I. Core / B. Divide and Conquer

- General introduction (last week Wednesday)
- Solving recurrences (last week Friday)
- The master method (Monday)
- Quick sort (**today**)
- (Begin advanced analysis techniques Friday)

Today:

- Algorithm itself
- Correctness
- Efficiency
- “Killer adversary”

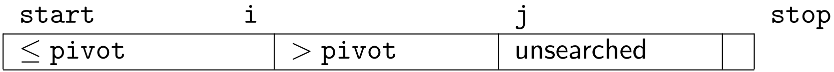
Why study quick sort in light of the facts that

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

(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*

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

Invariant (partition())

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

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

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

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

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

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

In either case, $j_{new} - \mathtt{start} = j_{old} + 1 - \mathtt{start} = \ell + 1$*.* □

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—that is, the input to the call of* `quicksort_r()`*—is backwards sorted. The pivot is the smallest element.*
>
> *Hence when the loop terminates, the less-than-the-pivot section is empty, and the greater-than-the-pivot section has no exchanges and hence is still backwards-sorted.* `quicksort_r()` *is then called on that subarray.*

For next time

Read sections Read Sec 8.(1-4), although really Sec 8.1 is the main thing we'll be talking about, so read that carefully.

Sections 8.(2-4) should be review from CSCI 345, but all that is stuff that you do need to know, so the review is worth it.

Do Ex 8.1-(1,3,4)

"Divide and Conquer" problem set due Wed, Sept 25.