Comparison-based Sorting Algorithms

Manuel Eberl

February 23, 2021

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

1 Linear orderings as relations

theory *Linorder-Relations*
imports
Complex-Main
HOL−Library.Multiset-Permutations
List−Index.List-Index
begin

1.1 Auxiliary facts

lemma *distinct-count-atmost-1'*:
$distinct\ xs = (\forall a.\ count\ (mset\ xs)\ a \leq 1)$
⟨*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*: $total\text{-}on\ B\ R \Longrightarrow A \subseteq B \Longrightarrow total\text{-}on\ A\ R$
⟨*proof*⟩

1.2 Sortedness w.r.t. a relation

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

lemma *sorted-wrt-Nil* [*simp*]: $sorted\text{-}wrt\ R\ []$
⟨*proof*⟩

lemma *sorted-wrt-Cons*: $sorted\text{-}wrt\ R\ (x\ \#\ xs) \longleftrightarrow (\forall y{\in}set\ xs.\ (x,y) \in R) \land sorted\text{-}wrt\ R\ xs$
⟨*proof*⟩

lemma *sorted-wrt-singleton* [*simp*]: $sorted\text{-}wrt\ R\ [x]$
⟨*proof*⟩

lemma *sorted-wrt-many*:
assumes $trans\ R$
shows $sorted\text{-}wrt\ R\ (x\ \#\ y\ \#\ xs) \longleftrightarrow (x,y) \in R \land sorted\text{-}wrt\ R\ (y\ \#\ xs)$
⟨*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*) $\in R$
⟨*proof*⟩

lemma *sorted-wrt-append*:
assumes *sorted-wrt R xs sorted-wrt R ys*
$\bigwedge x\ y.\ x \in$ *set xs* $\Longrightarrow y \in$ *set ys* $\Longrightarrow (x,y) \in R$ *trans R*
shows *sorted-wrt R* (*xs* @ *ys*)
⟨*proof*⟩

lemma *sorted-wrt-snoc*:
assumes *sorted-wrt R xs* (*last xs*, y) $\in R$ *trans R*
shows *sorted-wrt R* (*xs* @ [y])
⟨*proof*⟩

lemma *sorted-wrt-conv-nth*:
sorted-wrt R xs $\longleftrightarrow$ ($\forall\, i\ j.\ i < j \wedge j <$ *length xs* $\longrightarrow$ (*xs*!i, *xs*!j) $\in R$)
⟨*proof*⟩

1.3 Linear orderings

definition *linorder-on* :: *'a set* $\Rightarrow$ (*'a* $\times$ *'a*) *set* $\Rightarrow$ *bool* **where**
linorder-on A R $\longleftrightarrow$ *refl-on A R* $\wedge$ *antisym R* $\wedge$ *trans R* $\wedge$ *total-on A R*

lemma *linorder-on-cases*:
assumes *linorder-on A R* $x \in A$ $y \in A$
shows $x = y \vee ((x, y) \in R \wedge (y, x) \notin R) \vee ((y, x) \in R \wedge (x, y) \notin R)$
⟨*proof*⟩

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*
⟨*proof*⟩

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 $(x,y) \in R$
⟨*proof*⟩

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* $\longleftrightarrow (x,y) \in R$
⟨*proof*⟩

lemma *sorted-wrt-linorder-index-less-iff*:
assumes *linorder-on A R set xs* $\subseteq A$ *sorted-wrt R xs*

$x \in set\ xs$ $y \in set\ xs$
shows $index\ xs\ x < index\ xs\ y \longleftrightarrow (y,x) \notin R$
⟨*proof*⟩

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

1.4 Converting a list into a linear ordering

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

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

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

1.5 Insertion sort

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

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

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

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

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

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

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

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

⟨*proof*⟩

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

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

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

lemma *sorted-wrt-linorder-unique*:
assumes *linorder-on A R mset xs* = *mset ys sorted-wrt R xs 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**
sorted-wrt-list-of-set R A =
(*if finite A then* (*THE xs. set xs* = *A* $\wedge$ *distinct xs* $\wedge$ *sorted-wrt R xs*) *else* [])

lemma *mset-remdups*: *mset* (*remdups xs*) = *mset-set* (*set xs*)
⟨*proof*⟩

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

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

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

lemma *sorted-wrt-list-of-set-eqI*:
assumes *linorder-on B R A* $\subseteq$ *B set xs* = *A distinct xs sorted-wrt R xs*
shows *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**
linorder-rank R A x = card $\{y{\in}A{-}\{x\}.\ (y,x) \in R\}$

lemma *linorder-rank-le*:
assumes *finite A*
shows *linorder-rank R A x* $\leq$ *card A*
⟨*proof*⟩

lemma *linorder-rank-less*:
assumes *finite A* $x \in A$
shows *linorder-rank R A x* $<$ *card A*
⟨*proof*⟩

lemma *linorder-rank-union*:
assumes *finite A finite B* $A \cap B = \{\}$
shows *linorder-rank R* $(A \cup B)$ *x* = *linorder-rank R A x* + *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* = (*if* $x \neq y \wedge (y,x) \in R$ *then 1 else 0*)
⟨*proof*⟩

lemma *linorder-rank-insert*:
assumes *finite A* $y \notin A$
shows *linorder-rank R* (*insert y A*) *x* =
(*if* $x \neq y \wedge (y,x) \in R$ *then 1 else 0*) + *linorder-rank R A x*
⟨*proof*⟩

lemma *linorder-rank-mono*:
assumes *linorder-on B R finite A* $A \subseteq B$ $(x, y) \in R$
shows *linorder-rank R A x* $\leq$ *linorder-rank R A y*
⟨*proof*⟩

lemma *linorder-rank-strict-mono*:
assumes *linorder-on B R finite A* $A \subseteq B$ $y \in A$ $(y, x) \in R$ $x \neq y$
shows *linorder-rank R A y* $<$ *linorder-rank R A x*
⟨*proof*⟩

lemma *linorder-rank-le-iff*:
assumes *linorder-on B R finite A* $A \subseteq B$ $x \in A$ $y \in A$
shows *linorder-rank R A x* $\leq$ *linorder-rank R A y* $\longleftrightarrow$ $(x, y) \in R$

$\langle proof \rangle$

lemma *linorder-rank-eq-iff*:
assumes *linorder-on B R finite A* $A \subseteq B$ $x \in A$ $y \in A$
shows *linorder-rank R A x* = *linorder-rank R A y* $\longleftrightarrow$ $x = y$
$\langle proof \rangle$

lemma *linorder-rank-set-sorted-wrt*:
assumes *linorder-on B R* *set xs* $\subseteq B$ *sorted-wrt R xs* $x \in$ *set xs* *distinct xs*
shows *linorder-rank R* (*set xs*) *x* = *index xs x*
$\langle proof \rangle$

lemma *bij-betw-linorder-rank*:
assumes *linorder-on B R finite A* $A \subseteq B$
shows *bij-betw* (*linorder-rank R A*) *A* {..<*card A*}
$\langle proof \rangle$

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*}
$\langle proof \rangle$

corollary *card-finite-linorders*:
assumes *finite A*
shows *card* {*R. linorder-on A R*} = *fact* (*card A*)
$\langle proof \rangle$

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 ⇒ '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* × *'a*) *set* ⇒ *'a sorter* ⇒ *nat* **where**
  count-queries - (*Return* -) = *0*
| *count-queries R* (*Query a b f*) = *Suc* (*count-queries R* (*f* ((*a*, *b*) ∈ *R*)))

definition *count-wc-queries* :: (*'a* × *'a*) *set set* ⇒ *'a sorter* ⇒ *nat* **where**
  count-wc-queries Rs sorter = (*if Rs* = {} *then 0 else Max* ((λ*R*. *count-queries R sorter*) ‘ *Rs*))

lemma *count-wc-queries-empty* [*simp*]: *count-wc-queries* {} *sorter* = *0*
  ⟨*proof*⟩

lemma *count-wc-queries-aux*:
  assumes ⋀*R*. *R* ∈ *Rs* ⟹ *sorter* = *sorter' R Rs* ⊆ *Rs' finite Rs'*
  shows *count-wc-queries Rs sorter* ≤ *Max* ((λ*R*. *count-queries R* (*sorter' R*)) ‘ *Rs'*)
⟨*proof*⟩

primrec *eval-sorter* :: $('a \times 'a)$ *set* $\Rightarrow$ *'a sorter* $\Rightarrow$ *'a 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 *card* ((λ*R*. *eval-sorter R e*) ‘ *Rs*) $\leq$ *2* ^ *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* :: $('a \times 'a)$ *set* $\Rightarrow$ *'a list* $\Rightarrow$ *'a list* $\Rightarrow$ *bool* **where**
is-sorting R xs ys $\longleftrightarrow$ (*mset xs* = *mset ys*) $\wedge$ *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 sorter* **and** *xs* :: *'a list*
assumes *distinct*: *distinct xs*
assumes *sorter*: $\bigwedge$*R*. *linorder-on* (*set xs*) *R* $\Longrightarrow$ *is-sorting R xs* (*eval-sorter R sorter*)
defines *Rs* $\equiv$ {*R*. *linorder-on* (*set xs*) *R*}
shows *two-power-count-queries-ge*: *fact* (*length xs*) $\leq$ (*2* ^ *count-wc-queries Rs sorter* :: *nat*)
and *count-queries-ge*: *log 2* (*fact* (*length xs*)) $\leq$ *real* (*count-wc-queries Rs sorter*)
⟨*proof*⟩

lemma *ln-fact-bigo*: (λ*n*. *ln* (*fact n*) $-$ (*ln* (*2* $*$ *pi* $*$ *n*) / *2* + *n* $*$ *ln n* $-$ *n*)) $\in$ *O*(λ*n*. *1* / *n*)
and *asymp-equiv-ln-fact* [*asymp-equiv-intros*]: (λ*n*. *ln* (*fact n*)) $\sim$[*at-top*] (λ*n*. *n* $*$ *ln n*)
⟨*proof*⟩
include *asymp-equiv-notation*
⟨*proof*⟩

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* ⇒ *nat sorter*
assumes *sorter*: $\bigwedge$*n R. linorder-on* {..<*n*} *R* $\Longrightarrow$
is-sorting R [*0*..<*n*] (*eval-sorter R* (*sorter n*))
defines *Rs* ≡ λ*n*. {*R. linorder-on* {..<*n*} *R*}
shows (λ*n*. *count-wc-queries* (*Rs n*) (*sorter n*)) ∈ Ω(λ*n*. *n* ∗ *ln n*)
⟨*proof*⟩

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.