Slide 1

Administrivia

- (None.)

Slide 2

Quicksort

- You've probably heard of quicksort as a sorting algorithm? I think it's useful in this course as an example of several things: It's an example of divide and conquer (and one where the analysis of run time is a little tricky), and one where the "split" part is nontrivial and I think benefits from an approach based on a loop invariant.

Slide 3

Quicksort as Divide and Conquer

- One more algorithm to sort array A of size n, based on recursively sorting subarrays. So we need a program QUICKSORT(A, p, r) that sorts a subarray $A[p..r]$.
- Split here consists of partitioning $A[p..r]$ into two (possibly empty) subarrays $A[p..q-1]$ and $A[q+1..r]$, based on "pivot" $A[q]$ ($p \leq q \leq r$, how to choose q can vary), such that everything in subarray $A[p..q-1]$ is $\leq A[q]$ and everything in subarray $A[q+1..r]$ is $\geq A[q]$. For this we'll need a procedure PARTITION, which rearranges $A[p..r]$ and returns the index q that marks the position separating the subarrays.
- Sort two subarrays with recursive calls to QUICKSORT.
- Merge is basically nothing; arrays are sorted in place.
- Pseudocode on next slide . . .

Slide 4

Quicksort, Continued

- QUICKSORT(A, p, r)
  if $p < r$
  then
  $q =$PARTITION(A, p, r)
  QUICKSORT$(A, p, q-1)$
  QUICKSORT$(A, q+1, r)$
- Initial call is QUICKSORT$(A, 1, n)$.
- As with mergesort, the hard part is with the subprogram — merge for mergesort, divide here.

Slide 5

Partition

- What we want is a procedure PARTITION(A, p, r) that returns q with $p \leq q \leq r$ and rearranges $A[p : r]$ such that everything in $A[p..q-1]$ is $\leq A[q]$ and everything in $A[q+1..r]$ is $\geq A[q]$.

  (Note "rearranges" — ending A must be permutation of initial A.)
- The idea of this is easy enough, and it's not hard to work through examples pictorially, but code is tricky. What can help is an invariant based on splitting the subarray into four regions, some of which might be empty. Left to right:
  - Entries known to be $\leq$ the pivot.
  - Entries known to be $\geq$ the pivot.
  - Entries not yet examined.
  - The pivot.

Slide 6

Partition — Invariant

- PARTITION(A, p, r)

```
x = A[r] // the pivot
i = p − 1
for j = p to r − 1
        if A[j] ≤ x
                i = i + 1
                exchange A[i] and A[j]
exchange A[i + 1] and A[r]
return i + 1
```

- This code chooses $A[r]$ as the pivot and then scans through the subarray left to right (using j). Invariant on next slide.

Slide 7

Partition — Invariant

- Invariant restated in terms of variables used in algorithm:
  - All elements in $A[p..i]$ are $\leq$ pivot (x).
  - All elements in $A[i+1..j-1]$ are $\geq$ pivot (x).
  - Elements in $A[j..r-1]$ are not yet examined.
  - $A[r]$ is the pivot.
- Initialization for loop invariant:
  We will use $A[r]$ for the pivot (x in the code). $i == p-1$ and $j == p$, so in terms of algorithm variables the first two subarrays are $A[p..p-1]$ and $A[p..p-1]$, both empty.

Slide 8

Partition — Invariant, Continued

- Maintenance for loop invariant:
  Suppose true for some value of j. Consider what happens depending on how $A[j]$ compares to x. If $\leq$ then it becomes part of the first subarray (increment i, move $A[i]$, increment j); if not then it becomes part of second subarray (increment j). Now true again.
- Termination for loop invariant:
  Since loop is a **for** loop, it terminates, with $j = r-1$, $A[p..i]$ are all $\leq$ the pivot and $A[i+1..r-1]$ are all $\geq$ the pivot. If we then exchange $A[i+1]$ and $A[r]$ and return $i+1$ this accomplishes exactly what this procedure is supposed to do.

Slide 9

Quicksort — Argument for Correctness

- Does it do the right thing for base cases? Yes — subarrays of size 0 or 1, already sorted.
- Does it eventually reduce everything to a base case? Yes, because it splits every subarray into two subarrays, both no bigger than it is, and at least one of which is smaller (because it removes the pivot from the elements that still have to be sorted). (Maybe we note here that unlike mergesort this may split into pieces of very different sizes.)
- If the two recursive calls work, does the main program work? Yes: The pivot is in the right place, because PARTITION puts everything smaller to its left and everything larger to its right.
- And the recursive calls work — the overall argument is mathematical induction (remember that from Discrete?) on the size of the subarray.

Slide 10

Quicksort — Analysis of Execution Time

- Mergesort splits arrays into two subarrays of equal-or-nearly-so size, making it easy to write a recurrence relation and solve it using the Master Theorem. This doesn't work for quicksort, however — subarrays can be very different in size. (Think a minute about the range — more on next slide.)
- However, the textbook also gives an argument (section 4.4) for analyzing runtime efficiency based on a tree view of recursive calls, and we can do something similar for quicksort.

Slide 11

Quicksort — Analysis of Execution Time, Continued

- As noted previously, PARTITION splits a subarray of size n into two pieces, each of size at most n-1, because neither piece includes the pivot. (The maximum size of n-1 happens if the subarray is already either in order or in reverse order.)
- If we were to draw a recursion tree such as the ones in textbook section 4.4, we could observe that:
- Each level involves one or more calls to PARTITION, on subarrays whose sizes total at most n (size of whole array). PARTITION's running time is proportional to the size of the subarray it's called on, which means total work/time for each level of the tree is $\Theta(n)$.
- How many levels are in the tree?...

Slide 12

Quicksort — Analysis of Execution Time, Continued

- Worst case (input in either forward or reverse sorted order) is that there are $n - 1$ levels.
- Best case (split always splits into subarrays of as-equal-as-possible size) is that there are $\log_2 n$ levels, as for merge sort.
- Overall execution time is time for each level times number of levels, meaning a best case of $\Theta(n \log n)$ and a worst case of $\Theta(n^2)$.

Slide 13

Minute Essay

- Questions?