```
signature X_ARRAY =
sig
    type 'a array
    val all       : ('a -> bool) -> 'a array -> bool
    val append    : 'a array * 'a array -> 'a array (* O(1) *)
    val drop      : 'a array * int -> 'a array
    val find      : ('a -> bool) -> 'a array -> 'a option
    val rev       : 'a array -> 'a array
    val size      : 'a array -> int
    val sub       : 'a array * int -> 'a
    val tabulate  : int * (int -> 'a) -> 'a array
    val toList    : 'a array -> 'a list
    val update    : 'a array * int * 'a -> 'a array
end

structure XArray' (* : X_ARRAY *) =
struct
    datatype 'a array =
             E
           | L of 'a
           | T of {size:int, left:'a array, right:'a array}

    fun size E = 0
      | size (L _) = 1
      | size (T x) = #size x

    fun sub (L x,0) = x
      | sub (a as T t,n)
        = if size a <= n
          then raise Subscript
          else if size (#left t) <= n
          then sub(#right t,n-size (#left t))
          else sub(#left t,n)
      | sub _ = raise Subscript

    fun update (E,_,_) = raise Subscript
      | update (L x,0,y) = L y
      | update (L x,_,_) = raise Subscript
      | update (T{left=l,right=r,size=s},n,y)
        = if n>=s
          then raise Subscript
          else if n>= size l
          then T{left=l,size=s,right=update(r,n-size l,y)}
          else T{left=update(l,n,y),right=r,size=s}

    fun toList E = []
      | toList (L x) = [x]
      | toList other = let
            fun loop (nil,answer) = answer
              (* the following case should never happen *)
              | loop (E::rest,answer) = loop(rest,answer)
              | loop (L x::rest,answer) = loop(rest,x::answer)
              | loop (T{left=l,right=r,...}::rest,answer)
                = loop(r::l::rest,answer)
        in loop([other],[]) end

    fun constant (0,x) = E
      | constant (1,x) = L x
      | constant (n,x) = let
            fun constant2 (0,x) = (constant (0,x), constant (1,x))
              | constant2 (1,x) = let
                    val m = constant (1,x)
                in (m,T{left=m,right=m,size=2}) end
              | constant2 (n,x) = let
                    val (left,right) = constant2(n div 2,x)
                    val mid = if n mod 2 = 0 then left else right
                in (T{size=n,left=left,right=mid},
                    T{size=n+1,left=mid,right=right})
                end
        in (#1 o constant2) (n,x)
        end

    fun tabulate (0,f) = E
      | tabulate (1,f) = (L o f) (0)
      | tabulate (n,f) = let
            val m = n div 2
            val f' = f o (fn i=>i+m)
        in T{size=n,
             left=tabulate(m,f),
             right=tabulate(n-m,f')}
        end

    fun all f E = true
      | all f (L x) = f x
      | all f (T{left=l,right=r,...}) = all f l andalso all f r

    fun append (E,x) = x
      | append (x,E) = x
      | append (x,y) = T{left=x,right=y,size=size x + size y}

    fun find p E = NONE
      | find p (L x) = if p x then SOME x else NONE
      | find p (T{left=l,right=r,...}) =
        case find p l
         of NONE => find p r
          | other => other

    fun rev E = E
      | rev (L x) = L x
      | rev (T x) = T{left=rev (#right x),
                      right=rev (#left x),
                      size= #size x}

    fun drop (a,n) = let
        val s = size a
        (* drop' uses the invariant that 0 < n < size a *)
        fun drop' (T{left=l,right=r,size=s},n) = let
            val sl = size l
        in  case Int.compare (sl,n)
              of LESS    => append (drop'(l,n),   r)
               | EQUAL   => r
               | GREATER => drop' (r,n-sl)
        end
          | drop' _ = raise Fail "bad Invariant"
    in if n<=0 then a else if n>= s then E else drop' (a,n)
    end


    fun hd E = raise Subscript
      | hd (L x) = x
      | hd (T s) = hd (#left s)
end

structure XArray : X_ARRAY = XArray' ;
```