Announcements (2/20/09)

- Homework 5 due now
- Homework 6 out today, due next Friday
- Reading for this lecture: Chapter 7.

Stability

A sorting algorithm is **stable** if:

- Items in the input with the same value end up in the same order as when they began.

<table>
<thead>
<tr>
<th>Input</th>
<th>Unstable sort</th>
<th>Stable Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adams</td>
<td>Adams 1</td>
<td>Adams 1</td>
</tr>
<tr>
<td>Black</td>
<td>Smith 1</td>
<td>Smith 1</td>
</tr>
<tr>
<td>Brown</td>
<td>Washington 2</td>
<td>Black 2</td>
</tr>
<tr>
<td>Jackson</td>
<td>Jackson 2</td>
<td>Jackson 2</td>
</tr>
<tr>
<td>Jones</td>
<td>Black 2</td>
<td>Washington 2</td>
</tr>
<tr>
<td>Smith</td>
<td>White 3</td>
<td>White 3</td>
</tr>
<tr>
<td>Thompson</td>
<td>Wilson 3</td>
<td>Wilson 3</td>
</tr>
<tr>
<td>Washington</td>
<td>Thompson 4</td>
<td>Brown 4</td>
</tr>
<tr>
<td>White</td>
<td>Brown 4</td>
<td>Jones 4</td>
</tr>
<tr>
<td>Wilson</td>
<td>Jones 4</td>
<td>Thompson 4</td>
</tr>
</tbody>
</table>

“Divide and Conquer”

- Very important strategy in computer science:
  - Divide problem into smaller parts
  - Independently solve the parts
  - Combine these solutions to get overall solution

- **Idea 1**: Divide array into two halves, *recursively* sort left and right halves, then *merge* two halves → known as **Mergesort**

- **Idea 2**: Partition array into small items and large items, then recursively sort the two sets → known as **Quicksort**
Mergesort

- Divide it in two at the midpoint
- Conquer each side in turn (by recursively sorting)
- Merge two halves together

Mergesort Example

```
8  2   9   4   5   3   1   6
```

Merging: Two Pointer Method

- The merging requires an auxiliary array.

```
2  4  8  9  1  3  5  6
```

Auxiliary array
Merging: Two Pointer Method

• The merging requires an auxiliary array.

Auxiliary array

Merging: Finishing Up

Starting from here…

Left finishes up

or

Right finishes up

Merging: Two Pointer Method

• Final result

Auxiliary array

Merging

Merge(A[], Temp[], left, mid, right) {
    Int i, j, k, l, target
    i = left
    j = mid + 1
    target = left
    while (i < mid && j < right) {
        if (A[i] < A[j])
            Temp[target] = A[i++]
        else
            Temp[target] = A[j++]
        target++
    }
    if (i > mid) //left completed//
        for (k = left to target-1)
            A[k] = Temp[k];
    if (j > right) //right completed//
        k = mid
        l = right
        while (k > i)
            A[l--] = A[k--]
        for (k = left to target-1)
            A[k] = Temp[k]
}
Recursive Mergesort

MainMergesort(A[1..n], n) {
    Array Temp[1..n]
    Mergesort(A, Temp, 1, n)
}

Mergesort(A[], Temp[], left, right) {
    if (left < right) {
        mid = (left + right)/2
        Mergesort(A, Temp, left, mid)
        Mergesort(A, Temp, mid+1, right)
        Merge(A, Temp, left, mid, right)
    }
}

What is the recurrence relation?

Iterative Mergesort

Iterative Mergesort reduces copying.
Complexity?
Properties of Mergesort

• In-place?
• Stable?
• Sorted list complexity?
• Nicely extends to handle linked lists.
• Multi-way merge is basis of big data sorting.
• Java uses Mergesort on Collections and on Arrays of Objects.