# Quickselect (Recursive)

## Overview

The quickselect algorithm[[1]](#footnote-1) selects the n-th smallest element from an unordered array. It is related to quicksort, as it also chooses a pivot element according to which the array will be partitioned into two segments, separated by the pivot: one with all elements less than the pivot, the other with all elements greater or equal to the pivot. Quickselect then recurses into the side where the n-th smallest element must reside. The algorithm terminates if the pivot is the n-th element in the array, which implies that it is the n-th smallest element. Quickselect has an average complexity of O(*n*).

## Silver Encoding

The algorithm has been encoded using three methods. Variable arr is of type Array[Int], pw is of type Seq[Int], all other variables are Int-typed.

* swap(arr, i, j): Swaps arr[i] and arr[j]
* partition(arr, left, right, pivotIndex) returns (storeIndex, pw): Partitions the segment arr[left..right] according to the element at a[pivotIndex]. It returns storeIndex, which points to where the pivot element can be found after the partitioning, i.e. a[storeIndex] == old(a[pivotIndex]), and – for verification reasons only – a permutation witness pw, such that a[i] == old(a[pw[i]]). That is, the element that is now at a[i] was previously at a[pw[i]].
* select\_rec(arr, left, right, n) returns (storeIndex, pw): Recursive implementation of the main routine of quickselect; it selects the n-th smalltest element from arr[left..right], and returns its index storeIndex, and a permutation witness pw. It selects a pivot, partitions arr[left..right] (using partition), and it then branches over the position of the pivot, i.e. over storeIndex:
  + If storeIndex == n, then the pivot is the n-th smallest element, and quickselect terminates, returning the storeIndex and the permutation pw returned obtained from the invocation of partition.
  + If n < pivotIndex, the n-th smallest element must be on the left of the pivot, and quickselect recurses into the left partition, i.e. select\_rec(arr, left, pivotIndex – 1, n). The storeIndex returned from the recursive invocation points to the desired element, and quickselect terminates.

The final permutation witness will be a combination of the witness pwPar obtained from the invocation of partition, and the witness pwRec obtained from the recursive invocation of select\_rec: An element that is now at arr[i] was initially at

* arr[pwPar[pwRec[i]]] if left ≤ i ≤ pivotIndex – 1
* arr[pwPar[i]] if pivotIndex ≤ i ≤ right
  + Analogous for the case where n > pivotIndex.

## Encoding Variations

* arrays\_quickselect.sil: The default encoding. The permutation witness pw covers the array segment arr[..right], which simplifies indexing, e.g. pw[i] := pwPar[pwRec[i]].
* arrays\_quickselect\_rec\_index-shifting.sil: A variation of the default encoding, where pw only covers arr[left..right]. This complicates indexing, e.g. the previous assignment becomes  
  pw[i] := pwPar[pwRec[i - (pivotIndex + 1 - left)] - left].

1. <https://en.wikipedia.org/wiki/Quickselect> [↑](#footnote-ref-1)