CSE 101 Notes - Quicksort Correctness Proof

Refer to the QUICKSORT and PARTITION algorithms on page 154. This correctness proof will also apply to RANDOMIZED-QUICKSORT.

There are 2 parts to the proof. First, we show that QUICKSORT is correct, assuming that PARTITION is correct. Then, we verify the correctness of PARTITION.

First, assume that PARTITION is correct, defined as: it permutes its input and returns q, such that for all k_1 and k_2 satisfying $p \leq k_1 \leq q < k_2 \leq r$, $A[k_1] \leq A[k_2]$. Furthermore, the partition is nontrivial: $p \leq q < r$.

The QUICKSORT correctness proof is by induction on size of the input array, $n = r - p + 1$.

For the base case of $n \leq 1$, $p \geq r$ so that QUICKSORT doesn't execute anything past line 1; it doesn't touch A. This is correct, since an array with zero or one elements is trivially already sorted.

Now, assume that QUICKSORT is correct for inputs of size less than n (this is the induction hypothesis), and show it is correct for inputs of size n. Since we are assuming PARTITION is correct and that it returns a q satisfying $p \leq q < r$, both line 3 and line 4 make recursive calls on arrays with sizes between 1 and $n-1$. We can thus apply the induction hypothesis to these recursive calls, showing that $A[p \dots q]$ and $A[q+1 \dots r]$ are correctly sorted. So, $A[p \dots q]$ is sorted in increasing order, $A[q] \leq A[q+1]$ by the correctness of partition, and $A[q+1 \dots r]$ are sorted in increasing order. Thus, $A[p \dots r]$, the input to QUICKSORT, is correctly sorted.

Now we show that PARTITION is correct. We start with two basic facts: that i and j stay in bounds and that the partition is nontrivial. Then we show that the partition is correct.

We show that i and j stay in bounds. Note that j is decremented to r before it is ever used to access A, and i is similarly incremented to p before it is ever used to access A. After that point, the only changes to i and j are lines 5 and 7, which decrement j and increment i: so A is always indexed with $i \geq p$ and $j \leq r$. This shows that i and j each respect one of the two boundaries of the array.

For the other boundary, consider the first pass through PARTITION's outer loop. i stops at $i = p$ since $A[p] = x$, triggering the **until** condition in line 8. When decrementing in lines 5-6, j might stop (by triggering the **until** condition at some $j > p$, but if it doesn't then it certainly stops at $j = p$ since $A[p] = x$ triggers the **until** condition in line 6. Thus, on the first time that line 9 is reached, $i = p$ and $p \leq j \leq r$.

If $j < r$ after this first pass, then $A[r] > x$ (or else the **until** in line 6 would have been triggered when $j = r$). Otherwise, $j = r$ when line 9 is reached: $i < j$ since A has at least 2 elements, so the exchange at line 10 is performed and $A[r]$ now holds x. In either of these cases, $A[r] \geq x$ after the first pass through the main loop.

If the exchange is not performed on the first pass (because $i = j$ when line 9 is reached), then $A[p] = x$ as was the case in the original input. Otherwise, line 10 places into $A[p]$ the contents of $A[j]$, where $A[j] \leq x$ by the **until** condition in line 6. In either of these cases, $A[p] \leq x$ after the first pass through the main loop.

After this first pass through the loop, any subsequent pass will continue incrementing i and decrementing j, such that $i > p$ and $j < r$ for these. $A[p]$ and $A[r]$ will therefore not be touched by line 10 again, unless we reach a state where $j = p$ or $i = r$. Before the first time where we reach such a state, note that j cannot be decremented below p, because $A[p] \leq x$ (as shown above) triggers the **until** in line 6. Similarly, i cannot be incremented above r, because $A[r] \geq x$ triggers the **until** in line 8. The first time when $j = p$, since $i \geq p$, $j \leq i$ and PARTITION terminates. Similarly, the first time $i = r$, since $j \leq r$, $j \leq i$ and PARTITION terminates. Thus, after the first pass, i cannot move past r and j cannot move past p. As noted above, i will never be used to access A when $i < p$, and j will never be used when $j > r$. Combining these, we see that i and j stay in-bounds.

Note that, since each pass through the main loop of PARTITION, i gets incremented at least once (and is never decreased) and j gets decremented at least once (and is never increased), this shows that PARTITION must eventually terminate.

Now, we show that the partition is nontrivial: we have $j \geq p$ upon termination by the above argument, so the lower partition is of size at least 1. But we need $j < r$ upon termination to show that the lower partition does not contain all n elements. If after the first pass, $j < r$, then the j that eventually gets returned will be less than r, since j is only decremented after that first pass. Otherwise, $j = r$ after the first pass. $i = p$ after the first pass, as discussed above, so $i < j$ since $r > p$. Thus, a second pass is executed, and j gets decremented at least once in line 5, yielding $j < r$, and again the j that eventually gets returned will be less than r. In either case, we get a nontrivial partition with $j < r$ upon termination of PARTITION.

Finally, we need to show that the partition that gets returned is correct. We do this by induction on the number of passes through the main loop, using an *invariant* that stays true each time through the loop, and upon termination yields our desired conclusion of correctness. The invariant here specifies that all elements that i has already swept pass are correctly on the low side of the partition, and all elements that j has already swept pass are correctly on the high side of the partition. In particular, we show that the following invariant holds just before the first pass through the main loop (between lines 3 and 4), and then after each pass through the main loop (between lines 10 and 11, so that we can show it holds before returning the last time through the main loop):

$$
\begin{array}{l}
[A \text{ is a permutation of the input } \& \\
A[k] \leq x \text{ for } p \leq k < i \ \& \\
A[k] \geq x \text{ for } r \geq k > j \ \& \\
(((i < j)\&(i \geq p)\&(j \leq r)) \Rightarrow ((A[i] \leq x)\&(A[j] \geq x)))]
\end{array}
$$

The last line of the invariant says that, if we have had at least one pass through the main loop, but this was not the final pass through the loop, then the correctness condition also holds at the *current* values of i and j.

For the base case, note that after line 3 but before we have executed the main loop the first time, the input is unchanged (so it is a permutation of the input), and there are no values of k for which the remainder of the invariant needs to hold; so the invariant is true at that point.

Now show that the invariant holds at the end of a pass through the main loop, assuming it held after the previous pass through the main loop (this assumption is the inductive hypothesis). First, since i and j never get out of bounds, and exchange operations are the only changes to A, A remains a permutation of the input. This establishes that the first line of the invariant remains true.

Now, say that i started at i_0 and j at j_0 at the start of this pass. So $A[k] \leq x$ for $p \leq k \leq i_0$ by the inductive hypothesis, and similarly $A[k] \geq x$ for $r \geq k \geq j_0$. By the conditions on the **until**'s and the fact that i and j remain in bounds (shown above), we have $A[k] \leq x$ for $i_0 < k < i$ and $A[k] \geq x$ for $j_0 > k > j$.

Combining the above, $A[k] \leq x$ holds for the entire range $p \leq k < i$, and $A[k] \geq x$ for $r \geq k > j$; this establishes that the second and third lines of the invariant hold after this pass through the main loop. If $j < i$ at the end of this pass, then we are done because the last line of the invariant no longer applies. Otherwise, the **until** stopping conditions, followed by the exchange in line 10, give $A[i] \leq x$ and $A[j] \geq x$ after the exchange, so the last line of the invariant holds in this case.

Since each pass through the main loop preserves the invariant, the invariant is true upon termination of PARTITION, which returns $q = j$. Since $j \leq i$ at this point, $A[k] \leq x$ for all $k < q$ by the invariant. Also by the invariant, $A[k] \geq x$ for all $k > q$. By the **until** termination condition in line 6 on the last pass through the loop, j stops at a point satisfying $A[j] \leq x$: thus $A[q] \leq x$ as well. Together with the facts that i and j stay in bounds, that PARTITION terminates, and that the partition is nontrivial, this completes the proof that PARTITION is correct as defined above.