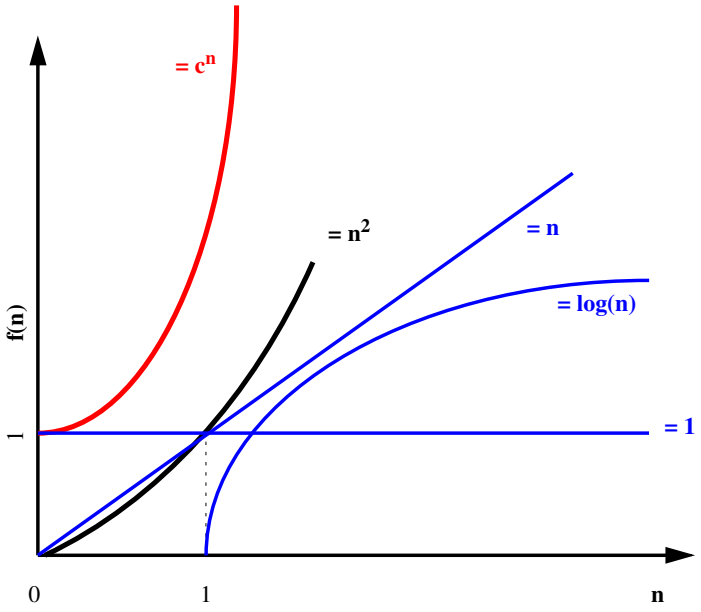
En tommelfinger regel:

Glem lavereordens termer og konstanter:

$$f(n) = 2 * \mathbf{n^3} + 16 * n^2 + 5000$$
$$= \boldsymbol{O}(\, n^3\,)$$

$$f(n) = 8 * \mathbf{n^2 * \log(n)} + 3*n$$
$$= \boldsymbol{O}(\, n^2 * \log(n)\,)$$

Mest vanlige kompleksitets-klasser



konstant $\boldsymbol{O}(\, 1\,)$

$1 =_{\boldsymbol{O}} 5 =_{\boldsymbol{O}} 100 =_{\boldsymbol{O}} 2^{100} =_{\boldsymbol{O}} \ldots <_{\boldsymbol{O}}$

logaritmisk $\boldsymbol{O}(\, \log(n)\,)$

$5 * \log(n) =_{\boldsymbol{O}} 2^{100} * \log(n) =_{\boldsymbol{O}} \ldots <_{\boldsymbol{O}}$

lineær $\boldsymbol{O}(\, n\,)$

$5 * n =_{\boldsymbol{O}} 7 * n - 3 =_{\boldsymbol{O}} n + \log(n) =_{\boldsymbol{O}} \ldots <_{\boldsymbol{O}}$

kvadratisk $\boldsymbol{O}(\, \mathbf{n^2}\,)$

$5 * n^2 =_{\boldsymbol{O}} 125 * n^2 + 5 * n - 3 =_{\boldsymbol{O}} \ldots <_{\boldsymbol{O}}$

polynomisk $\boldsymbol{O}(\, \mathbf{n^k}\,)$ **k >1**

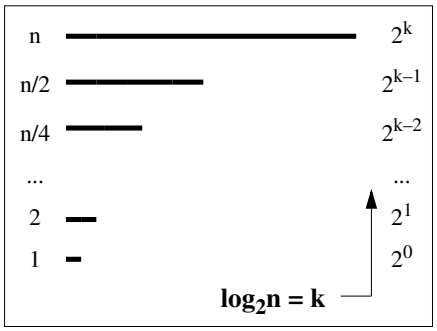
$5 * n^3 + 5 * n^2 + 5 * n + 3 <_{\boldsymbol{O}} n^4 <_{\boldsymbol{O}} n^6 <_{\boldsymbol{O}} ..$

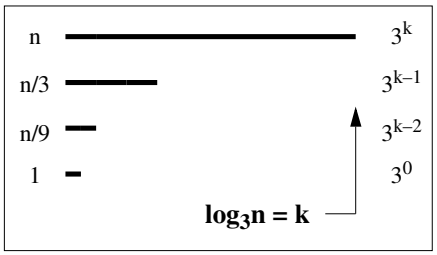
eksponensielt $\boldsymbol{O}(\, \mathbf{k^n}\,)$ **k >1**

$5 * 2^n <_{\boldsymbol{O}} 3^n =_{\boldsymbol{O}} 3^n + n <_{\boldsymbol{O}} 5^n <_{\boldsymbol{O}} \ldots$

Noen enkle fakta

$\log_b n = k$ hviss $b^k = n$





$$\log_b n = \frac{\log_c n}{\log_c b}$$

$$\log_3 n = \frac{\log_2 n}{\log_2 3} \quad =_O$$

$$\log(n * m) = \log(n) + \log(m)$$

$$\log(n^c) = c * \log(n) \quad =_O$$

$$\log(n/m) = \log(n) - \log(m)$$

$$1 + 2 + 3 + \ldots + n = \sum_{k=1}^{n} k = \frac{n^2 + n}{2}$$

$$a^0 + a^1 + \ldots + a^n = \sum_{k=0}^{n} a^k = \frac{1 - a^{n+1}}{1 - a} \qquad 0 < a \neq 1$$

Er det noen vits...???

$5 =_O 2^{100}$

men er en algoritme som utfører 5 operasjoner like god som en som bruker 2^{100} .....?

Nei – i praksis burde man ta hensyn til slike konstanter skjult bak $O(_)$ – cf. Oblig.1

Vi har jo stadig raskere maskiner ...

$O(\ )$	faktisk	problemstørrelse – løses på 1 sekund		
		dagens maskin	60-gngr raskere	3600-gngr raskere
n	400 * n	2 500	150 000	9 000 000
n * log(n)	20*n*log(n)	4 000	170 000	8 000 000
n^2	$2 * n^2$	700	5 500	43 000
n^4	n^4	30	90	250
2^n	2^n	20	25	30

3600-ganger raskere maskin vil løse et problem 9.000.000-stor dersom i dag kan den løse et problem 2.500 – dersom vi har en linær algoritme !!!

Men er algoritmen eksponensielt, vil dagens maskin kunne løse et problem 20-stor, mens en 3600-ganger raskere maskin vil, i samme tid, kunne løse bare et 30-stort problem !!!

Typisk ja, men i Verste Fall ...

Finn indeks til et element x i en array A[0...n]

```
/* int finn(x)
 *      i= 0; funnet= false;
 *      while (!funnet) {
 *          if  (Ar[i] == x) r = i; funnet = true;
 *          else i++; }
 *      if (funnet) return r
 *      else return –1;
 */
```

	Ar
0	**A**
1	**B**
2	**C**
3	**Z**
4	**V**
5	**Y**
n= 6	**X**

finn(A) – 1

i beste fall : 1 ***O*** (1)

finn(V) – 5

gjennomsnittlig : n/2 ***O*** (n)

finn(X) – 7

i verste fall : n ***O*** (n)

O-notasjon brukes nesten utelukkende for verste-fall analyse

Oppsummering

Implementasjon av (A)DT

- *Data Representasjon*
- *Data Struktur*
- *Data Invariant*
- *Korrekthet*
  - *opprettholdelse av Data Invarianten*
  - *korrekt implementasjon av abstrakte operasjoner*

Effektivitet:

- *tidskompleksitet = hvordan avhenger tidsforbruket av størrelsen på input*
- *kompleksitets-klasser*
  - *lineær*
  - *logaritmisk*
  - *polynomisk (kvadratisk)*
  - *eksponensiell*
- *verste-fall analyse*