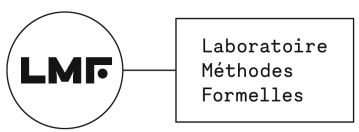
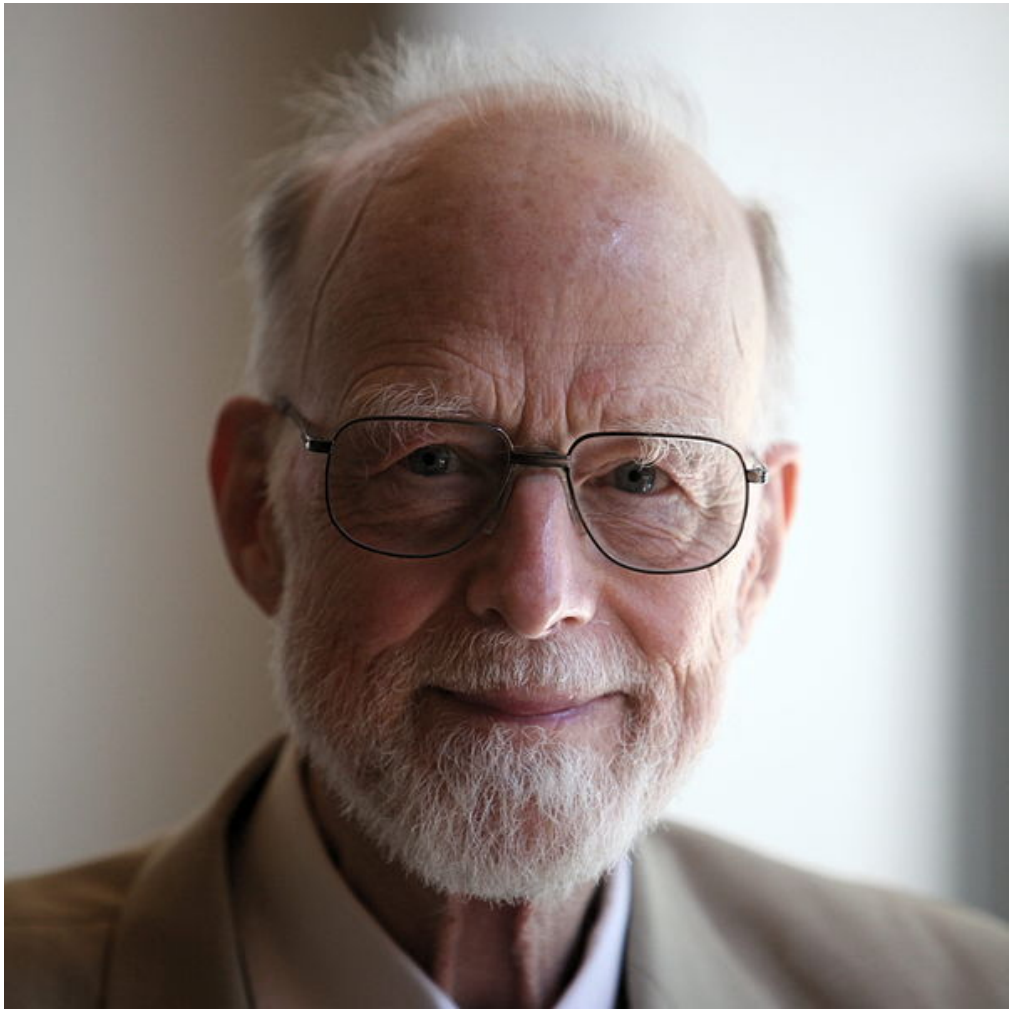
64 versions of algorithm 64, verified

Jean-Christophe Filliâtre

IFIP WG 1.9/2.15
April 2026
Torino, Italy



Sir Tony Hoare



1934–2026

1980
Turing
Award
Lecture

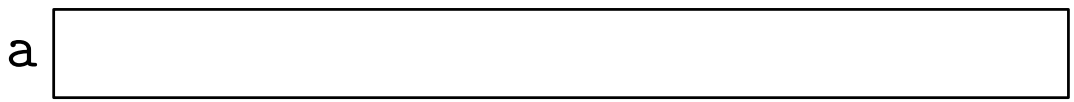
The Emperor's Old Clothes

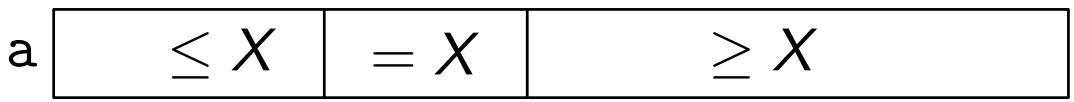
CHARLES ANTONY RICHARD HOARE
Oxford University, England

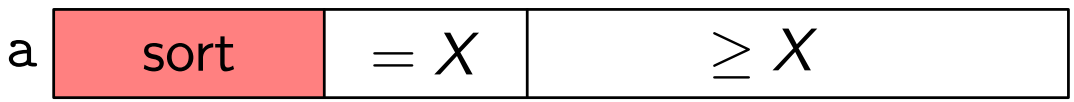
Around Easter 1961, a course on ALGOL60 was offered in Brighton [...]. It was there that I first learned about recursive procedures and saw how to program the sorting method which I had earlier found such difficulty in explaining. It was there that I wrote the procedure, immodestly named QUICKSORT, on which my career as a computer scientist is founded.

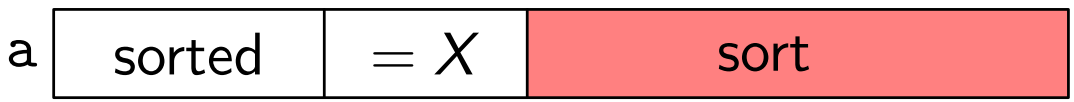
Around Easter 1961, a course on ALGOL60 was offered in Brighton [...]. It was there that I first learned about recursive procedures and saw how to program the sorting method which I had earlier found such difficulty in explaining. It was there that I wrote the procedure, immodestly named QUICKSORT, on which my career as a computer scientist is founded.

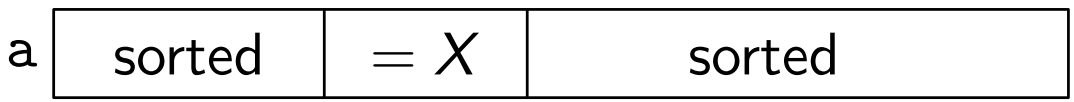
Due credit must be paid to the genius of the designers of ALGOL 60 who included recursion in their language and enabled me to describe my invention so elegantly to the world. I have regarded it as the highest goal of programming language design to enable good ideas to be elegantly expressed.


a


a
$\leq X$
$= X$
$\geq X$


a
sort
$= X$
$\geq X$

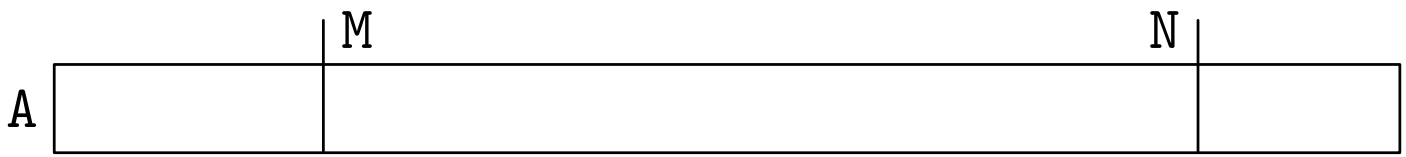

a
sorted
$= X$
sort


a
sorted
$= X$
sorted

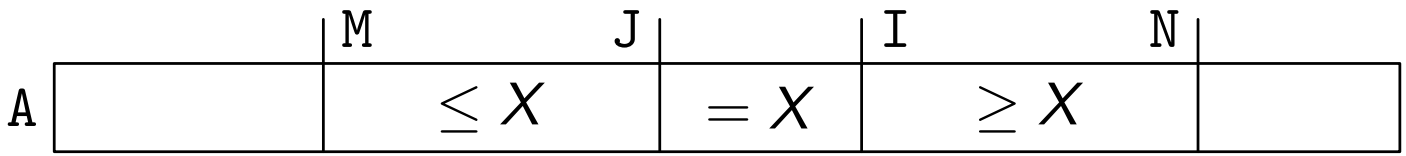
1. so many ways to implement quicksort
2. verification

Algorithm 63: Partition and Algorithm 64: Quicksort
Communications of the ACM (CACM), 4(7), July 1961

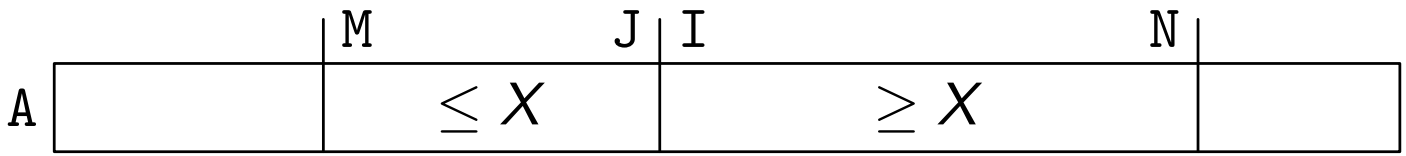
Algorithm 63: Partition and Algorithm 64: Quicksort
Communications of the ACM (CACM), 4(7), July 1961



Algorithm 63: Partition and Algorithm 64: Quicksort
Communications of the ACM (CACM), 4(7), July 1961



Algorithm 63: Partition and Algorithm 64: Quicksort
Communications of the ACM (CACM), 4(7), July 1961

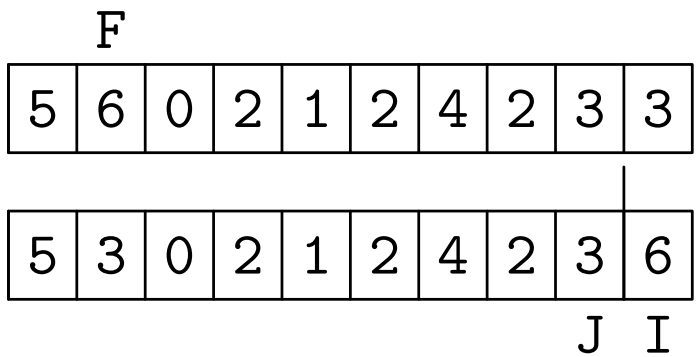


Hoare's partition is sound — it ensures a correct and terminating quicksort — but it is suboptimal

example

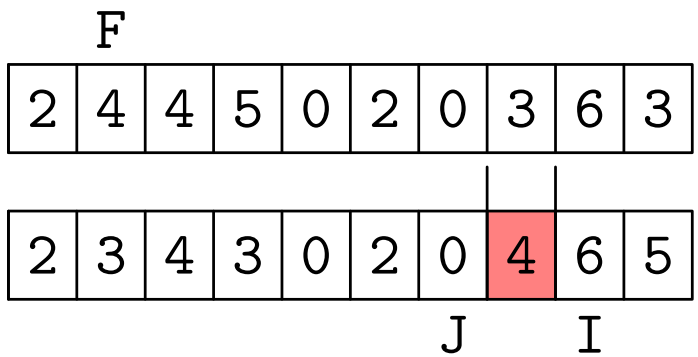
Hoare's partition is sound — it ensures a correct and terminating quicksort — but it is suboptimal

example



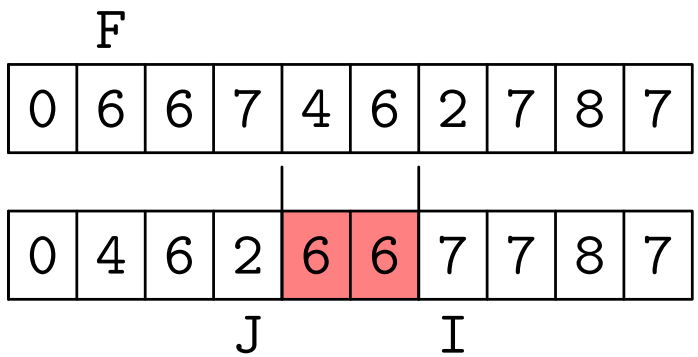
Hoare's partition is sound — it ensures a correct and terminating quicksort — but it is suboptimal

example



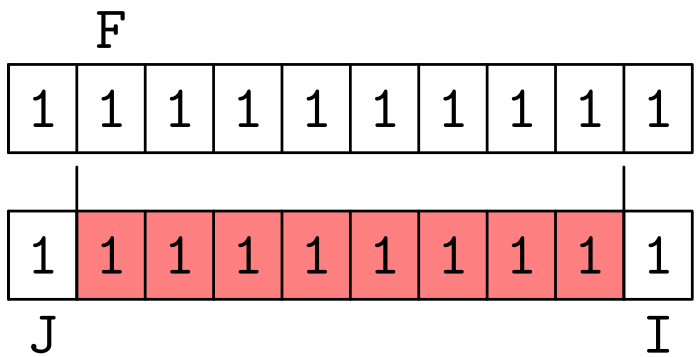
Hoare's partition is sound — it ensures a correct and terminating quicksort — but it is suboptimal

example



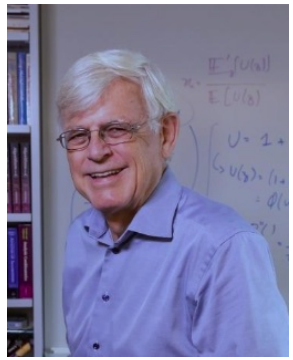
Hoare's partition is sound — it ensures a correct and terminating quicksort — but it is suboptimal

example



later came Bob Sedgewick

PhD Thesis, Stanford University, 1975

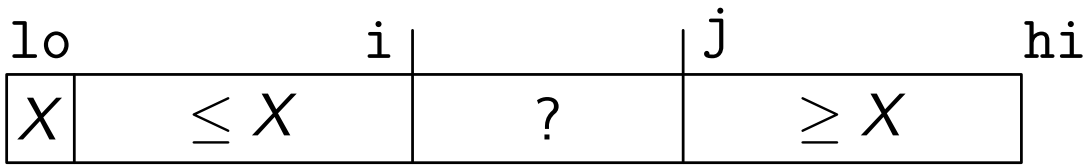


QUICKSORT

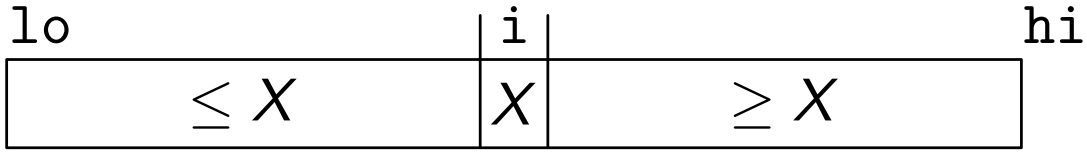
by Robert Sedgewick

Abstract

A complete study is presented of the best general purpose method for sorting by computer: C. A. R. Hoare's Quicksort algorithm. Special attention is paid to the methods of mathematical analysis which are used to demonstrate the practical utility of the algorithm. The most efficient known form of Quicksort is developed, and exact formulas are derived for the average, best case, and worst case running times. The


lo
i
j
hi
X
≤ X
?
≥ X


Algorithms
FOURTH EDITION

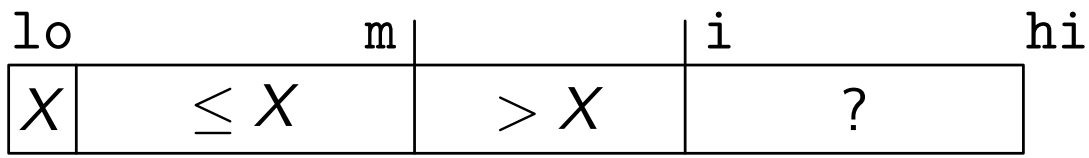

lo
i
hi
$\leq X$
X
$\geq X$


Algorithms
FOURTH EDITION

Bentley's partition schemes

popularized by *Programming Pearls*, Jon Bentley, 1986

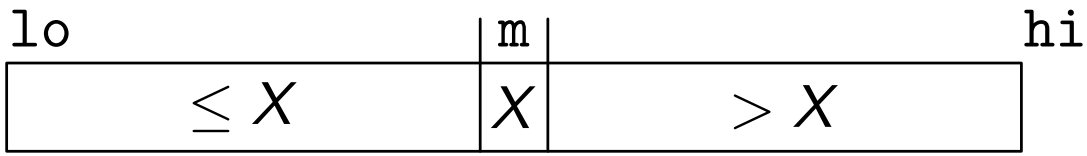
2-way partition



Bentley's partition schemes

popularized by *Programming Pearls*, Jon Bentley, 1986

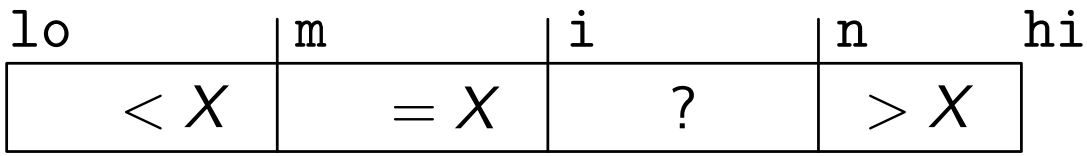
2-way partition



Bentley's partition schemes

popularized by *Programming Pearls*, Jon Bentley, 1986

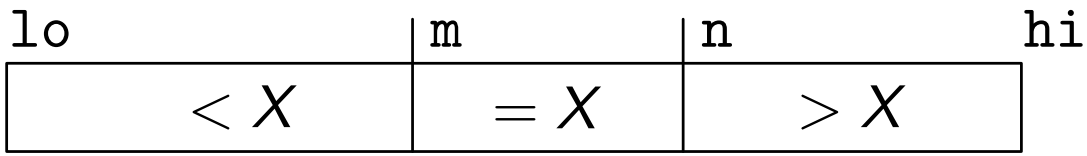
3-way partition



Bentley's partition schemes

popularized by *Programming Pearls*, Jon Bentley, 1986

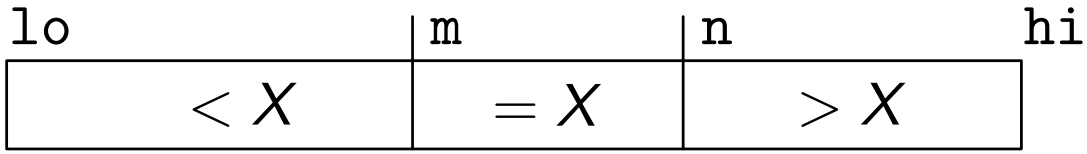
3-way partition



Bentley's partition schemes

popularized by *Programming Pearls*, Jon Bentley, 1986

3-way partition



this is Dijkstra's Dutch national flag!

Engineering a Sort Function

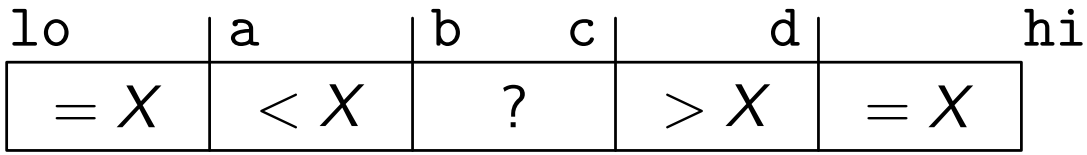
JON L. BENTLEY
M. DOUGLAS McILROY
AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, U.S.A.

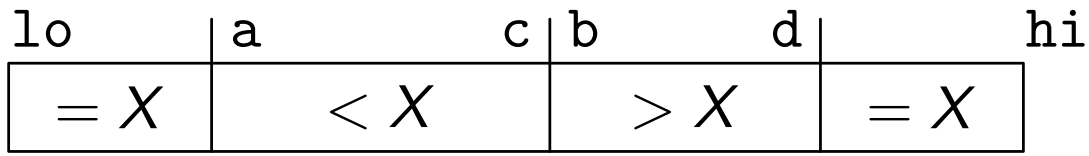
SUMMARY

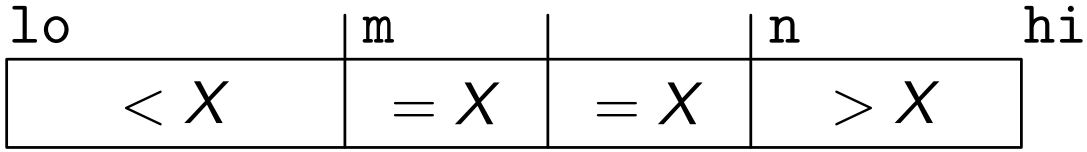
We recount the history of a new `qsort` function for a C library. Our function is clearer, faster and more robust than existing sorts. It chooses partitioning elements by a new sampling scheme; it partitions by a novel solution to Dijkstra's Dutch National Flag problem; and it swaps efficiently. Its behavior was assessed with timing and debugging testbeds, and with a program to certify performance. The design techniques apply in domains beyond sorting.

KEY WORDS Quicksort Sorting algorithms Performance tuning Algorithm design and implementation Testing

(Software—Practice and Experience, 23(11), 1993)


lo
a
b
c
d
hi
$= X$
$< X$
?
$> X$
$= X$


lo
a
c
b
d
hi
$= X$
$< X$
$> X$
$= X$


lo
m
n
hi
$< X$
$= X$
$= X$
$> X$

in addition to the partition scheme,

- pivot choice
  - shuffle the whole array before sorting
  - random choice of the pivot (possibly swapping it with `a[lo]`)
  - median of 3, median of 9, etc.
- turn to insertion sort when `hi` $-$ `lo` is small (e.g. ≤ 7)
- ensure log space complexity
  - first recursive call on the smaller half
  - the second recursive call is tail and thus optimized
    - or you make a loop is your compiler is stupid

verification

let's verify several implementations of Quicksort

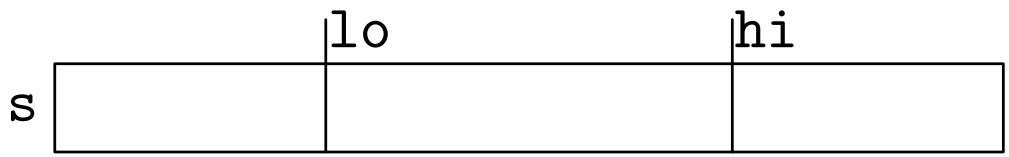
- for safety, including termination
- for correctness: it sorts and it only permutes
- but not for complexity

using the Why3 program verifier

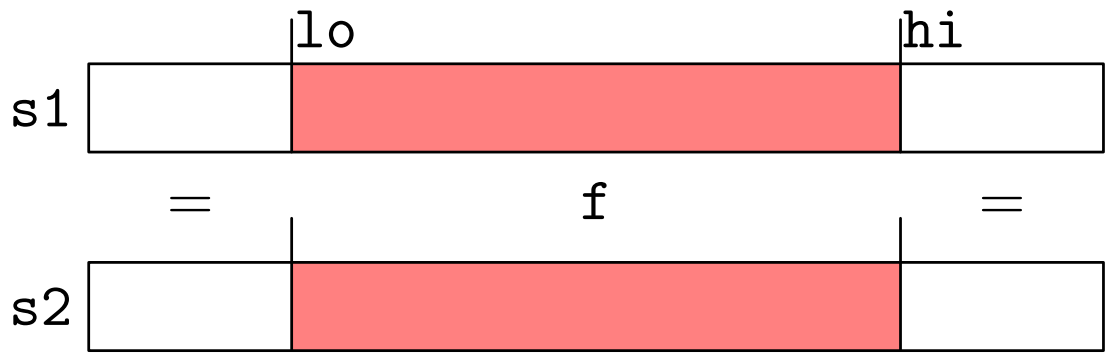
- its own programming language, WhyML
- a WP-based verification condition generator
- several SMT solvers as backend

```
type elt

val predicate le elt elt

axiom le_refl : forall x.     le x x
axiom le_trans: forall x y z. le x y -> le y z -> le x z
axiom le_total: forall x y.   le x y \/ le y x
```

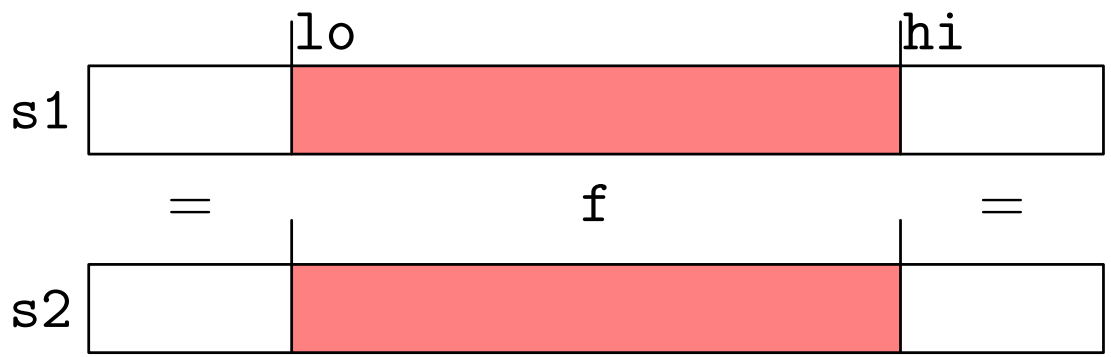
```
predicate sorted (s: seq elt) (lo hi: int) =
  0 <= lo <= hi <= size s /\
  forall i j. lo <= i <= j < hi -> le s[i] s[j]
```



```
predicate permut (s1 s2: seq elt) (lo hi: int) =
  0 <= lo <= hi <= size s1 = size s2 /\
  (forall i. 0  <= i < lo      -> s1[i] = s2[i]) /\
  (forall i. hi <= i < size s1 -> s1[i] = s2[i]) /\
  exists f: int -> int. bijection f s1 s2 lo hi
```



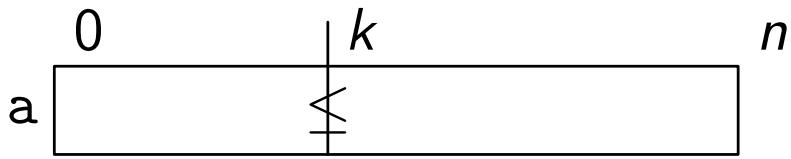
```
predicate bijection (f: int -> int)
                    (s1 s2: seq elt) (lo hi: int) =
  (forall i. lo <= i < hi ->
     lo <= f i < hi /\ s1[i] = s2[f i]) /\
  (forall j. lo <= j < hi ->
     exists i. lo <= i < hi /\ j = f i)
```



we show that the predicate `permut`

- is reflexive
- is transitive
- extends to a larger interval

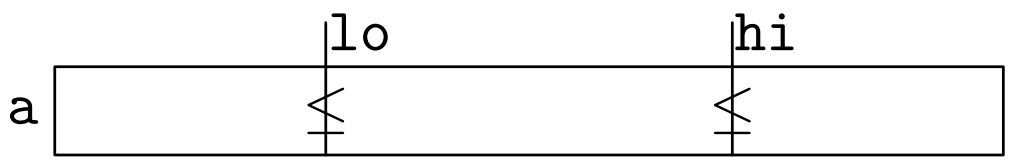
one predicate to verify them all



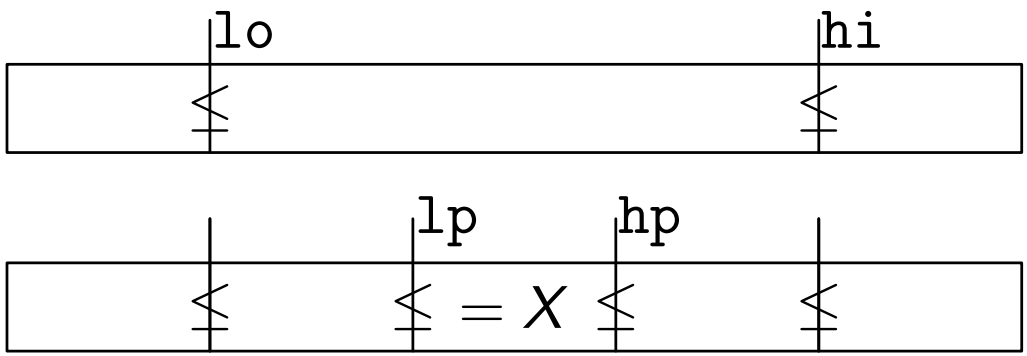

```
predicate cutter (s: seq elt) (k: int) =
  0 <= k <= size s /\
  forall i j. 0 <= i < k <= j < size s -> le s[i] s[j]
```

both `partition` and `quicksort` require and preserve

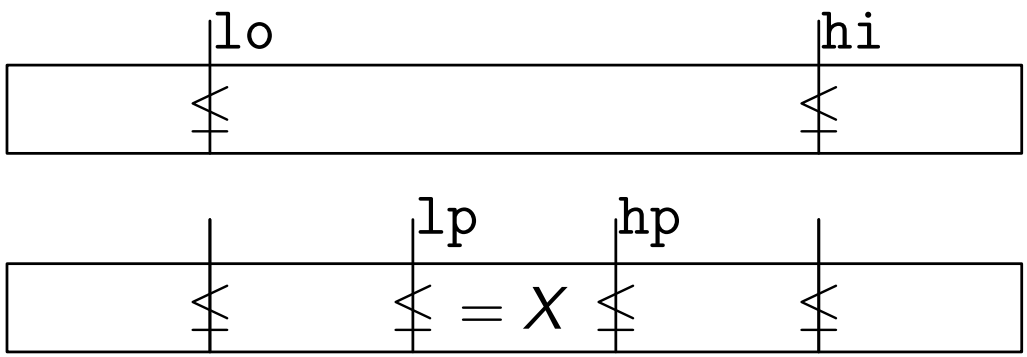
```
cutter a lo
cutter a hi
```



```
val partition (a:array elt) (lo hi:int) : (lp:int, hp:int)
  requires { 0 <= lo < hi-1 < size a }
  requires { cutter a lo }
  requires { cutter a hi }
  writes   { a }
  ensures  { lo <= lp < hp <= hi }
  ensures  { permut (old a) a lo hi }
  ensures  { cutter a lp }
  ensures  { forall i. lp <= i < hp -> a[i] = a[lp] }
  ensures  { cutter a hp }
```



```
val quicksort (a: array elt) (lo hi: int) : unit
  requires { 0 <= lo <= hi <= size a }
  requires { cutter a lo }
  requires { cutter a hi }
  variant  { hi - lo }
  ensures  { permut (old a) a lo hi }
  ensures  { sorted         a lo hi }
```

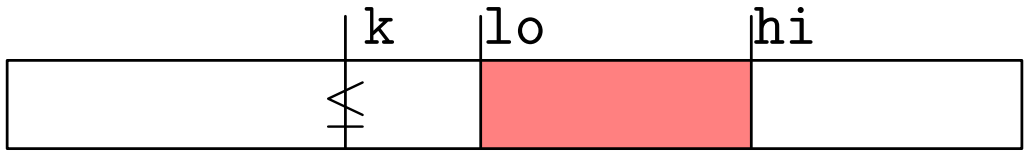


using these specifications, I could verify

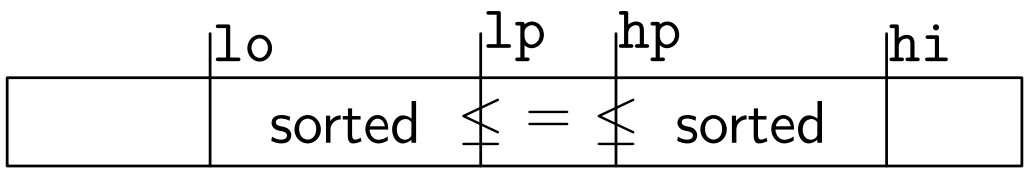
- several partition schemes
  - Sedgewick's
  - Bentley's 2-way
  - Bentley's 3-way
- several variants of `quicksort`
  - with and without shuffle
  - with and without insertion sort cutoff
  - with and without log space
    - with and without TCO

only two lemmas were needed

- we can permute on either side of a cutter



- `quicksort`'s conclusion



thank you

all figures drawn with `mlpost`, an OCaml frontend to MetaPost

```
Arrays.create ~left:(tt "s") 14
  |> tick  4. |> index  4. (tt "lo")
  |> tick 10. |> index 10. (tt "hi")
  |> Box.draw |> Metapost.emit "file"
```