Strukturen

```
signature ISET = sig
   type set
   val set : int list -> set
   val union : set -> set -> set
   val subset : set -> set -> bool
end

structure ISet :> ISET = struct
   type set = int list
   fun set cs = xs
   fun makeset xs = xs
   fun union xs ys = xs @ ys
   fun elem ys x = List.exists (fn y=>y=x) ys
   fun subset xs ys = List.all (elem ys) xs
end
```

Binäre Suche

lineare Suche

```
fun position compare xs x =
let
   fun position'(n,[]) = NONE
     | position'(n,x::xr) =
     if compare (x,c) = EQUAL
     then (SOME n)
     else position'(n+1, xr)
in
   position'(0,xs)
end
```

binäre Suche

```
fun position compare v x = let
   fun position'(l,r) =
     if (l>r) then NONE
     else let
       val m = (l+r) div 2
       val y = Vector.sub(v,m)
     in case compare(c,y) of
        EQUAL => SOME m
      | LESS => position (1,m-1)
      | GREATER => position(m+1,r)
     end
in position' (0,Vector.length v-1)
end
```

Schlangen

- **Signatur:**

```
signature QUEUE = sig
   type 'a queue
   val empty : 'a queue
   val snoc : 'a queue -> 'a -> 'a queue
   val head : 'a queue -> 'a
   val tail : 'a queue -> a' queue
end
```

- **Modellimplementierung:**

```
structure queue :> QUEUE = struct
   type 'a queue = 'a list
   val empty = nil
   fun snoc q x = q @ [x]
   val head = hd
   val tail = tl
end
```

Schnelle Schlangen

```
structure FQueue :> QUEUE = struct

   type 'a queue = 'a list * 'a list

   val empty = ([],[])

   fun snoc ([],_) x = ([x],[])
     | snoc (q,e) x = (q, x::r)

   val head q r = hd q

   val tail ([x],r) = (rev r, [])
     | tail (q,e) = (tl q, r)

end
```