

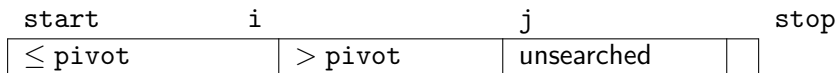
Why study quick sort in light of the facts that

- ▶ you've seen it in earlier courses
- ▶ other sorts (counting sort, radix sort, merge sort, Tim sort) beat it under some circumstances

?

Because

- ▶ It's a beautiful algorithm.
- ▶ It's a good context in which to apply what we've done recently.
- ▶ This chapter has some really good exercises and problems in it.
- ▶ There is a nifty side note I want to show you.



Invariant (`partition()`)

- (a) $\forall k \in [\text{start}, i], A[k] \leq \text{pivot}$
- (b) $\forall k \in (i, j), A[k] > \text{pivot}$
- (c) $A[\text{stop} - 1] = \text{pivot}$
- (d) $j - \text{start}$ is the number of iterations

Invariant (partition())

- (a) $\forall k \in [\text{start}, i], A[k] \leq \text{pivot}$
- (b) $\forall k \in (i, j), A[k] > \text{pivot}$
- (c) $A[\text{stop} - 1] = \text{pivot}$
- (d) $j - \text{start}$ is the number of iterations

Initialization. Before the loop starts, a and b are trivial, and c is true by assignment. Moreover, $j - \text{start} = 0$, so d.

Maintenance. Suppose the invariant holds after some ℓ iterations. On the $\ell + 1$ st iteration, either $A_{\text{old}}[j] \leq \text{pivot}$ or $A_{\text{old}}[j] > \text{pivot}$.

Case 1. Suppose $A_{\text{old}}[j] \leq \text{pivot}$. Then

$$\begin{aligned}i_{\text{new}} &= i_{\text{old}} + 1 \\A_{\text{new}}[i_{\text{new}}] &= A_{\text{old}}[j_{\text{old}}] \leq \text{pivot} \\A_{\text{new}}[j_{\text{new}} - 1] &= A_{\text{old}}[j_{\text{old}}] = A[i_{\text{new}}] \\&= A[i_{\text{old}} + 1] > \text{pivot}\end{aligned}$$

Invariant (partition())

- (a) $\forall k \in [\text{start}, i], A[k] \leq \text{pivot}$
- (b) $\forall k \in (i, j), A[k] > \text{pivot}$
- (c) $A[\text{stop} - 1] = \text{pivot}$
- (d) $j - \text{start}$ is the number of iterations

[Continued...]

On the $\ell + 1$ st iteration, either $A_{old}[j] \leq \text{pivot}$ or $A_{old}[j] > \text{pivot}$.

Case 2. *Suppose $A_{old}[j] > \text{pivot}$. Then*

$$A[j_{new} - 1] = A[j_{old}] > \text{pivot}$$

In either case, $j_{new} - \text{start} = j_{old} + 1 - \text{start} = \ell + 1$. \square

Ex 7.2-3. Not-quite-right solution. Find the error.

Recursion Invariant. For each call to `quicksort_r()` on the range $[start, stop)$, A is backward sorted on the range.

Proof. *By induction on the structure of the recursive calls to `quicksort_r()`.*

Initialization. *This is given, that is, that the initial array is backwards sorted.*

Maintenance. *Suppose the current subarray—the input to the call of `quicksort_r()` is backwards sorted.. The pivot will be the smallest element. This means the less-than-the-pivot section will be empty, and the greater-than-the-pivot section will have no exchanges and hence is still backwards-sorted. `quicksort_r()` will be called on that subarray.*