Sorting

Chapter 19

Sorting

- The process of arranging a list of items in a particular order
- There are many sorting algorithms, which vary in efficiency

SELECTION SORT

Selection Sort

- The general approach of Selection Sort:
  - Select a value and put it in its final place in the list
  - Repeat for all values

Selection Sort (continued)

- Find the smallest value in the list
  - Swap it with the value in the first position
- Find the 2nd smallest value in the list
  - Swap it with the value in the second position
- Find the 3rd smallest value in the list
  - Swap it with the value in the third position
- Continue until all values are in their proper place

Selection Sort (cont.)

- Essentially, we are finding the min and swapping it with the current index, then incrementing the index until we reach the end
- Each time we go through the list, the smallest remaining value is found and exchanged with the element in the "next" position to be filled
- Note: the current min is *swapped*- values are **not** shifted.

Swapping

- Selection Sort relies on *swapping* two values
- *Swapping* requires three assignment statements:

```
temp = first;
first = second;
second = temp;
```

Polymorphism in Sorting

- Any class that implements the `Comparable` interface defines a `compareTo` method to determine the relative order of objects
- The `compareTo` method returns:
  - n < 0 if the invoking object is less than the parameter
  - 0 if the objects are equal
  - n > 0 if the invoking object is greater than the parameter

Polymorphism in Sorting (continued)

- So we can use polymorphism to develop a generic sort for *any* set of `Comparable` object
- The sorting method will accept as a parameter an array of `Comparable` objects
- At runtime, the JVM will figure out what the *actual* type is (e.g., `Employee`, `Student`, etc.) and will call the `compareTo` method in that class

Selection Sort Examples

- Review the trace and the code.
  - Sort numbers then Students.
- More resources:
  - http://en.wikipedia.org/wiki/Selection_sort#mediaviewer/File:Selection-Sort-Animation.gif
  - http://www.youtube.com/watch?v=MZ-ZeQnUL1Q
  - http://www.youtube.com/watch?v=6kg9Dx72pzs

INSERTION SORT

Insertion Sort

- The general approach of Insertion Sort:
  - Pick any item and insert it into its proper place in a sorted sublist
  - Repeat until all items have been inserted

Insertion Sort (continued)

- Consider the first item to be a sorted sublist (of one item)
- Insert the second item into this sorted sublist, *shifting* the first item as needed to make room to insert the new addition
- Insert the third item into the sorted sublist (of two items), shifting items as necessary
- Repeat until all values are inserted into their proper position

Insertion Sort Examples

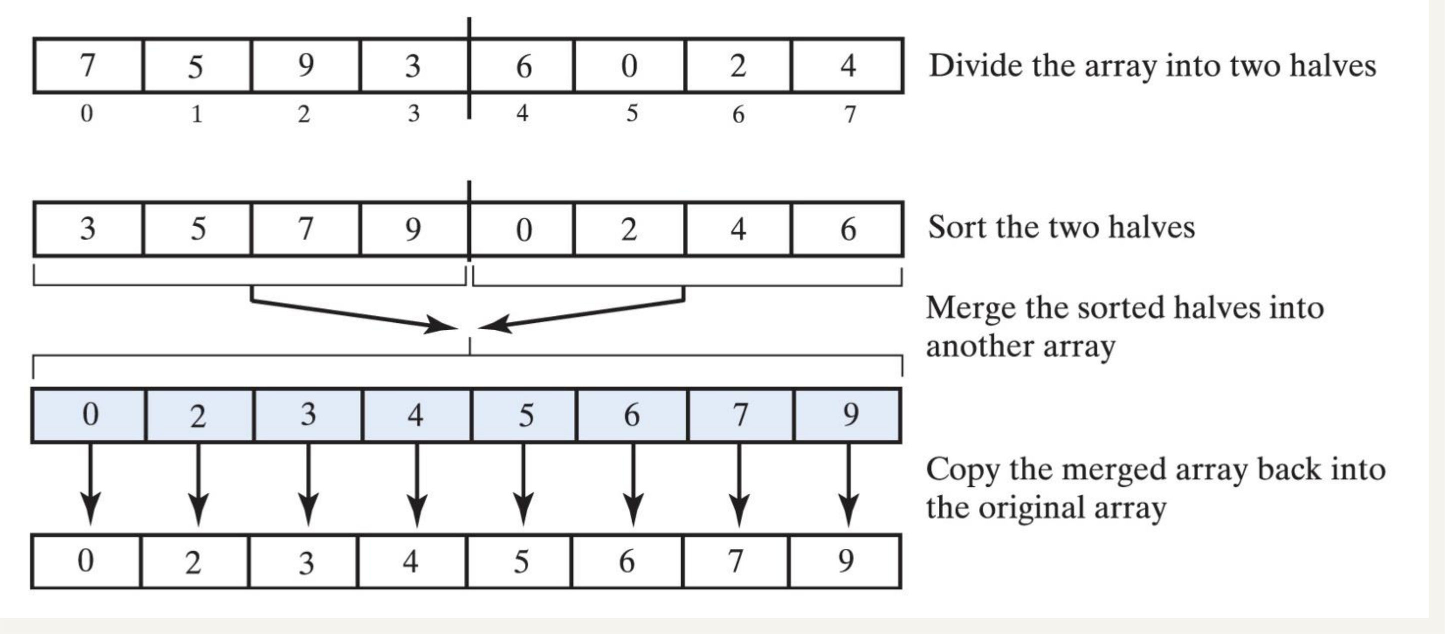
- Review the trace and the code.
  - Sort numbers then Students.
- More resources:
  - http://en.wikipedia.org/wiki/Insertion_sort#mediaviewer/File:Insertion-sort-example-300px.gif
  - https://www.khanacademy.org/computing/computer-science/algorithms/insertion-sort/a/insertion-sort
  - https://www.youtube.com/watch?v=c4BRHC7kTaQ

MERGE SORT

Merge Sort

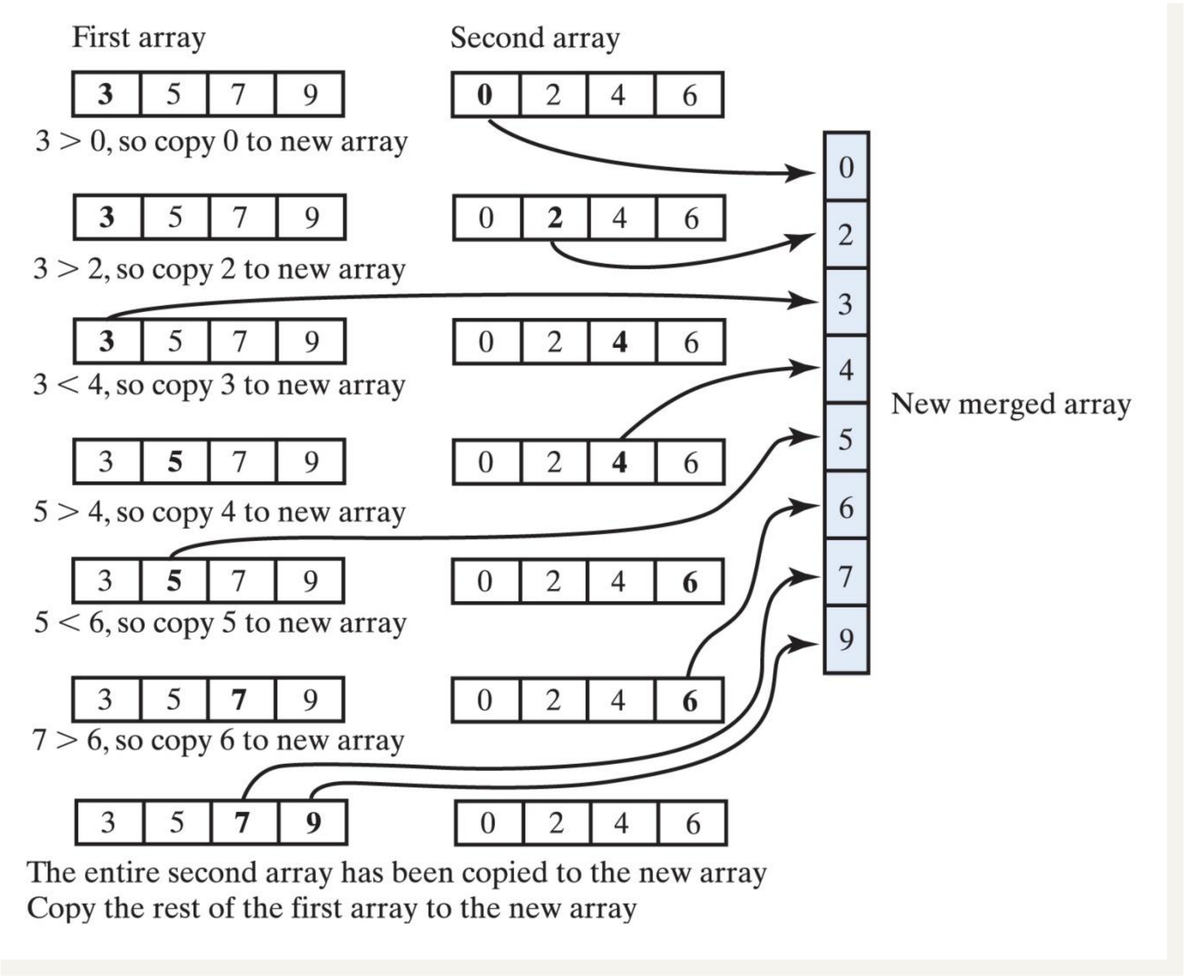
- Divide an array into halves
  - Sort the two halves
  - Merge them into one sorted array
- Referred to as a divide and conquer algorithm
- Often programmed with recursion

Merge Sort (continued)

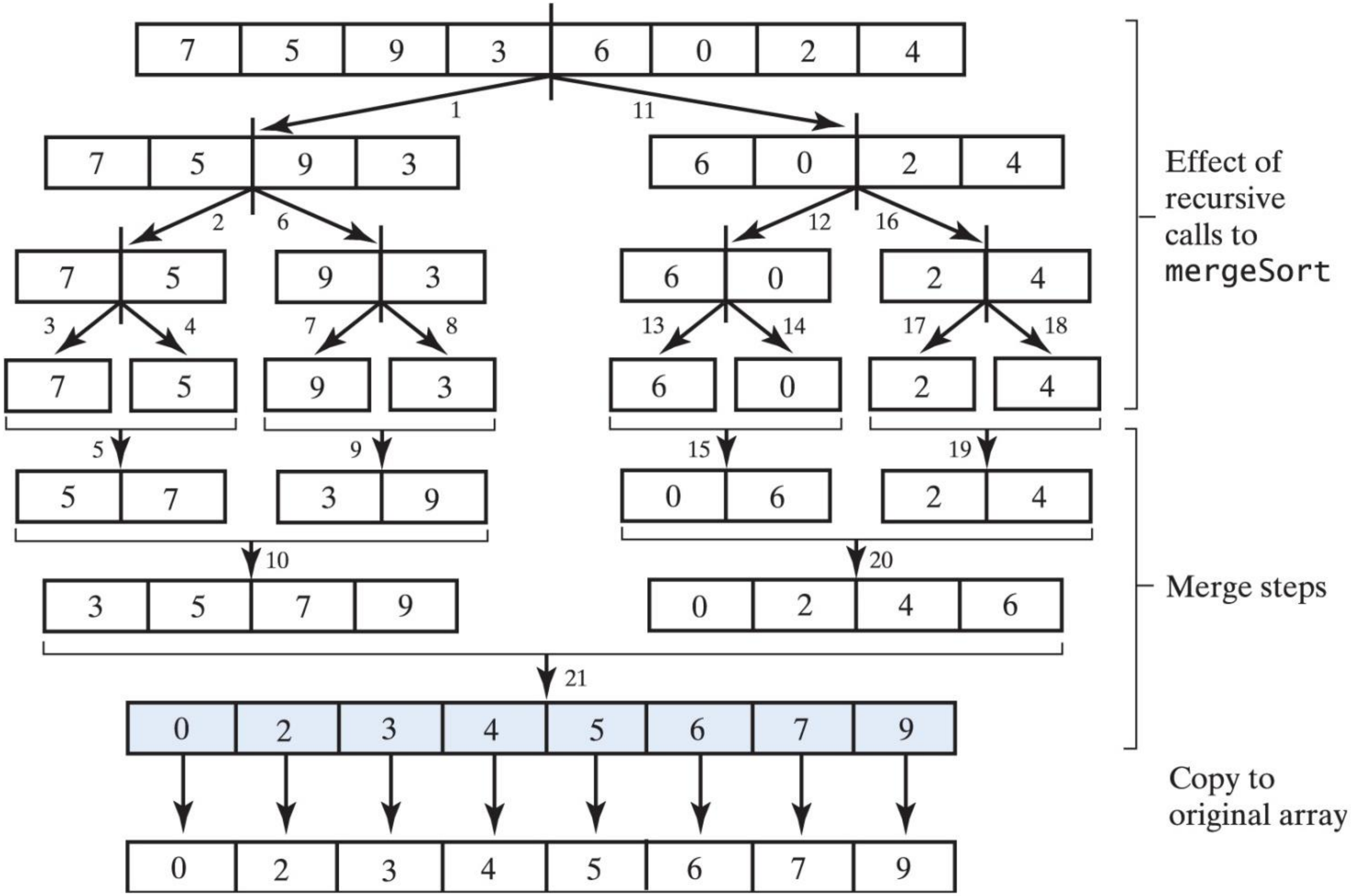

7
5
9
3
6
0
2
4
0
1
2
3
4
5
6
7
Divide the array into two halves
3
5
7
9
0
2
4
6
Sort the two halves
Merge the sorted halves into another array
0
2
3
4
5
6
7
9
Copy the merged array back into the original array
0
2
3
4
5
6
7
9

Merge Sort (continued)

Merging two sorted arrays into one sorted array.


First array
Second array
3 5 7 9
0 2 4 6
3 > 0, so copy 0 to new array
3 > 2, so copy 2 to new array
3 < 4, so copy 3 to new array
5 > 4, so copy 4 to new array
5 < 6, so copy 5 to new array
7 > 6, so copy 6 to new array
The entire second array has been copied to the new array
Copy the rest of the first array to the new array
0 2 3 4 5 6 7 9
New merged array

Merge Sort (continued)



Merge Sort (continued)

- http://www.youtube.com/watch?v=GCae1WNvnZM

QUICK SORT

Quick Sort

- Divides the array into two pieces
  - Not necessarily halves of the array
  - An element of the array is selected as the pivot
- Elements are rearranged so that:
  - The pivot is in its final position in sorted array
  - Elements in positions before pivot are less than the pivot
  - Elements after the pivot are greater than the pivot

Quick Sort (continued)

- Quick sort rearranges the elements in an array during partitioning process
- After each step in the process
  - One element (the pivot) is placed in its correct sorted position
- The elements in each of the two sub arrays
  - Remain in their respective subarrays

Quick Sort (continued)

- Choose a pivot point (or partition value).
- Scan from the right looking for a value that we need to move (a value smaller than the pivot). Stop when we find one.
- Scan from the left looking for a value that we need to move (a value larger than the pivot).
- Swap these values.
- Keep looking and repeating.
- Once the scans cross, swap the pivot with the value from the right-side scan.
- The pivot is now in the correct position.
- Repeat recursively on the left and right of the pivot.

Quick Sort (continued)

- http://www.youtube.com/watch?v=8hHWpuAPBHo
- https://www.youtube.com/watch?v=mN5ib1XasSA

COMPARING SORTS

Comparing the Algorithms

	Average Case	Best Case	Worst Case
Insertion Sort	$O(n^2)$	$O(n)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$

Sorting Algorithms Pros and Cons

- Selection Sort
  - Pro: simple and easy to implement
  - Con: Inefficient for large lists (worse than insertion)
- Insertion Sort
  - Pro: simple and easy to implement
  - Con: Inefficient for large lists
- Merge Sort
  - Pro: Fast
  - Con: Memory requirements, recursive
- Quick Sort
  - Pros: Fast
  - Cons: Complex, recursive, inefficient in worst case (a pre-sorted list)