Announcements (5/12/08)

- Homework due on Friday at the beginning of class.
- 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</td>
<td>Adams</td>
</tr>
<tr>
<td>Black</td>
<td>Smith</td>
<td>Smith</td>
</tr>
<tr>
<td>Brown</td>
<td>Washington</td>
<td>Black</td>
</tr>
<tr>
<td>Jackson</td>
<td>Jackson</td>
<td>Jackson</td>
</tr>
<tr>
<td>Jones</td>
<td>Black</td>
<td>Washington</td>
</tr>
<tr>
<td>Smith</td>
<td>White</td>
<td>White</td>
</tr>
<tr>
<td>Thompson</td>
<td>Wilson</td>
<td>Wilson</td>
</tr>
<tr>
<td>Washington</td>
<td>White</td>
<td>Washington</td>
</tr>
<tr>
<td>White</td>
<td>Brown</td>
<td>Jones</td>
</tr>
<tr>
<td>Wilson</td>
<td>Jones</td>
<td>Thompson</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

Merging: Two Pointer Method

- The merging requires an auxiliary array.

Merging: Two Pointer Method

- The merging requires an auxiliary array.
Merging: Two Pointer Method

- The merging requires an auxiliary array.

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

Auxiliary array

```
1 2 3 4 5 6
```

Merging: Finishing Up

- Starting from here...

```
i j
```

Target

- Left finishes up

```
copy
```

```
i j
```

Target

or

```

```

```

First copy this...

```

```

```

Merging: Two Pointer Method

- Final result

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

Auxiliary array

```
1 2 3 4 5 6
```

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[--] = A[--]
        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?

\[ T(1) = a \quad T(n) = 2T(n/2) + bn + c \]

Mergesort: Complexity

\[
T(i) = a \\
T(n) = 2T(n/2) + bn + c \\
= 2[2T(n/4) + bn + c] + bn + c \\
= 2[2[2T(n/8) + bn + c] + bn + c] + bn + c \\
= 2^3T(n/8) + bn + 2^2c + 2^1c + 2^0c \\
= 2^kT(n/2^k) + bkn + \frac{k-1}{2}2^c
\]

Assume \( n = 2^k \), \( k = \log_2 n \)

\[ T(n) = nT(1) + bn \log_2 n + (2^k - 1)c \\
= an + bn \log_2 n + cn - c \in O(n \log n) \]

Iterative Mergesort

Iterative Mergesort reduces copying.
Complexity? \( O(n \log n) \)
Properties of Mergesort

- In-place? \( \text{N (not easy to do...)} \)
- Stable? \( \text{Y} \)
- Sorted list complexity? \( O(n \log n) \) \( \text{w/ neat trick} \) \( O(n) \)
- 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.