Sorting

It is incredibly useful to be able to sort lists efficiently. There are a number of algorithms to achieve this. We shall discuss a few here.

Insertion sort works by inserting the first element into the tail (which is sorted recursively), where it is inserted after all the values that are less then it but before all the values that are greater than it.

```
insert : 'a -> 'a list -> 'a list
sort : 'a list -> 'a list

let rec sort l =
  let rec insert x s =
    match s with
      [] -> [x]
    | h::t ->
        if x <= h
          then x :: h :: t
          else h :: insert x t
  in
    match l with
      [] -> []
    | h::t -> insert h (sort t)
```

When discussing algorithms, it is important to consider their efficiency. Each insert takes O(n) as the element could be inserted anywhere in the list. The insert function must be run n times so insertion sort has efficiency $O(n^2)$.

A more efficient sorting algorithm is *merge sort*.

```
merge : 'a list -> 'a list -> 'a list
msort : 'a list -> 'a list

let rec merge x y =
  match x, y with
    [], l -> l
  | l, [] -> l
  | hx::tx, hy::ty ->
      if hx < hy
        then hx :: merge tx (hy :: ty)
        else hy :: merge (hx :: tx) ty

let rec msort l =
  match l with
    [] -> []
  | [x] -> [x]
  | _ ->
      let x = length l / 2 in
        let left = take x l in
          let right = drop x l in
            merge (msort left) (msort right)
```

This works by merging two sorted lists, where these lists have been recursively sorted. This has time efficiency $O(n \log n)$ (see Whitington for a visual explaination).

As an aside, when sorting a list of lists, OCaml sorts initially by the first element of the lists. If these are equal, it then compares the next element and so on. If one list is shorter than another but contains the same first n elements (where n is the length of the shorter list), the shorter list will evaluate to less than the longer list when applying an operator $<$. Note that this is exactly how a dictionary sorts words alphabetically.