

Efficient Mergesort

Christian Sternagel

May 26, 2024

Abstract

We provide a formalization of the mergesort algorithm as used in GHC's `Data.List` module, proving correctness and stability. Furthermore, experimental data suggests that generated (Haskell-)code for this algorithm is much faster than for previous algorithms available in the Isabelle distribution.

```
theory Efficient-Sort  
  imports HOL-Library.Multiset  
begin
```

1 GHC Version of Mergesort

In the following we show that the mergesort implementation used in GHC (see <http://hackage.haskell.org/package/base-4.11.1.0/docs/src/Data.OldList.html#sort>) is a correct and stable sorting algorithm. Furthermore, experimental data suggests that generated code for this implementation is much more efficient than for the implementation provided by *HOL-Library.Multiset*. A high-level overview of an older version of this formalization as well as some experimental data is to be found in [1].

1.1 Definition of Natural Mergesort

```
context  
  fixes key :: 'a ⇒ 'k::linorder  
begin
```

Split a list into chunks of ascending and descending parts, where descending parts are reversed on the fly. Thus, the result is a list of sorted lists.

```
fun sequences :: 'a list ⇒ 'a list list  
  and asc :: 'a ⇒ ('a list ⇒ 'a list) ⇒ 'a list ⇒ 'a list list  
  and desc :: 'a ⇒ 'a list ⇒ 'a list ⇒ 'a list list  
  where  
    sequences (a # b # xs) =
```

```

    (if key a > key b then desc b [a] xs else asc b ((#) a) xs)
| sequences [x] = [[x]]
| sequences [] = []
| asc a as (b # bs) =
    (if key a ≤ key b then asc b (λys. as (a # ys)) bs
    else as [a] # sequences (b # bs))
| asc a as [] = [as [a]]
| desc a as (b # bs) =
    (if key a > key b then desc b (a # as) bs
    else (a # as) # sequences (b # bs))
| desc a as [] = [a # as]

```

fun merge :: 'a list ⇒ 'a list ⇒ 'a list

where

```

    merge (a # as) (b # bs) =
        (if key a > key b then b # merge (a # as) bs else a # merge as (b # bs))
| merge [] bs = bs
| merge as [] = as

```

fun merge-pairs :: 'a list list ⇒ 'a list list

where

```

    merge-pairs (a # b # xs) = merge a b # merge-pairs xs
| merge-pairs xs = xs

```

lemma length-merge [simp]:

```

length (merge xs ys) = length xs + length ys
⟨proof⟩

```

lemma length-merge-pairs [simp]:

```

length (merge-pairs xs) = (1 + length xs) div 2
⟨proof⟩

```

fun merge-all :: 'a list list ⇒ 'a list

where

```

    merge-all [] = []
| merge-all [x] = x
| merge-all xs = merge-all (merge-pairs xs)

```

fun msort-key :: 'a list ⇒ 'a list

where

```

msort-key xs = merge-all (sequences xs)

```

1.2 The Functional Argument of *local.asc*

f is a function that only adds some prefix to a given list.

definition ascP *f* = (∀ *xs*. *f xs* = *f* [] @ *xs*)

lemma ascP-Cons [simp]: ascP ((#) *x*) ⟨proof⟩

lemma *ascP-comp-append-Cons* [simp]:
 $ascP (\lambda xs. f [] @ x \# xs)$
 ⟨proof⟩

lemma *ascP-f-Cons*:
assumes *ascP f*
shows $f (x \# xs) = f [] @ x \# xs$
 ⟨proof⟩

lemma *ascP-comp-Cons* [simp]:
assumes *ascP f*
shows $ascP (\lambda ys. f (x \# ys))$
 ⟨proof⟩

lemma *ascP-f-singleton*:
assumes *ascP f*
shows $f [x] = f [] @ [x]$
 ⟨proof⟩

1.3 Facts about Lengths

lemma
shows *length-sequences*: $length (sequences\ xs) \leq length\ xs$
and *length-asc*: $ascP\ f \implies length (asc\ a\ f\ ys) \leq 1 + length\ ys$
and *length-desc*: $length (desc\ a\ xs\ ys) \leq 1 + length\ ys$
 ⟨proof⟩

lemma *length-concat-merge-pairs* [simp]:
 $length (concat (merge-pairs\ xss)) = length (concat\ xss)$
 ⟨proof⟩

1.4 Functional Correctness

lemma *mset-merge* [simp]:
 $mset (merge\ xs\ ys) = mset\ xs + mset\ ys$
 ⟨proof⟩

lemma *set-merge* [simp]:
 $set (merge\ xs\ ys) = set\ xs \cup set\ ys$
 ⟨proof⟩

lemma *mset-concat-merge-pairs* [simp]:
 $mset (concat (merge-pairs\ xs)) = mset (concat\ xs)$
 ⟨proof⟩

lemma *set-concat-merge-pairs* [simp]:
 $set (concat (merge-pairs\ xs)) = set (concat\ xs)$
 ⟨proof⟩

lemma *mset-merge-all* [simp]:

$mset (merge\text{-}all\ xs) = mset (concat\ xs)$
<proof>

lemma *set-merge-all* [simp]:
 $set (merge\text{-}all\ xs) = set (concat\ xs)$
<proof>

lemma
shows *mset-sequences* [simp]: $mset (concat (sequences\ xs)) = mset\ xs$
and *mset-asc*: $ascP\ f \implies mset (concat (asc\ x\ f\ ys)) = \{\#x\} + mset (f\ [])$
 $+ mset\ ys$
and *mset-desc*: $mset (concat (desc\ x\ xs\ ys)) = \{\#x\} + mset\ xs + mset\ ys$
<proof>

lemma *mset-msort-key*:
 $mset (msort\text{-}key\ xs) = mset\ xs$
<proof>

lemma *sorted-merge* [simp]:
assumes *sorted* (map key xs) **and** *sorted* (map key ys)
shows *sorted* (map key (merge xs ys))
<proof>

lemma *sorted-merge-pairs* [simp]:
assumes $\forall x \in set\ xs. sorted (map\ key\ x)$
shows $\forall x \in set (merge\text{-}pairs\ xs). sorted (map\ key\ x)$
<proof>

lemma *sorted-merge-all*:
assumes $\forall x \in set\ xs. sorted (map\ key\ x)$
shows *sorted* (map key (merge-all xs))
<proof>

lemma
shows *sorted-sequences*: $\forall x \in set (sequences\ xs). sorted (map\ key\ x)$
and *sorted-asc*: $ascP\ f \implies sorted (map\ key (f\ [])) \implies \forall x \in set (f\ []). key\ x \leq$
 $key\ a \implies \forall x \in set (asc\ a\ f\ ys). sorted (map\ key\ x)$
and *sorted-desc*: $sorted (map\ key\ xs) \implies \forall x \in set\ xs. key\ a \leq key\ x \implies \forall x \in set$
 $(desc\ a\ xs\ ys). sorted (map\ key\ x)$
<proof>

lemma *sorted-msort-key*:
 $sorted (map\ key (msort\text{-}key\ xs))$
<proof>

1.5 Stability

lemma
shows *filter-by-key-sequences* [simp]: $[y \leftarrow concat (sequences\ xs). key\ y = k] =$

```

[y←xs. key y = k]
  and filter-by-key-asc: ascP f ⇒ [y←concat (asc a f ys). key y = k] = [y←f
[a] @ ys. key y = k]
  and filter-by-key-desc: sorted (map key xs) ⇒ ∀x∈set xs. key a ≤ key x ⇒
[y←concat (desc a xs ys). key y = k] = [y←a # xs @ ys. key y = k]
⟨proof⟩

```

lemma *filter-by-key-merge-is-append* [simp]:

assumes sorted (map key xs)

shows [y←merge xs ys. key y = k] = [y←xs. key y = k] @ [y←ys. key y = k]

⟨proof⟩

lemma *filter-by-key-merge-pairs* [simp]:

assumes ∀xs∈set xss. sorted (map key xs)

shows [y←concat (merge-pairs xss). key y = k] = [y←concat xss. key y = k]

⟨proof⟩

lemma *filter-by-key-merge-all* [simp]:

assumes ∀xs∈set xss. sorted (map key xs)

shows [y←merge-all xss. key y = k] = [y←concat xss. key y = k]

⟨proof⟩

lemma *filter-by-key-merge-all-sequences* [simp]:

[x←merge-all (sequences xs) . key x = k] = [x←xs . key x = k]

⟨proof⟩

lemma *msort-key-stable*:

[x←msort-key xs. key x = k] = [x←xs. key x = k]

⟨proof⟩

lemma *sort-key-msort-key-conv*:

sort-key key = msort-key

⟨proof⟩

end

Replace existing code equations for *sort-key* by *msort-key*.

declare *sort-key-by-quicksort-code* [code del]

declare *sort-key-msort-key-conv* [code]

end

theory *Mergesort-Complexity*

imports

Efficient-Sort

Complex-Main

begin

lemma *log2-mono*:

$x > 0 \implies x \leq y \implies \log 2 x \leq \log 2 y$

<proof>

2 Counting the Number of Comparisons

context

fixes *key* :: 'a \Rightarrow 'k::linorder

begin

fun *c-merge* :: 'a list \Rightarrow 'a list \Rightarrow nat

where

$c\text{-merge } (x \# xs) (y \# ys) =$

$1 + (\text{if } \text{key } y < \text{key } x \text{ then } c\text{-merge } (x \# xs) \text{ ys else } c\text{-merge } xs (y \# ys))$

| $c\text{-merge } [] \text{ ys} = 0$

| $c\text{-merge } xs [] = 0$

fun *c-merge-pairs* :: 'a list list \Rightarrow nat

where

$c\text{-merge-pairs } (xs \# ys \# zss) = c\text{-merge } xs \text{ ys} + c\text{-merge-pairs } zss$

| $c\text{-merge-pairs } [] = 0$

| $c\text{-merge-pairs } [x] = 0$

fun *c-merge-all* :: 'a list list \Rightarrow nat

where

$c\text{-merge-all } [] = 0$

| $c\text{-merge-all } [x] = 0$

| $c\text{-merge-all } xss = c\text{-merge-pairs } xss + c\text{-merge-all } (\text{merge-pairs } \text{key } xss)$

fun *c-sequences* :: 'a list \Rightarrow nat

and *c-asc* :: 'a \Rightarrow 'a list \Rightarrow nat

and *c-desc* :: 'a \Rightarrow 'a list \Rightarrow nat

where

$c\text{-sequences } (x \# y \# zs) = 1 + (\text{if } \text{key } y < \text{key } x \text{ then } c\text{-desc } y \text{ zs else } c\text{-asc } y$
 $zs)$

| $c\text{-sequences } [] = 0$

| $c\text{-sequences } [x] = 0$

| $c\text{-asc } x (y \# ys) = 1 + (\text{if } \neg \text{key } y < \text{key } x \text{ then } c\text{-asc } y \text{ ys else } c\text{-sequences } (y$
 $\# ys))$

| $c\text{-asc } x [] = 0$

| $c\text{-desc } x (y \# ys) = 1 + (\text{if } \text{key } y < \text{key } x \text{ then } c\text{-desc } y \text{ ys else } c\text{-sequences } (y$
 $\# ys))$

| $c\text{-desc } x [] = 0$

fun *c-msort* :: 'a list \Rightarrow nat

where

$c\text{-msort } xs = c\text{-sequences } xs + c\text{-merge-all } (\text{sequences } \text{key } xs)$

lemma *c-merge*:

c-merge $xs\ ys \leq \text{length } xs + \text{length } ys$
(proof)

lemma *c-merge-pairs*:

c-merge-pairs $xss \leq \text{length } (\text{concat } xss)$
(proof)

lemma *c-merge-all*:

c-merge-all $xss \leq \text{length } (\text{concat } xss) * \lceil \log 2 (\text{length } xss) \rceil$
(proof)

lemma

shows *c-sequences*: $c\text{-sequences } xs \leq \text{length } xs - 1$
and *c-asc*: $c\text{-asc } x\ ys \leq \text{length } ys$
and *c-desc*: $c\text{-desc } x\ ys \leq \text{length } ys$
(proof)

lemma

shows *length-concat-sequences* [*simp*]: $\text{length } (\text{concat } (\text{sequences key } xs)) = \text{length } xs$
and *length-concat-asc*: $\text{ascP } f \implies \text{length } (\text{concat } (\text{asc key } a\ f\ ys)) = 1 + \text{length } (f\ []) + \text{length } ys$
and *length-concat-desc*: $\text{length } (\text{concat } (\text{desc key } a\ xs\ ys)) = 1 + \text{length } xs + \text{length } ys$
(proof)

lemma

shows *sequences-ne*: $xs \neq [] \implies \text{sequences key } xs \neq []$
and *asc-ne*: $\text{ascP } f \implies \text{asc key } a\ f\ ys \neq []$
and *desc-ne*: $\text{desc key } a\ xs\ ys \neq []$
(proof)

lemma *c-msort*:

assumes [*simp*]: $\text{length } xs = n$
shows *c-msort* $xs \leq n + n * \lceil \log 2\ n \rceil$
(proof)

end

end

theory *Natural-Mergesort*

imports *HOL-Data-Structures.Sorting*
begin

2.1 Auxiliary Results

lemma *C-merge-adj'*:

$C\text{-merge-adj } xss \leq \text{length } (\text{concat } xss)$
 ⟨proof⟩

lemma *length-concat-merge-adj*:
 $\text{length } (\text{concat } (\text{merge-adj } xss)) = \text{length } (\text{concat } xss)$
 ⟨proof⟩

lemma *C-merge-all'*:
 $C\text{-merge-all } xss \leq \text{length } (\text{concat } xss) * \lceil \log 2 (\text{length } xss) \rceil$
 ⟨proof⟩

2.2 Definition of Natural Mergesort

Partition input into ascending and descending subsequences. (The latter are reverted on the fly.)

fun *runs* :: ('a::linorder) list ⇒ 'a list list **and**
 $\text{asc} :: 'a \Rightarrow ('a \text{ list} \Rightarrow 'a \text{ list}) \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list list}$ **and**
 $\text{desc} :: 'a \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list list}$
where
 $\text{runs } (a \# b \# xs) = (\text{if } a > b \text{ then } \text{desc } b [a] xs \text{ else } \text{asc } b ((\#) a) xs)$
 $|\text{ runs } [x] = [[x]]$
 $|\text{ runs } [] = []$
 $|\text{ asc } a \text{ as } (b \# bs) = (\text{if } \neg a > b \text{ then } \text{asc } b (\text{as } \circ (\#) a) bs \text{ else } \text{as } [a] \# \text{runs } (b \# bs))$
 $|\text{ asc } a \text{ as } [] = [\text{as } [a]]$
 $|\text{ desc } a \text{ as } (b \# bs) = (\text{if } a > b \text{ then } \text{desc } b (a \# \text{as}) bs \text{ else } (a \# \text{as}) \# \text{runs } (b \# bs))$
 $|\text{ desc } a \text{ as } [] = [a \# \text{as}]$

definition *nmsort* :: ('a::linorder) list ⇒ 'a list
where
 $\text{nmsort } xs = \text{merge-all } (\text{runs } xs)$

2.3 Functional Correctness

definition *ascP* $f = (\forall xs \ ys. f (xs @ ys) = f xs @ ys)$

lemma *ascP-Cons [simp]*: $\text{ascP } ((\#) x)$ ⟨proof⟩

lemma *ascP-comp-Cons [simp]*: $\text{ascP } f \implies \text{ascP } (f \circ (\#) x)$
 ⟨proof⟩

lemma *ascP-simp [simp]*:
assumes $\text{ascP } f$
shows $f [x] = f [] @ [x]$
 ⟨proof⟩

lemma
shows *mset-runs*: $\sum \# (\text{image-mset } \text{mset } (\text{mset } (\text{runs } xs))) = \text{mset } xs$

and *mset-asc*: $ascP f \implies \sum \# (image-mset\ mset\ (mset\ (asc\ x\ f\ ys))) = \{\#x\# \}$
 $+ mset\ (f\ []) + mset\ ys$
and *mset-desc*: $\sum \# (image-mset\ mset\ (mset\ (desc\ x\ xs\ ys))) = \{\#x\# \} + mset\ xs + mset\ ys$
 <proof>

lemma *mset-nmsort*:
 $mset\ (nmsort\ xs) = mset\ xs$
 <proof>

lemma
shows *sorted-runs*: $\forall x \in set\ (runs\ xs). sorted\ x$
and *sorted-asc*: $ascP f \implies sorted\ (f\ []) \implies \forall x \in set\ (f\ []). x \leq a \implies \forall x \in set\ (asc\ a\ f\ ys). sorted\ x$
and *sorted-desc*: $sorted\ xs \implies \forall x \in set\ xs. a \leq x \implies \forall x \in set\ (desc\ a\ xs\ ys). sorted\ x$
 <proof>

lemma *sorted-nmsort*:
 $sorted\ (nmsort\ xs)$
 <proof>

2.4 Running Time Analysis

fun *C-runs* :: ('a::linorder) list \Rightarrow nat **and**
C-asc :: ('a::linorder) \Rightarrow 'a list \Rightarrow nat **and**
C-desc :: ('a::linorder) \Rightarrow 'a list \Rightarrow nat
where
 $C-runs\ (a\ \# b\ \# xs) = 1 + (if\ a > b\ then\ C-desc\ b\ xs\ else\ C-asc\ b\ xs)$
 $C-runs\ xs = 0$
 $C-asc\ a\ (b\ \# bs) = 1 + (if\ \neg a > b\ then\ C-asc\ b\ bs\ else\ C-runs\ (b\ \# bs))$
 $C-asc\ a\ [] = 0$
 $C-desc\ a\ (b\ \# bs) = 1 + (if\ a > b\ then\ C-desc\ b\ bs\ else\ C-runs\ (b\ \# bs))$
 $C-desc\ a\ [] = 0$

fun *C-nmsort* :: ('a::linorder) list \Rightarrow nat
where
 $C-nmsort\ xs = C-runs\ xs + C-merge-all\ (runs\ xs)$

lemma
fixes $a :: 'a::linorder$ **and** $xs\ ys :: 'a\ list$
shows *C-runs*: $C-runs\ xs \leq length\ xs - 1$
and *C-asc*: $C-asc\ a\ ys \leq length\ ys$
and *C-desc*: $C-desc\ a\ ys \leq length\ ys$
 <proof>

lemma
shows *length-runs*: $length\ (runs\ xs) \leq length\ xs$
and *length-asc*: $ascP\ f \implies length\ (asc\ a\ f\ ys) \leq 1 + length\ ys$

and *length-desc*: $\text{length } (\text{desc } a \text{ } xs \text{ } ys) \leq 1 + \text{length } ys$
(*proof*)

lemma

shows *length-concat-runs* [*simp*]: $\text{length } (\text{concat } (\text{runs } xs)) = \text{length } xs$
and *length-concat-asc*: $\text{ascP } f \implies \text{length } (\text{concat } (\text{asc } a \text{ } f \text{ } ys)) = 1 + \text{length } (f$
[]) $+ \text{length } ys$
and *length-concat-desc*: $\text{length } (\text{concat } (\text{desc } a \text{ } xs \text{ } ys)) = 1 + \text{length } xs + \text{length } ys$
(*proof*)

lemma *log2-mono*:

$x > 0 \implies x \leq y \implies \log 2 \ x \leq \log 2 \ y$
(*proof*)

lemma

shows *runs-ne*: $xs \neq [] \implies \text{runs } xs \neq []$
and *ascP f* $\implies \text{asc } a \text{ } f \text{ } ys \neq []$
and *desc a xs ys* $\neq []$
(*proof*)

lemma *C-nmsort*:

assumes [*simp*]: $\text{length } xs = n$
shows *C-nmsort* $xs \leq n + n * \lceil \log 2 \ n \rceil$
(*proof*)

end

References

- [1] Christian Sternagel. Proof pearl - a mechanized proof of GHC's mergesort. *Journal of Automated Reasoning*, 2012. doi:10.1007/s10817-012-9260-7.