

# Comparison-based Sorting Algorithms

Manuel Eberl

February 6, 2026

## Abstract

This article contains a formal proof of the well-known fact that number of comparisons that a comparison-based sorting algorithm needs to perform to sort a list of length  $n$  is at least  $\log_2(n!)$  in the worst case, i. e.  $\Omega(n \log n)$ .

For this purpose, a shallow embedding for comparison-based sorting algorithms is defined: a sorting algorithm is a recursive datatype containing either a HOL function or a query of a comparison oracle with a continuation containing the remaining computation. This makes it possible to force the algorithm to use only comparisons and to track the number of comparisons made.

## Contents

<b>1</b>	<b>Linear orderings as relations</b>	<b>2</b>
1.1	Auxiliary facts . . . . .	2
1.2	Sortedness w.r.t. a relation . . . . .	2
1.3	Linear orderings . . . . .	3
1.4	Converting a list into a linear ordering . . . . .	4
1.5	Insertion sort . . . . .	4
1.6	Obtaining a sorted list of a given set . . . . .	5
1.7	Rank of an element in an ordering . . . . .	6
1.8	The bijection between linear orderings and lists . . . . .	7
<b>2</b>	<b>Lower bound on costs of comparison-based sorting</b>	<b>7</b>
2.1	Abstract description of sorting algorithms . . . . .	8
2.2	Lower bounds on number of comparisons . . . . .	9

# 1 Linear orderings as relations

```
theory Linorder-Relations
  imports
    Complex-Main
    HOL-Combinatorics.Multiset-Permutations
    List-Index.List-Index
begin
```

## 1.1 Auxiliary facts

```
lemma distinct-count-atmost-1':
  distinct xs = ( $\forall a. \text{count } (\text{mset } xs) a \leq 1$ )
<proof>
```

```
lemma distinct-mset-mono:
  assumes distinct ys mset xs  $\subseteq\#$  mset ys
  shows distinct xs
<proof>
```

```
lemma mset-eq-imp-distinct-iff:
  assumes mset xs = mset ys
  shows distinct xs  $\longleftrightarrow$  distinct ys
<proof>
```

```
lemma total-on-subset: total-on B R  $\implies$  A  $\subseteq$  B  $\implies$  total-on A R
<proof>
```

## 1.2 Sortedness w.r.t. a relation

```
inductive sorted-wrt :: ('a  $\times$  'a) set  $\Rightarrow$  'a list  $\Rightarrow$  bool for R where
  sorted-wrt R []
| sorted-wrt R xs  $\implies$  ( $\bigwedge y. y \in \text{set } xs \implies (x,y) \in R$ )  $\implies$  sorted-wrt R (x # xs)
```

```
lemma sorted-wrt-Nil [simp]: sorted-wrt R []
<proof>
```

```
lemma sorted-wrt-Cons: sorted-wrt R (x # xs)  $\longleftrightarrow$  ( $\forall y \in \text{set } xs. (x,y) \in R$ )  $\wedge$ 
sorted-wrt R xs
<proof>
```

```
lemma sorted-wrt-singleton [simp]: sorted-wrt R [x]
<proof>
```

```
lemma sorted-wrt-many:
  assumes trans R
  shows sorted-wrt R (x # y # xs)  $\longleftrightarrow$  (x,y)  $\in$  R  $\wedge$  sorted-wrt R (y # xs)
<proof>
```

```
lemma sorted-wrt-imp-le-last:
```

**assumes** *sorted-wrt*  $R$   $xs$   $xs \neq []$   $x \in set\ xs$   $x \neq last\ xs$   
**shows**  $(x, last\ xs) \in R$   
 $\langle proof \rangle$

**lemma** *sorted-wrt-append*:

**assumes** *sorted-wrt*  $R$   $xs$  *sorted-wrt*  $R$   $ys$   
 $\bigwedge x\ y. x \in set\ xs \implies y \in set\ ys \implies (x,y) \in R\ trans\ R$   
**shows** *sorted-wrt*  $R$   $(xs @ ys)$   
 $\langle proof \rangle$

**lemma** *sorted-wrt-snoc*:

**assumes** *sorted-wrt*  $R$   $xs$   $(last\ xs, y) \in R\ trans\ R$   
**shows** *sorted-wrt*  $R$   $(xs @ [y])$   
 $\langle proof \rangle$

**lemma** *sorted-wrt-conv-nth*:

*sorted-wrt*  $R$   $xs \longleftrightarrow (\forall i\ j. i < j \wedge j < length\ xs \longrightarrow (xs!i, xs!j) \in R)$   
 $\langle proof \rangle$

### 1.3 Linear orderings

**definition** *linorder-on* ::  $'a\ set \implies ('a \times 'a)\ set \implies bool$  **where**

*linorder-on*  $A\ R \longleftrightarrow R \subseteq A \times A \wedge refl\text{-on}\ A\ R \wedge antisym\ R \wedge trans\ R \wedge total\text{-on}\ A\ R$

**lemma** *linorder-on-cases*:

**assumes** *linorder-on*  $A\ R$   $x \in A$   $y \in A$   
**shows**  $x = y \vee ((x, y) \in R \wedge (y, x) \notin R) \vee ((y, x) \in R \wedge (x, y) \notin R)$   
 $\langle proof \rangle$

**lemma** *sorted-wrt-linorder-imp-index-le*:

**assumes** *linorder-on*  $A\ R$   $set\ xs \subseteq A$  *sorted-wrt*  $R$   $xs$   
 $x \in set\ xs$   $y \in set\ xs$   $(x,y) \in R$   
**shows**  $index\ xs\ x \leq index\ xs\ y$   
 $\langle proof \rangle$

**lemma** *sorted-wrt-linorder-index-le-imp*:

**assumes** *linorder-on*  $A\ R$   $set\ xs \subseteq A$  *sorted-wrt*  $R$   $xs$   
 $x \in set\ xs$   $y \in set\ xs$   $index\ xs\ x \leq index\ xs\ y$   
**shows**  $(x,y) \in R$   
 $\langle proof \rangle$

**lemma** *sorted-wrt-linorder-index-le-iff*:

**assumes** *linorder-on*  $A\ R$   $set\ xs \subseteq A$  *sorted-wrt*  $R$   $xs$   
 $x \in set\ xs$   $y \in set\ xs$   
**shows**  $index\ xs\ x \leq index\ xs\ y \longleftrightarrow (x,y) \in R$   
 $\langle proof \rangle$

**lemma** *sorted-wrt-linorder-index-less-iff*:

**assumes** *linorder-on*  $A$   $R$  *set*  $xs \subseteq A$  *sorted-wrt*  $R$   $xs$   
 $x \in \text{set } xs \ y \in \text{set } xs$   
**shows**  $\text{index } xs \ x < \text{index } xs \ y \longleftrightarrow (y,x) \notin R$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-wrt-distinct-linorder-nth*:  
**assumes** *linorder-on*  $A$   $R$  *set*  $xs \subseteq A$  *sorted-wrt*  $R$   $xs$  *distinct*  $xs$   
 $i < \text{length } xs \ j < \text{length } xs$   
**shows**  $(xs!i, xs!j) \in R \longleftrightarrow i \leq j$   
 $\langle \text{proof} \rangle$

## 1.4 Converting a list into a linear ordering

**definition** *linorder-of-list* ::  $'a \text{ list} \Rightarrow ('a \times 'a) \text{ set}$  **where**  
 $\text{linorder-of-list } xs = \{(a,b). a \in \text{set } xs \wedge b \in \text{set } xs \wedge \text{index } xs \ a \leq \text{index } xs \ b\}$

**lemma** *linorder-linorder-of-list* [*intro, simp*]:  
**assumes** *distinct*  $xs$   
**shows** *linorder-on*  $(\text{set } xs)$   $(\text{linorder-of-list } xs)$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-wrt-linorder-of-list* [*intro, simp*]:  
 $\text{distinct } xs \Longrightarrow \text{sorted-wrt } (\text{linorder-of-list } xs) \ xs$   
 $\langle \text{proof} \rangle$

## 1.5 Insertion sort

**primrec** *insert-wrt* ::  $('a \times 'a) \text{ set} \Rightarrow 'a \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$  **where**  
 $\text{insert-wrt } R \ x \ [] = [x]$   
 $|\ \text{insert-wrt } R \ x \ (y \# ys) = (\text{if } (x, y) \in R \ \text{then } x \# y \# ys \ \text{else } y \# \text{insert-wrt } R \ x \ ys)$

**lemma** *set-insert-wrt* [*simp*]:  $\text{set } (\text{insert-wrt } R \ x \ xs) = \text{insert } x \ (\text{set } xs)$   
 $\langle \text{proof} \rangle$

**lemma** *mset-insert-wrt* [*simp*]:  $\text{mset } (\text{insert-wrt } R \ x \ xs) = \text{add-mset } x \ (\text{mset } xs)$   
 $\langle \text{proof} \rangle$

**lemma** *length-insert-wrt* [*simp*]:  $\text{length } (\text{insert-wrt } R \ x \ xs) = \text{Suc } (\text{length } xs)$   
 $\langle \text{proof} \rangle$

**definition** *insort-wrt* ::  $('a \times 'a) \text{ set} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$  **where**  
 $\text{insort-wrt } R \ xs = \text{foldr } (\text{insert-wrt } R) \ xs \ []$

**lemma** *set-insort-wrt* [*simp*]:  $\text{set } (\text{insort-wrt } R \ xs) = \text{set } xs$   
 $\langle \text{proof} \rangle$

**lemma** *mset-insort-wrt* [*simp*]:  $\text{mset } (\text{insort-wrt } R \ xs) = \text{mset } xs$   
 $\langle \text{proof} \rangle$

**lemma** *length-insort-wrt* [simp]:  $\text{length } (\text{insort-wrt } R \ xs) = \text{length } xs$   
 ⟨proof⟩

**lemma** *sorted-wrt-insert-wrt* [intro]:  
 $\text{linorder-on } A \ R \implies \text{set } (x \# \ xs) \subseteq A \implies$   
 $\text{sorted-wrt } R \ xs \implies \text{sorted-wrt } R \ (\text{insert-wrt } R \ x \ xs)$   
 ⟨proof⟩

**lemma** *sorted-wrt-insort* [intro]:  
**assumes**  $\text{linorder-on } A \ R \ \text{set } xs \subseteq A$   
**shows**  $\text{sorted-wrt } R \ (\text{insort-wrt } R \ xs)$   
 ⟨proof⟩

**lemma** *distinct-insort-wrt* [simp]:  $\text{distinct } (\text{insort-wrt } R \ xs) \longleftrightarrow \text{distinct } xs$   
 ⟨proof⟩

**lemma** *sorted-wrt-linorder-unique*:  
**assumes**  $\text{linorder-on } A \ R \ \text{mset } xs = \text{mset } ys \ \text{sorted-wrt } R \ xs \ \text{sorted-wrt } R \ ys$   
**shows**  $xs = ys$   
 ⟨proof⟩

## 1.6 Obtaining a sorted list of a given set

**definition** *sorted-wrt-list-of-set* **where**  
 $\text{sorted-wrt-list-of-set } R \ A =$   
 (if finite  $A$  then (THE  $xs$ .  $\text{set } xs = A \wedge \text{distinct } xs \wedge \text{sorted-wrt } R \ xs$ ) else [])

**lemma** *mset-remdups*:  $\text{mset } (\text{remdups } xs) = \text{mset-set } (\text{set } xs)$   
 ⟨proof⟩

**lemma** *sorted-wrt-list-set*:  
**assumes**  $\text{linorder-on } A \ R \ \text{set } xs \subseteq A$   
**shows**  $\text{sorted-wrt-list-of-set } R \ (\text{set } xs) = \text{insort-wrt } R \ (\text{remdups } xs)$   
 ⟨proof⟩

**lemma** *linorder-sorted-wrt-exists*:  
**assumes**  $\text{linorder-on } A \ R \ \text{finite } B \ B \subseteq A$   
**shows**  $\exists xs. \text{set } xs = B \wedge \text{distinct } xs \wedge \text{sorted-wrt } R \ xs$   
 ⟨proof⟩

**lemma** *linorder-sorted-wrt-list-of-set*:  
**assumes**  $\text{linorder-on } A \ R \ \text{finite } B \ B \subseteq A$   
**shows**  $\text{set } (\text{sorted-wrt-list-of-set } R \ B) = B \ \text{distinct } (\text{sorted-wrt-list-of-set } R \ B)$   
 $\text{sorted-wrt } R \ (\text{sorted-wrt-list-of-set } R \ B)$   
 ⟨proof⟩

**lemma** *sorted-wrt-list-of-set-eqI*:  
**assumes**  $\text{linorder-on } B \ R \ A \subseteq B \ \text{set } xs = A \ \text{distinct } xs \ \text{sorted-wrt } R \ xs$   
**shows**  $\text{sorted-wrt-list-of-set } R \ A = xs$

*<proof>*

## 1.7 Rank of an element in an ordering

The ‘rank’ of an element in a set w.r.t. an ordering is how many smaller elements exist. This is particularly useful in linear orders, where there exists a unique  $n$ -th element for every  $n$ .

**definition** *linorder-rank* **where**

$$\text{linorder-rank } R \ A \ x = \text{card } \{y \in A - \{x\}. (y, x) \in R\}$$

**lemma** *linorder-rank-le*:

**assumes** *finite*  $A$

**shows** *linorder-rank*  $R \ A \ x \leq \text{card } A$

*<proof>*

**lemma** *linorder-rank-less*:

**assumes** *finite*  $A \ x \in A$

**shows** *linorder-rank*  $R \ A \ x < \text{card } A$

*<proof>*

**lemma** *linorder-rank-union*:

**assumes** *finite*  $A \ \text{finite } B \ A \cap B = \{\}$

**shows** *linorder-rank*  $R \ (A \cup B) \ x = \text{linorder-rank } R \ A \ x + \text{linorder-rank } R \ B \ x$

*<proof>*

**lemma** *linorder-rank-empty* [*simp*]: *linorder-rank*  $R \ \{\} \ x = 0$

*<proof>*

**lemma** *linorder-rank-singleton*:

*linorder-rank*  $R \ \{y\} \ x = (\text{if } x \neq y \wedge (y, x) \in R \text{ then } 1 \text{ else } 0)$

*<proof>*

**lemma** *linorder-rank-insert*:

**assumes** *finite*  $A \ y \notin A$

**shows** *linorder-rank*  $R \ (\text{insert } y \ A) \ x =$

$(\text{if } x \neq y \wedge (y, x) \in R \text{ then } 1 \text{ else } 0) + \text{linorder-rank } R \ A \ x$

*<proof>*

**lemma** *linorder-rank-mono*:

**assumes** *linorder-on*  $B \ R \ \text{finite } A \ A \subseteq B \ (x, y) \in R$

**shows** *linorder-rank*  $R \ A \ x \leq \text{linorder-rank } R \ A \ y$

*<proof>*

**lemma** *linorder-rank-strict-mono*:

**assumes** *linorder-on*  $B \ R \ \text{finite } A \ A \subseteq B \ y \in A \ (y, x) \in R \ x \neq y$

**shows** *linorder-rank*  $R \ A \ y < \text{linorder-rank } R \ A \ x$

*<proof>*

**lemma** *linorder-rank-le-iff*:  
**assumes** *linorder-on B R finite A A ⊆ B x ∈ A y ∈ A*  
**shows** *linorder-rank R A x ≤ linorder-rank R A y ↔ (x, y) ∈ R*  
 ⟨*proof*⟩

**lemma** *linorder-rank-eq-iff*:  
**assumes** *linorder-on B R finite A A ⊆ B x ∈ A y ∈ A*  
**shows** *linorder-rank R A x = linorder-rank R A y ↔ x = y*  
 ⟨*proof*⟩

**lemma** *linorder-rank-set-sorted-wrt*:  
**assumes** *linorder-on B R set xs ⊆ B sorted-wrt R xs x ∈ set xs distinct xs*  
**shows** *linorder-rank R (set xs) x = index xs x*  
 ⟨*proof*⟩

**lemma** *bij-betw-linorder-rank*:  
**assumes** *linorder-on B R finite A A ⊆ B*  
**shows** *bij-betw (linorder-rank R A) A {..<card A}*  
 ⟨*proof*⟩

## 1.8 The bijection between linear orderings and lists

**theorem** *bij-betw-linorder-of-list*:  
**assumes** *finite A*  
**shows** *bij-betw linorder-of-list (permutations-of-set A) {R. linorder-on A R}*  
 ⟨*proof*⟩

**corollary** *card-finite-linorders*:  
**assumes** *finite A*  
**shows** *card {R. linorder-on A R} = fact (card A)*  
 ⟨*proof*⟩

**end**

## 2 Lower bound on costs of comparison-based sorting

**theory** *Comparison-Sort-Lower-Bound*  
**imports**  
*Complex-Main*  
*Linorder-Relations*  
*Stirling-Formula.Stirling-Formula*  
*Landau-Symbols.Landau-More*  
**begin**

## 2.1 Abstract description of sorting algorithms

We have chosen to model a sorting algorithm in the following way: A sorting algorithm takes a list with distinct elements and a linear ordering on these elements, and it returns a list with the same elements that is sorted w. r. t. the given ordering.

The use of an explicit ordering means that the algorithm must look at the ordering, i. e. it has to use pair-wise comparison of elements, since all the information that is relevant for producing the correct sorting is in the ordering; the elements themselves are irrelevant.

Furthermore, we record the number of comparisons that the algorithm makes by not giving it the relation explicitly, but in the form of a comparison oracle that may be queried.

A sorting algorithm (or ‘sorter’) for a fixed input list (but for arbitrary orderings) can then be written as a recursive datatype that is either the result (the sorted list) or a comparison query consisting of two elements and a continuation that maps the result of the comparison to the remaining computation.

**datatype**  $'a$  *sorter* = *Return 'a list* | *Query 'a 'a bool*  $\Rightarrow$   $'a$  *sorter*

Cormen *et al.* [1] use a similar ‘decision tree’ model where an sorting algorithm for lists of fixed size  $n$  is modelled as a binary tree where each node is a comparison of two elements. They also demand that every leaf in the tree be reachable in order to avoid ‘dead’ subtrees (if the algorithm makes redundant comparisons, there may be branches that can never be taken). Then, the worst-case number of comparisons made is simply the height of the tree.

We chose a subtly different model that does not have this restriction on the algorithm but instead uses a more semantic way of counting the worst-case number of comparisons: We simply use the maximum number of comparisons that occurs for any of the (finitely many) inputs.

We therefore first define a function that counts the number of queries for a specific ordering and then a function that counts the number of queries in the worst case (ranging over a given set of allowed orderings; typically, this will be the set of all linear orders on the list).

**primrec** *count-queries* ::  $('a \times 'a)$  *set*  $\Rightarrow$   $'a$  *sorter*  $\Rightarrow$  *nat* **where**  
 $\text{count-queries } - \text{ (Return } -) = 0$   
|  $\text{count-queries } R \text{ (Query } a \ b \ f) = \text{Suc } (\text{count-queries } R \ (f \ ((a, b) \in R)))$

**definition** *count-wc-queries* ::  $('a \times 'a)$  *set set*  $\Rightarrow$   $'a$  *sorter*  $\Rightarrow$  *nat* **where**  
 $\text{count-wc-queries } Rs \ \text{sorter} = (\text{if } Rs = \{\} \text{ then } 0 \text{ else } \text{Max } ((\lambda R. \text{count-queries } R \ \text{sorter}) \ 'Rs))$

**lemma** *count-wc-queries-empty* [simp]:  $\text{count-wc-queries } \{\} \ \text{sorter} = 0$

*<proof>*

**lemma** *count-wc-queries-aux*:

**assumes**  $\bigwedge R. R \in Rs \implies \text{sorter} = \text{sorter}' R Rs \subseteq Rs'$  *finite*  $Rs'$

**shows**  $\text{count-wc-queries } Rs \text{ sorter} \leq \text{Max } ((\lambda R. \text{count-queries } R (\text{sorter}' R)) ' Rs')$

*<proof>*

**primrec** *eval-sorter* ::  $('a \times 'a) \text{ set} \Rightarrow 'a \text{ sorter} \Rightarrow 'a \text{ list}$  **where**

*eval-sorter* - (*Return*  $ys$ ) =  $ys$

| *eval-sorter*  $R$  (*Query*  $a b f$ ) = *eval-sorter*  $R$  ( $f ((a,b) \in R)$ )

We now get an obvious bound on the maximum number of different results that a given sorter can produce.

**lemma** *card-range-eval-sorter*:

**assumes** *finite*  $Rs$

**shows**  $\text{card } ((\lambda R. \text{eval-sorter } R e) ' Rs) \leq 2 \wedge \text{count-wc-queries } Rs e$

*<proof>*

The following predicate describes what constitutes a valid sorting result for a given ordering and a given input list. Note that when the ordering is linear, the result is actually unique.

**definition** *is-sorting* ::  $('a \times 'a) \text{ set} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow \text{bool}$  **where**

*is-sorting*  $R xs ys \longleftrightarrow (\text{mset } xs = \text{mset } ys) \wedge \text{sorted-wrt } R ys$

## 2.2 Lower bounds on number of comparisons

For a list of  $n$  distinct elements, there are  $n!$  linear orderings on  $n$  elements, each of which leads to a different result after sorting the original list. Since a sorter can produce at most  $2^k$  different results with  $k$  comparisons, we get the bound  $2^k \geq n!$ :

**theorem**

**fixes** *sorter* ::  $'a \text{ sorter}$  **and**  $xs$  ::  $'a \text{ list}$

**assumes** *distinct*: *distinct*  $xs$

**assumes** *sorter*:  $\bigwedge R. \text{linorder-on } (\text{set } xs) R \implies \text{is-sorting } R xs (\text{eval-sorter } R \text{ sorter})$

**defines**  $Rs \equiv \{R. \text{linorder-on } (\text{set } xs) R\}$

**shows** *two-power-count-queries-ge*:  $\text{fact } (\text{length } xs) \leq (2 \wedge \text{count-wc-queries } Rs \text{ sorter} :: \text{nat})$

**and** *count-queries-ge*:  $\log 2 (\text{fact } (\text{length } xs)) \leq \text{real } (\text{count-wc-queries } Rs \text{ sorter})$

*<proof>*

**lemma** *ln-fact-bigo*:  $(\lambda n. \ln (\text{fact } n) - (\ln (2 * \pi * n) / 2 + n * \ln n - n)) \in O(\lambda n. 1 / n)$

**and** *asympt-equiv-ln-fact* [*asympt-equiv-intros*]:  $(\lambda n. \ln (\text{fact } n)) \sim[\text{at-top}] (\lambda n. n * \ln n)$

$\langle proof \rangle$   
**include** *asympt-equiv-syntax*  
 $\langle proof \rangle$

This leads to the following well-known Big-Omega bound on the number of comparisons that a general sorting algorithm has to make:

**corollary** *count-queries-bigomega*:  
**fixes** *sorter* ::  $nat \Rightarrow nat \text{ sorter}$   
**assumes** *sorter*:  $\bigwedge n R. \text{linorder-on } \{..<n\} R \implies$   
 $\text{is-sorting } R [0..<n] (\text{eval-sorter } R (\text{sorter } n))$   
**defines**  $Rs \equiv \lambda n. \{R. \text{linorder-on } \{..<n\} R\}$   
**shows**  $(\lambda n. \text{count-wc-queries } (Rs \ n) (\text{sorter } n)) \in \Omega(\lambda n. n * \ln n)$   
 $\langle proof \rangle$

**end**

## References

- [1] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.