Isabelle/HOL Exercises
Advanced

# Merge Sort

## Sorting with lists

For simplicity we sort natural numbers.

Define a predicate *sorted* that checks if each element in the list is less or equal to the following ones; *le n xs* should be true iff *n* is less or equal to all elements of *xs*.

```
consts
  le     :: "nat ⇒ nat list ⇒ bool"
  sorted :: "nat list ⇒ bool"

primrec
  "le a []     = True"
  "le a (x#xs) = (a <= x & le a xs)"

primrec
  "sorted []     = True"
  "sorted (x#xs) = (le x xs & sorted xs)"
```

Define a function *count xs x* that counts how often *x* occurs in *xs*.

```
consts
  count :: "nat list => nat => nat"

primrec
  "count []     y = 0"
  "count (x#xs) y = (if x=y then Suc(count xs y) else count xs y)"
```

## Merge sort

Implement *merge sort*: a list is sorted by splitting it into two lists, sorting them separately, and merging the results.

With the help of *recdef* define two functions

```
consts merge :: "nat list × nat list ⇒ nat list"
       msort :: "nat list ⇒ nat list"
```

```
recdef merge "measure (%(xs,ys). size xs + size ys)"
  "merge (x#xs, y#ys) = (if x <= y then x # merge(xs,y#ys) else y #
merge(x#xs,ys))"
  "merge (xs,   [])   = xs"
  "merge ([],   ys)   = ys"

recdef msort "measure size"
  "msort []  = []"
  "msort [x] = [x]"
  "msort xs  = merge (msort(take (size xs div 2) xs), msort(drop (size xs div 2)
xs))"
```

and show

```
theorem "sorted (msort xs)"
theorem "count (msort xs) x = count xs x"
lemma [simp]: "x ≤ y ⟹ le y xs ⟶ le x xs"
  apply (induct_tac xs)
  apply auto
done

lemma [simp]: "count (merge(xs,ys)) x = count xs x + count ys x"
  apply(induct xs ys rule: merge.induct)
  apply auto
done

lemma [simp]: "le x (merge (xs,ys)) = (le x xs ∧ le x ys)"
  apply (induct xs ys rule: merge.induct)
  apply auto
done

lemma [simp]: "sorted (merge(xs,ys)) = (sorted xs ∧ sorted ys)"
  apply(induct xs ys rule: merge.induct)
  apply (auto simp add: linorder_not_le order_less_le)
done

lemma [simp]: "1 < x ⟹ min x (x div 2::nat) < x"
  by (simp add: min_def linorder_not_le)

lemma [simp]: "1 < x ⟹ x - x div (2::nat) < x"
  by arith

theorem "sorted (msort xs)"
```

```
  apply (induct_tac xs rule: msort.induct)
  apply auto
done

lemma count_append[simp]: "count (xs @ ys) x = count xs x + count ys x"
  apply (induct xs)
  apply auto
done

theorem "count (msort xs) x = count xs x"
  apply (induct xs rule: msort.induct)
      apply simp
    apply simp
  apply simp
  apply (simp del:count_append add:count_append[symmetric])
done
```

You may have to prove lemmas about `sol.sorted` and `count`.

Hints:

- For `recdef` see Section 3.5 of the Isabelle/HOL tutorial.
- To split a list into two halves of almost equal length you can use the functions `n div 2`, `take` und `drop`, where `take n xs` returns the first `n` elements of `xs` and `drop n xs` the remainder.
- Here are some potentially useful lemmas:

```
linorder_not_le: (¬ x ≤ y) = (y < x)
order_less_le: (x < y) = (x ≤ y ∧ x ≠ y)
min_def: min a b = (if a ≤ b then a else b)
```