List of Pairs

```
fun zip ([], []) = []
  | zip (h1::t1,h2::t2) =
            (h1,h2)::zip(t1,t2);
```

```
! Warning: pattern matching is not
           exhaustive
```

```
> val zip = fn :
     'a list * 'b list -> ('a * 'b) list
```

Creates a list of pairs from a pair of lists.

What happens when the two lists are of different length?

Unzipping

```
fun unzip []                 = ([],[])
  | unzip ((x,y)::pairs) =
       let val (t,u) = unzip pairs in
         (x::t, y::u)
         end;
```

Note the local declaration

`let` D `in` E `end`

Compare this against applying functions `first` and `second` to extract the components of the pair.

Equality Types

We can test certain expressions for equality:

```
- 2 = 1+1;
> val it = true : bool
- 1.414*1.414 = 2.0;
> val it = false : bool
- [] = [1];
> val it = false : bool
```

Equality testing can be used with the basic types, and with tuples and lists, *but not with functions.*

```
- (fn x => x+2) = (fn x => 2+x);

! Type clash: match rule of type
!    'a -> 'b
! cannot have equality type ''c
```

Testing for Membership

```
fun member (x, [])    = false
  | member (x, h::t) =
       (x=h) orelse member (x,t);
```

```
val member = fn : ''a * ''a list -> bool
```

''a is an *equality type variable*.

```
- op=;
> val it = fn : ''a * ''a -> bool
```

```
fun inter ([], l)  = []
  | inter (h::t,l) =
        if member (h,l) then h::inter(t,l)
                        else inter(t,l);
```

```
fn : ''a list * ''a list -> ''a list
```

Insertion Sort

```
fun insert(x:real, []) = [x]
  | insert(x, h::t)    =
        if x<= h then x::h::t
                 else h::insert(x,t);
```

```
fun insort []      = []
  | insort (h::t) = insert (h, insort t);
```

```
  fn : real list -> real list
```

Insertion sort takes $O(n^2)$ comparisons on average and in the worst case.

Merge Sort

```
fun merge ([], l)        = l : real list
  | merge (l, [])        = l
  | merge (h1::t1,h2::t2)=
       if h1 <= h2
           then h1::merge(t1, h2::t2)
           else h2::merge(h1::t1, t2);


fun mergesort []    = []
  | mergesort [x] = [x]
  | mergesort l   =
     let val k = length l div 2 in
      merge(mergesort (List.take(l, k)),
            mergesort (List.drop(l, k)))
     end;
```

Merge sort takes $O(n \log n)$ comparisons on average and in the worst case.

Quick Sort

```
fun quick []       = []
  | quick [x]      = [x] : real list
  | quick (h::t) =
 let fun part (left, right, []) =
              (quick left)@(h::quick right)
           | part (left, right, x::l) =
              if x<=h
                then part (x::left, right, l)
                else part (left, x::right, l)
 in
   part( [], [], t) end;
```

Quick sort takes $O(n \log n)$ comparisons on average and $O(n^2)$ in the worst case.

QS without Append

```
fun quik ([], sorted)    = sorted
  | quik ([x], sorted)  = (x:real)::sorted
  | quik (h::t, sorted) =
 let
  fun part (left, right, [])  =
       quik(left, h::quik(right, sorted))
    | part (left, right, x::l) =
       if x<= h
          then part (x::left, right, l)
          else part (left, x::right, l)
 in
   part([], [], t) end;
```