Slide 1

Administrivia

- Reminder: Homework 6 due Tuesday after holiday.
- Reminder: Quizzes 5 and 6 next week also. (Or drop Quiz 6?)
- Grades for homeworks 2 and 3 mailed.

Slide 2

Sorting and Searching — Recap/Review

- (Review slides I would have used . . .)

Slide 3

Recursion Revisited, and More Sorts

- We used recursion to repeat things earlier in the semester (before learning about collection methods and loops). Useful, and can be elegant, but many thing are probably as simple, or simpler, with other methods.
- However, some algorithms are *much* easier to express with recursion, particularly those where a complete solution involves more than one recursive call. (Classic example is computing elements of the Fibonacci series.)
- Two more classic — and more efficient — sorts also work this way, namely quicksort and mergesort.

Slide 4

Quicksort — Executive-Level Summary

- The idea is to pick one element of the array as a "pivot", rearrange the array so that everything smaller than the pivot is to its left and everything larger is to its right (meaning that the pivot is now in exactly the right spot), and use recursion to sort the elements to the left and the elements to the right.
- Code sketch:

```
// sort nums[start .. end]
def quicksort(nums:Array[Int], start:Int, end:Int) = {
  if (start < end) {
    val pivot = nums[start]
    val pivotIndex = partition(pivot, nums, start, end)
    quicksort(nums, start, pivotIndex-1)
    quicksort(nums, pivotIndex+1, end)
  }
// rearrange nums[start .. end] so that:
// all elements <= pivot are to the left
// all elements > pivot are to the right
// pivot is at nums[index]
// index is returned
def partition(pivot:Int, nums:Array[Int], start:Int, end:Int) : Int = { /* .. */ }
}
```

Slide 5

Mergesort — Executive-Level Summary

- The idea is to split the array into two pieces of equal size (or as close as we can get), sort the pieces using recursion, and merge the two sorted pieces. (So this cannot be done in place.)
- Code sketch (involving more copying than is strictly needed):

```
// sort nums[start .. end] into numsOut[start .. end]
def mergesort(nums:Array[Int], numsOut:Array[Int], start:Int, end:Int) {
  if (start < end) {
    val split = (start+end)/2
    mergesort(nums, numsOut, start, split)
    mergesort(nums, numsOut, split+1, end)
    copy(numsOut, nums, start, end)
    merge(nums, numsOut, start, split, end)
  }
}
// copy nums[start .. end] to numsOut[start .. end]
def copy(nums:Array[Int], numsOut:Array[Int], start, end) { /* .. */ }
// merge sorted subarrays nums[start .. split] and nums[split+1 .. end]
// to produce numsOut[start .. end]
def merge(nums:Array[Int], numsOut:Array[Int], start, end) { /* .. */ }
```

Slide 6

GUIs and Event-Driven Programming

- Up to now our programs have all interacted with their environment in a fairly primitive way — getting text input from standard input and files and writing text output to standard output and files, and interacting with the human user in a basically synchronous way.
- Programs with GUIs, though, are typically somewhat different — the main program (which is often hidden in library code) is often just a loop that waits for keyboard/mouse input delivered by the program's environment (operating system, graphical environment, window manager, etc.).
- This leads to an "event-driven" programming model that can seem rather different from what's used for text-based programs. Rather than defining the whole interaction in the way we've been doing, you typically write code that defines (often in terms of library components) what appears on the screen and how each component responds to user input.

Slide 7

GUIs in Scala

- Java (which is what's under Scala's hood in some sense) defines not one but two libraries that provide functionality for building GUIS — predefined components such as buttons and checkboxes and menus, plus frameworks/mechanisms for laying things out and defining interaction with users. (Why two? Historical reasons.)
- Scala gives you access to all of that if you want it, and also provides a less verbose syntax for using some of the more-commonly-used things.
- In writing programs, often useful to think in terms of "what should the program's interface look like?" (appearance) and "how should the program respond to user actions/inputs?" (behavior).

Slide 8

Examples

- (Quick look at some examples. More next time.)

Slide 9

Minute Essay

- TBA