2.1

```
fun ipower acc (x,0) = acc : real
  | ipower acc (x,n) =
      if n mod 2 = 0
        then ipower acc (x * x,n div 2)
        else ipower (acc * x) (x * x,n div 2);
val power = ipower 1.0;
```

3.1

```
(* recursive *)
fun rsum [] = 0
  | rsum (x::xs) = x + rsum xs;

(* iterative *)
fun isum acc [] = acc
  | isum acc (x::xs) = isum (x + acc) xs;
val sum = isum 0;
```

The recurisve function takes O(n) space on the call stack (where n is the length of the list). The iterative version, with optimisations, takes O(1) space (not including the list, of course). They both take O(n) time.

3.2

```
fun last [x] = x
  | last (x::xs) = last xs
  | last [] = raise Empty;
```

O(n) time and O(1) overhead space, assuming optimisations (tail-call and not copying the tail of the list).

3.3

```
(* even? I thought we used 0-basing. :*)
fun oddindexed (x::y::xs) = y :: oddindexed xs
  | oddindexed _ = [];
```