PROVING QUICKSORT CORRECT

1. MAIN PROGRAM

1.1. Specifications.

QuickSort\((A, n)\)

Inputs:
\(n\) an integer, \(n \geq 1\),
\(A\) an array with elements \(A[1], \ldots, A[n]\).

Effect:
The elements are permuted such that \(A[1] \leq \ldots \leq A[n]\).

1.2. Implementation. Our implementation is non-recursive and uses a stack to store index pairs. The index pair \((L, R)\) represents the unsorted subarray \(A[L], \ldots, A[R]\).

\[
S \leftarrow \text{empty}
\]

\[
push(S, (1, n))
\]

do

\[
(L, R) \leftarrow \text{pop}(S)
\]

if \((L < R)\)

\[
a \leftarrow A[L]
\]

\[
P \leftarrow \text{Partition}(A, L, R, a)
\]

\[
push(S, (L, P))
\]

\[
push(S, (P + 2, R))
\]

while not(stack_empty(S))

The subroutine Partition has the following specifications.

\[
p' \leftarrow \text{Partition}(A, p, q, a)
\]

Inputs:
\(p, q\) integers, \(p \leq q\).
\(A[p], \ldots, A[q]\) an array of integers.
\(a\) an integer.

Outputs:
\(p'\) an integer.

Effect:
\(A[p], \ldots, A[q]\) is a permutation of the input array
such that \(A[p], \ldots, A[p'] < a\) and \(A[p' + 1], \ldots, A[q] \geq a\).
This notation is meant to imply that if \(p' < p\) then there are
no array elements < \(a\), and if \(p' + 1 > q\) then there are
no array elements \(\geq a\).
Note that our *Quicksort* implementation pushes subarrays on the stack even if they have only one element or no elements at all. The latter case arises whenever \( a = \min(A[L], \ldots, A[R]) \) and, hence, \( P < L \). Subsequent executions of the loop body remove such arrays from the stack without any further processing.

1.3. **Proof of partial correctness.**

**Definition 1.1.** Let \( R, S \) be sets of real numbers. Then

\[
R < S \Leftrightarrow \forall r \in R \forall s \in S \ r < s.
\]

Note that, for any set \( S, \emptyset < S \).

**Definition 1.2.** Every state of the stack \( S \) in *Quicksort* induces a partition \( \Pi_S \) of the index set \( \{1, \ldots, n\} \). The partition consists of the sets of indices of each unsorted subarray on the stack and of one-element sets for those elements that are already in their final position.

\[
\Pi_S := \left\{ \{i \mid L \leq i \leq R\} \mid (L, R) \in S \right\} \cup \left\{ \{i\} \mid 1 \leq i \leq n \land \forall (L, R) \in S \ (L \leq R \Rightarrow i \notin \{L, \ldots, R\}) \right\}.
\]

Note that, if the stack contains an empty array \( (L, R) \) with \( R < L \) then \( \Pi_S \) contains the empty set as an element. For every index \( i \in \{1, \ldots, n\} \) let \( \pi_i \) denote the element of \( \Pi_S \) that contains \( i \).

**Theorem 1.3.** Before “do” we have

\[
\forall 1 \leq i < j \leq n \quad \pi_i < \pi_j \Rightarrow A[i] \leq A[j].
\]

**Proof.** The only element on the stack is \( (1, n) \), so \( \Pi_S = \{\{1, \ldots, n\}\} \). Therefore, the assertion \( \pi_i < \pi_j \) is false for all \( 1 \leq i < j \leq n \), hence the implication in (1) is true for all \( 1 \leq i < j \leq n \), and so (1) is true. \( \square \)

**Theorem 1.4.** Let (1) hold before one execution of the body of the loop. Let \( \Pi_S \) be the partition associated with the stack \( S' \) after the execution of the body of the loop, and, for every index \( i \in \{1, \ldots, n\} \), let \( \pi'_i \) denote the element of \( \Pi_{S'} \) that contains \( i \). Then

\[
\forall 1 \leq i < j \leq n \quad \pi'_i < \pi'_j \Rightarrow A[i] \leq A[j].
\]

**Proof.** If \( L \geq R \) then \( \Pi_{S'} = \Pi_S \) or \( \Pi_{S'} = \Pi_S - \{\emptyset\} \), and the assertion coincides with (1). Now assume \( L < R \). Then we have \( \Pi_{S'} = (\Pi_S - \{\{L, \ldots, R\}\}) \cup \{\{L, \ldots, P\}, \{P + 1\}, \{P + 2, \ldots, R\}\} \).

Let \( 1 \leq i < j \leq n \) be such that \( \pi'_i < \pi'_j \). Since each of \( \pi'_i \) and \( \pi'_j \) is either an element of \( \Pi_S - \{\{L, \ldots, R\}\} \) or equal to \( \{L, \ldots, P\} \) or \( \{P + 1\} \) or \( \{P + 2, \ldots, R\} \), the condition (1) and the post-condition of *Partition* guarantee together that we have \( A[i] \leq A[j] \) in every case. \( \square \)

**Theorem 1.5.** After the while-loop,

\[
\forall 1 \leq i < j \leq n \quad A[i] \leq A[j].
\]

**Proof.** After the while-loop we have \( (1) \land S = \emptyset \). Since \( S = \emptyset \) implies \( \Pi_S = \{\{1\}, \ldots, \{n\}\} \), we have, for all \( 1 \leq i < j \leq n \), \( \pi_i < \pi_j \). Now (1) implies \( A[i] \leq A[j] \). \( \square \)
1.4. **Termination.** Each execution of the body of the loop removes either an empty array or a one-point array from the stack or it removes the index $P + 1$ from the set of indices that lie in some array that is on the stack. So, eventually, this set of indices will become empty.

1.5. **Remark.**

(1) If the program is modified so that it always pushes the larger subarray first on the stack, then the stack will hold at most $\lfloor \log_2 n \rfloor + 3$ index pairs at any time.

(2) Picking the partition element $a$ as follows improves the chances of getting a good computing time

$$a \leftarrow \text{median}(A[L], A[(L + R)/2], A[R]).$$