CSE 326: Data Structures
Mergesort

Steve Seitz
Winter 2009

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</td>
<td>Adams</td>
</tr>
<tr>
<td>Smith</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>Thompson</td>
<td>Brown</td>
</tr>
<tr>
<td>White</td>
<td>Jones</td>
<td>Jones</td>
</tr>
<tr>
<td>Wilson</td>
<td>Thompson</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**

[Sedgewick]
Mergesort

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

Mergesort Example

Mergesort Example

Merging: Two Pointer Method

- The merging requires an auxiliary array.

Merging: Two Pointer Method

- The merging requires an auxiliary array.

Auxiliary array

0(n)
Merging: Two Pointer Method

- The merging requires an auxiliary array.

Merging: Finishing Up

Starting from here...

Left finishes up

or

Right finishes up

Merging: Two Pointer Method

- Final result

Merging

```c
void 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++]
    }
    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?

\[ T(1) = 1 \]
\[ T(n) = 2T(n/2) + n \]

**Mergesort: Complexity**

\[ T(1) = 1 \]
\[ T(n) = 2T(n/2) + n \]
\[ = 2(2T(n/4) + n/2) + n \]
\[ = 2(2(2T(n/8) + n/4) + n/2) + n \]
\[ = 8T(n/8) + n + n + n \]
\[ = 8T(n/8) + 3n \]
\[ = 2^3T(n/2^3) + 3n \]
\[ = 2^kT(n/2^k) + kn \]
\[ = nT(1) + (\log_2 n) n \]
\[ = n + n(\log_2 n) = O(n \log n) \]

Iterative Mergesort

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

- In-place? no
- Stable? yes
- Sorted list complexity? \( n \log n \)
- 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.