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.

`start`	`i`	`j`	`stop`
$\leq$ `pivot`	$>$ `pivot`	unsearched	

Invariant (`partition()`)

(a) $\forall\ k \in [\texttt{start}, i], A[k] \leq \texttt{pivot}$

(b) $\forall\ k \in (i, j), A[k] > \texttt{pivot}$

(c) $A[\texttt{stop} - 1] = \texttt{pivot}$

(d) $j - \texttt{start}$ *is the number of iterations*

Invariant (partition())

(a) $\forall\ k \in [\texttt{start}, i], A[k] \leq \texttt{pivot}$

(b) $\forall\ k \in (i, j), A[k] > \texttt{pivot}$

(c) $A[\texttt{stop} - 1] = \texttt{pivot}$

(d) $j - \texttt{start}$ *is the number of iterations*

Initialization. *Before the loop starts, a and b are trivial, and c is true by assignment. Moreover, $j - \texttt{start} = 0$, so d.*
Maintenance. *Suppose the invariant holds after some ℓ iterations. On the $\ell + 1$st iteration, either $A_{old}[j] \leq \texttt{pivot}$ or $A_{old}[j] > \texttt{pivot}$.*
Case 1. *Suppose $A_{old}[j] \leq \texttt{pivot}$. Then*

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

Invariant (`partition()`)

(a) $\forall\ k \in [\texttt{start}, i], A[k] \leq \texttt{pivot}$

(b) $\forall\ k \in (i, j), A[k] > \texttt{pivot}$

(c) $A[\texttt{stop} - 1] = \texttt{pivot}$

(d) $j - \texttt{start}$ *is the number of iterations*

[Continued. . .]
On the $\ell + 1$st iteration, either $A_{old}[j] \leq \texttt{pivot}$ or $A_{old}[j] > \texttt{pivot}$.
Case 2. *Suppose $A_{old}[j] > \texttt{pivot}$. Then*

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

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

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* $\mathtt{quicksort\_r()}$*.*
> **Initialization.** *This is given, that is, that the initial array is backwards sorted.*
> **Maintenance.** *Suppose the current subarray—the input to the call of* $\mathtt{quicksort\_r()}$ *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.* $\mathtt{quicksort\_r()}$ *will be called on that subarray.*