Implementasjon

I. En ADT av en annen DT

I.1 Data representasjon

I.2 Data Invariant

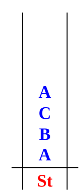
II. AlgoritmeAnalyse

II.1 Asymptotisk notasjon

Kap. 1-2

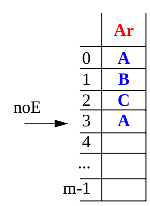
Implementasjon

(Stabel med array)




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

Skal bruke array ....



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 ?

Object[] Ar

int noE

int max

III. Data Invariant

DS kan være hva som helst – den 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: max > noE >= -1

Opprettholdelse av DI

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

```
public class Stab
  IMPLEMENTS STACK {

  private Object[] Ar;
  private int noE, max=10;

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

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

oppdaterer DS

```
  public Object peek() throws EmptyStackException {     // hvis: DI ok ved kall
     if (empty()) throw new EmptyStackException("Tom ved peek");
     else return Ar[noE];                               // ? er det ok resultat
  }
  public boolean empty() { return noE < 0; }            //...
  private void Copy(Object[] fra, Object[] til) {       //...
     for (int k=0; k<fra.length; k++)  til[k]= fra[k]; }
}
```

observerer DS

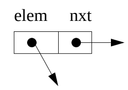
DI kan - og bør - implementeres !

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

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

  public Stab()  throws DIException
  { Ar= new Object[max]; noE= -1;
     if (! DI()) throw DIException("Feil initialisering");}

  public void push(Object o)  throws DIException
  {  if (! DI())
        throw DIException("DI holder ikke ved kall (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 DIException("DI holder ikke ved utgang (push)");
  }

  public Object pop()  throws EmptyStackException, DIException
  {  if (! DI())
        throw DIException("DI holder ikke ved kall (pop)");
     if (empty()) throw new EmptyStackException("Tom ved pop");
     else {      noE--;
                 if (! DI())
                   throw DIException("DI holder ikke ved utgang (pop)");
                 return Ar[noE+1]; }
  }

  private boolean DI()
  {  if (noE < -1 || noE >= max) return false;
     else return true; }
```

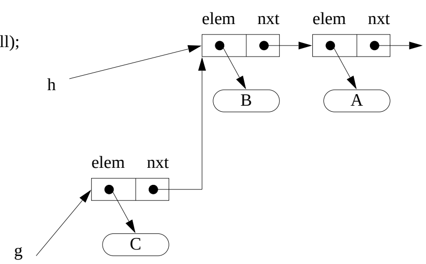
Linket Liste DT

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



Node h= new Node(A,null);
h= new Node(B,h);

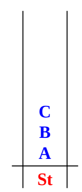
Node g= new Node(C,h)



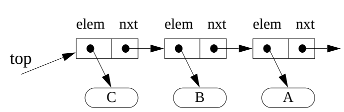
Implementasjon

(Stabel med Linket Liste)

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



Skal bruke Linket Liste ....



I. Data Representasjon

Hvordan skal en stabel representeres ?

en klasse med en peker til en top Node

– innsetting skjer ved starten av Liste

II. Data Struktur

Hvilke konkrete data trenger jeg å holde rede på ?

Class LStab med

Node top

III. Data Invariant

DS kan være hva som helst – den må bindes på en måte som tilsvarer Stabel og denne "binding" må opprettholdes

top – peker alltid på toppen av stabel

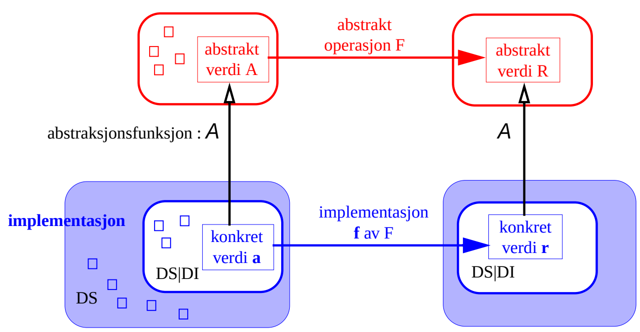
== null hviss stabel er tom

Opprettholdelse av DI

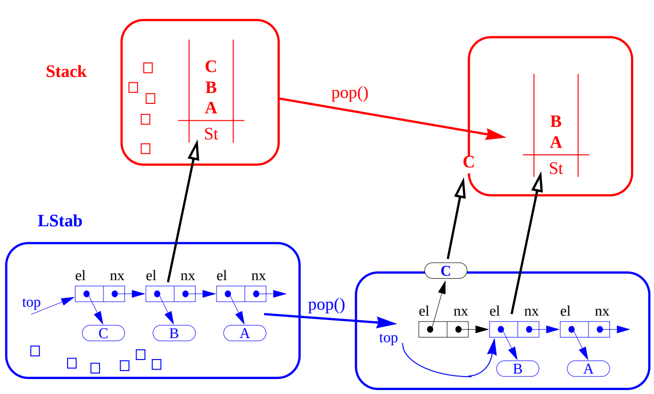
```
public class LStab
  IMPLEMENTS STACK {

  private Node top;

  public LStab() { 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 return
  }

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

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

  public boolean empty()
     { return (top==null); }
}
```

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

Korrekthet av implementasjon



For hver abstrakt verdi A og operasjon F:
for hver konkret representasjon a av A : $A(\mathbf{f}(\mathbf{a}))$ = F(A(**a**)) = F(A)



Oppsummering: implementasjon

av en ADT S med en annen DT I

1. **bestem hvordan strukturen av S kan representeres v.hj.a. I,**
   - **det kan godt hende at du trenger flere forskjellige DT'er I1...In for å holde all nødvendig informasjon**
2. **velg en konkret Data Struktur som binder sammen de relevante aspektene fra I1...In**
   - **beskriv abstraksjonsfunksjonen** *A*
3. **spesifiser Data Invariant for DS**
   - **som beskriver lovlige sammensetting av I1...In (dvs. slike som faktisk representerer noen S)**
4. **implementer operasjonene fra S på den valgte DS: verifiser/implementer DI, dvs. at**
   - **DI gjelder etter initialisering og**
   - **for hver konkret operasjon op, hvis DI holder før kallet, så holder den etter at op returnerer**
5. **se hvor effektiv din implementasjon er**

Effektivitet = tidskompleksitet

Tidsforbruk = antall **P**rimitive (atomære) operasjoner

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

```
LS()                                   P(op):
  { top= null; }                          1
void push(Object o) {
  top= new Node(o,top);                   2
}
Object pop() {
  if (empty()) return null;               2
  else {
    Object u= top.getElem();              2
    top= top.getNext();                   2
    return u;                             1
  }                                     2/6
}
Object peek() {
  if (empty()) return null;               2
  else return top.getElem();              2
}                                       2/3
boolean empty()
  { return (top==null); }                 2
```

```
Stab()                                 P(op):
 { A= new Object[max]; noE= -1;}          3
void push(Object o) {
 if (noE=max-1) {                         1
   Object[]temp= new Object[max];         2
   Copy(A,tab);                         max
   max= 2*max;                            2
   A= new Object[max];                    2
   Copy(tab,A);  }                    2*max
 noE++; A[noE]= o;                        3
}                               4 /+ 3*max+6
Object pop() {
 if (empty()) return null;                2
 else { noE--;return Ar[noE+1];}          3
}                                       2/4
Object peek() {
 if (empty()) return null;                2
 else return A[noE];                      2
}                                       2/3
boolean empty()
 { return (noE < 0); }                    2
```

- **P**(LS.push(**n**)) = 2 < 10+3*max = **P**(S.push(**n**))
- Alt i LS tar konstant - uavhengig av **n** - tid
  P(pop(**n**))=2 eller =6
- og Stab.push avhenger av **max** – ikke av **n**

Tidsforbruk som funksjon av InputStørrelsen n !

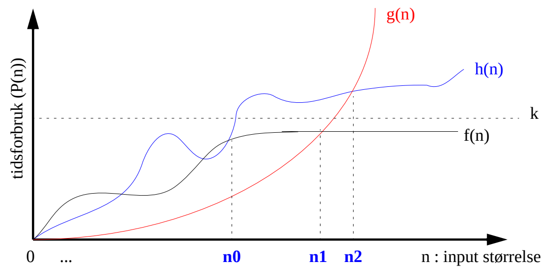
konstanter – som ikke avhenger av input størrelse – ignoreres!!

En operasjon op **er av orden** 1, op = O(1), **hviss**
det finnes konstanter c og n0 : n>n0 → **P(**op(**n**)**)** ≤ **c*1**

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

Asymptotisk notasjon: O(g(n))

En funksjon f(n) **er av orden** g(n), f(n) = O(g(n)), **hviss det finnes konstanter c** og **n0** : 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) = O(g(n)) det finnes c, n0: n > n0 ! f(n) ≤ c * g(n) f ≤ g
  o(g(n)) f < g
- f(n) = Ω(g(n)) hvis g(n) = O(f(n)) f ≥ g
  ω(g(n)) f > g
- f(n) = Θ(g(n)) hvis f(n) = O(g(n)) og g(n) = O(f(n)) f = g

Dette er ikke vanlig likhet :
O, o, Ω, ω, Θ kan ikke forekomme på venstre side av =

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

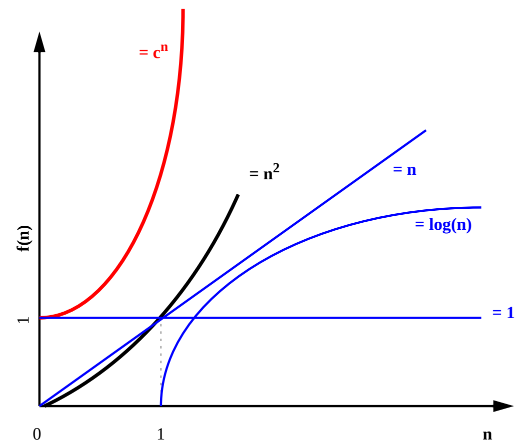
En tommelfinger regel:

Glem lavereordens termer og konstanter:

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

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

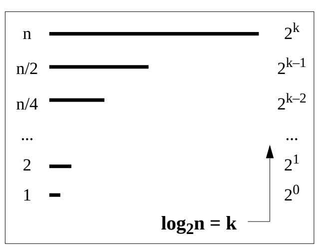
Mest vanlige klasser

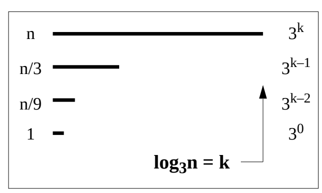


konstant	$O(1)$	$1 =^O 5 =^O 100 =^O 2^{100} =^O \dots <^O$
logaritmisk	$O(\log(n))$	$5 * \log(n) =^O 2^{100} * \log(n) =^O \dots <^O$
linær	$O(n)$	$5 * n =^O 7 * n - 3 =^O n + \log(n) =^O \dots <^O$
kvadratisk	$O(\mathbf{n^2})$	$5 * n^2 =^O 125 * n^2 + 5 * n - 3 =^O \dots <^O$
polynomisk	$O(\mathbf{n^k})$ **k>1**	$5 * n^3 + 5 * n^2 + 5 * n + 3 <^O n^4 <^O n^6 <^O \dots$
eksponensielt	$O(\mathbf{k^n})$ **k>1**	$5 * 2^n <^O 3^n =^O 3^n + n <^O 5^n <^O \dots$

Noen enkle fakta

$\mathbf{\log_b n = k}$ **hviss** $\mathbf{b^k = n}$





$$\mathbf{\log_b n = \log_c n / \log_c b}$$

$$\log_3 n = \log_2 n / \log_2 10 \qquad =^O$$

$$\mathbf{\log(n * m) = \log(n) + \log(m)}$$

$$\log(n^c) = c * \log(n) \qquad =^O$$

$$\log(n/m) = \log(n) - \log(m) \qquad =^O$$

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

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

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

dvs.
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 !!!

Seleksjon-Sort

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

k= 1	2	4	<u>1</u>	3	5	n-1
k= 2	1	4	<u>2</u>	3	5	n–2
k= 3	1	2	4	<u>3</u>	5	n–3
k= 4	1	2	3	<u>4</u>	5	...
k= 5	1	2	3	4	5	1

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], dvs.

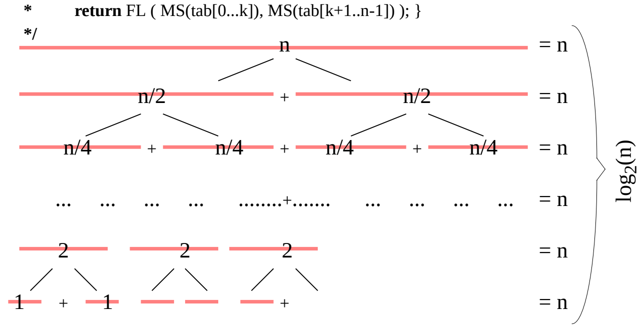
$$SS(n) = O\left(\sum_{k=1}^{n-1} k\right) = O\left(\frac{(n-1)\times n}{2}\right) = O(n^2)$$

Merge-Sort

```
/*    FL - fletter to sorterte array:
 *        @param  - int t1[0...n1], t2[0...n2] - sorterte
 *        @return - sortert t[0......n1+n2]
 * la i1 gåfra 0...n1 og i2 fra 0...n2
 * if     t1[i1] < t2[i2]  t[i]= t1[i1]; i1++; i++;
 * else                    t[i]= t2[i2]; 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]) ); }
 */
```



$$MS(n) = O\,(n * \log(n))$$

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 $O\,(n)$

Vi bruker O-notasjon nesten utelukkende for verste-fall analyse