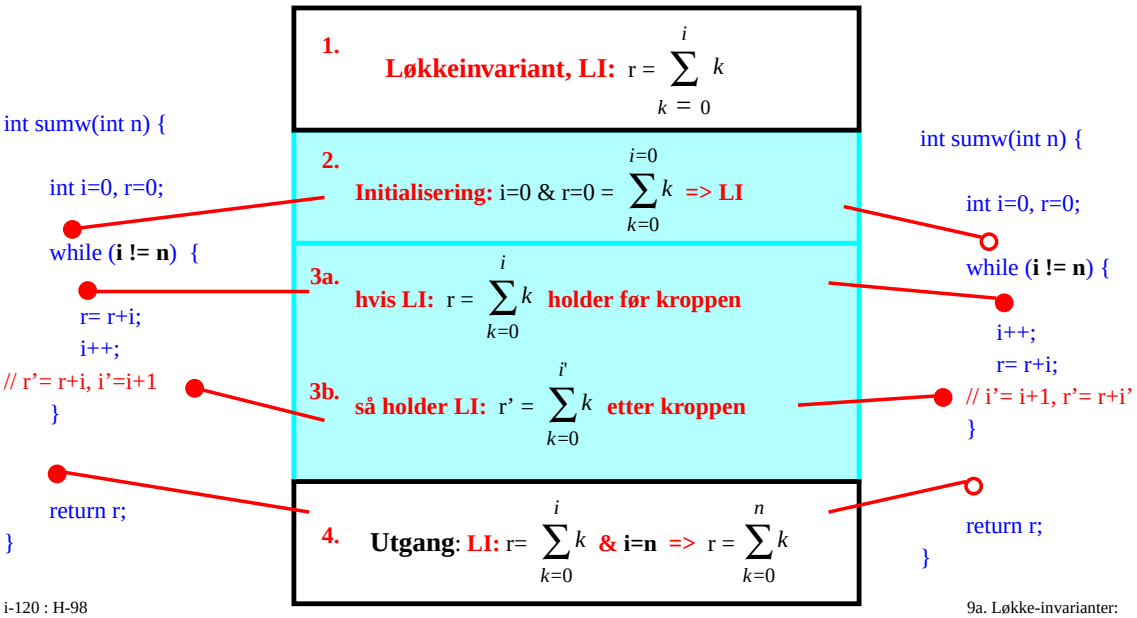
Løkkeinvariant

```
int sum(int n) {
    if (n == 0) return 0;
    else return n + sum(n–1);
}
```

basis: n==0 – sum(0) = 0

hvis sum(n–1) returnerer riktig = $\sum_{i=0}^{n-1} i$

så er sum(n) = n + sum(n–1) = $\sum_{i=0}^{n} i$



Bobblesortering

/ sorterer array A[0...n]:**
PRE @param A array av int @return sortert A */

```
BS(int[] A)
  // A[n+1..n] ...        0 <= r < s <= n & s > i=n ... ingen slik s
1: for (i = n, i >= 1, i--) {
     li1: A[i+1...n] er sortert og har n-i største elementer
     LI1:   0 <= r < s <= n & s > i → A[r] <= A[s]
     // A[0] er størst i A[0...0] ...   0 <= t < j=0 ... ingen slik t
2:   for (j = 0, j < i, j++) {
       li2: A[j] er størst i A[0...j]
       LI2:  0 <= t < j <= n → A[t] <= A[j]
       if (A[j] > A[j+1])  swap(A[j], A[j+1]);
         j'=j+1: er A[j'] størst i A[0...j']
         hvis A[j] <= A[j+1] & li2 →  YES
         ellers A[j'] = A[j] > A[j+1] = A[j'–1] & li2 → YES
       LI2': 0 <= t < j' <= n → A[t] <= A[j']
     }  UT2:  li2 & j=i : A[i] er størst bland A[0...i]
              LI2 & j=i: 0 <= t < i <= n → A[t] <= A[i]
     li1 & UT2:  A[0...i–1] <= A[i] <= A[i+1...n]  → li1' der i'=i–1
       li1': A[i...n] er sortert og har n-(i–1) største elementer
     LI1': i'= i–1: 1 <= r < s <= n, s > i' → A[r] <= A[s]
   }  UT1:  li1 & i=0 :  A[1...n] er sortert og har n-1 største elem.
            men da også A[0] har et element <= A[r] for 0 < r <= n
            LI1 &  0=i : 0 <= r < s <= n & s > 0 → A[r] <= A[s]
            spesielt, for s=1 og r=0 : A[0] <= A[1]
```

Heltallsdivisjon

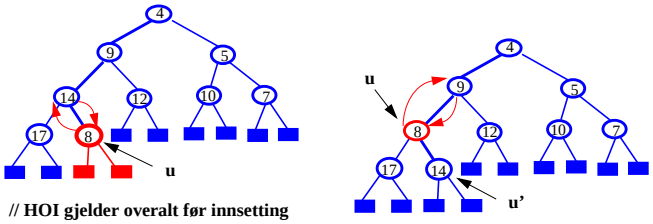
```
/**   beregner heltalls kvosient samt resten
PRE @param x >= 0
      @param y > 0
      @return (q, r) sa. x= q*y + r & r < y */
divr(x, y)  {
   q = 0 ; r = x ;
// q=0=k & r= x >= 0 & x= x + 0*y
   while (y <= r)  {
etter k-te iterasjon: q=k & x = r + k*y
LI:   r >= 0 & x = r + q*y
      q = q+1 ;
      r = r – y;
   q' = q+1 & r'= r–y >= 0 &
   r' + q'*y = (r–y) + (q+1)*y
             = r + q*y – y + y = x
   }
UT:  LI & r<y => x = r + q*y & r < y
   return (q, r) ;
}
```

```
/**   beregner største felles divisor
PRE @param x1 > 0
      @param x2 > 0
      @return y2 = gcd(x1,x2) */
gcd(x1,x2) {
   y1= x1; y2= x2;
// gcd(x1,x2) == gcd(x1,x2)
   while (y1 != 0) {
LI:   gcd(y1,y2) == gcd(x1,x2)
      if (y2 >= y1) y2= y2–y1;
//        gcd(y2,y1) == gcd(y2–y1,y1)
      else (y1,y2) = (y2,y1);
//        gcd(y2,y1) == gcd(y1,y2)
   }
UT:  LI & gcd(y1,y2) == gcd(0,y2) == y2
   return y2;
}
```

gcd(y1,y2) = z >=1
y1 = z*k1 <= z*k2 = y2 & gcd(k1,k2) = 1 :
y2'= y2–y1 = z*(k2-k1) & gcd(k1, k2-k1) = 1
hvis ikke, da for noen v > 1
  k1= v*a & k2-k1= v*b, så
  k2= v*b+v*a = v*(b+a)
dvs. gcd(k1,k2) = v >1

Oppover bobbling i Heap

Heap-Ordering invariant (HOI) :
for hver node n (unntatt roten) : n.elem >= parent(n).elem




```
// HOI gjelder overalt før innsetting
u = nylig innsatt node
// nå:  1. u ligger 'nederst',  2. andre er som før
while (u != root() && (u.elem < parent(u).elem )  {
LI:   HOI kan bli ugyldig kun på stien S fra u til roten
      1. u og parent(u) er eneste paret der HOI muligens ikke holder &
      2. for alle nøkler x, y (unntatt u's): hvis 'x er over y' (etter løkken)
         så også 'x var over y' før innsetting
         swap(u , parent(u));
         u = parent(u);
//    etter 'swap' har vi
//       u' = den gamle u med dens gamle parent(u).elem
//       og u = parent'(u') = den gamle parent(u) med den gamle u.elem
// 1. –   men da: u'.elem = parent(u).elem > u.elem = parent'(u').elem
//    –   og videre: la b være det andre barnet av den gamle parent(u),
//        b.elem >= parent(u).elem > u.elem = parent'(b).elem
//    dvs. HOI holder for u' og dens 2 barn,
// 2. –  hvis u' er over en x, så var u (og dermed også parent(u)) over x før 'swap'
//       men 2. holdt før løkken og dermed opprettholdes den
//    => 2.' & 1.'
}
UT: LI & (u==root || u.elem >= parent(u).elem)  => HOI for hele Heap
```