CS171 Introduction to Computer Science II

MergeSort

Li Xiong
Announcement/Reminders

• Quiz 2 today
• Hw4 to be assigned today
Roadmap

• Elementary sorting (quadratic cost)
  – Bubble sort
  – Selection sort
  – Insertion sort

• Advanced sorting
  – MergeSort
  – QuickSort

• Analysis of recursive algorithms
MergeSort

• Basic idea
  – Divide an array into two halves
  – Sort each half
  – Merge the two sorted halves into a sorted array

Mergesort overview
Mergesort

• A divide and conquer approach using recursion
  – Partition the original problem into two sub-problems
  – Solve each sub-problem using recursion (sort)
  – Combine the results to solve the original problem (merge)
Merge Two Sorted Arrays

• A key step in mergesort
  • Assume subarrays a[lo..mid] (left half) and a[mid+1 ... Hi] (right half) are sorted
  • copy a[] to an auxiliary array aux[]
  • merge the two halves of aux[] to a[] such that it contains all elements and remains sorted

• Example

<table>
<thead>
<tr>
<th>input</th>
<th>a[]</th>
<th>aux[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>G</td>
</tr>
<tr>
<td>G</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>R</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>R</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

```plaintext
<table>
<thead>
<tr>
<th>k</th>
<th>0 1 2 3 4 5 6 7 8 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>G</td>
</tr>
<tr>
<td>M</td>
<td>R</td>
</tr>
<tr>
<td>R</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>R</td>
</tr>
<tr>
<td>R</td>
<td>T</td>
</tr>
</tbody>
</table>
```

```plaintext
<table>
<thead>
<tr>
<th>aux[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>
```
Merging Two Sorted Arrays

1. Start from the **first** elements of each half;
2. Compare and copy the **smaller** element to a[];
3. Increment indices, and continue;
4. If reaching the end of either half; copy the remaining elements of the other half
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {
    assert isSorted(a, lo, mid); // precondition: a[lo..mid] sorted
    assert isSorted(a, mid+1, hi); // precondition: a[mid+1..hi] sorted

    for (int k = lo; k <= hi; k++)
        aux[k] = a[k];

    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++)
    {
        if (i > mid) a[k] = aux[j++];
        else if (j > hi) a[k] = aux[i++];
        else if (less(aux[j], aux[i])) a[k] = aux[j++];
        else a[k] = aux[i++];
    }

    assert isSorted(a, lo, hi); // postcondition: a[lo..hi] sorted
}
Recursive Mergesort

```java
public class Merge {
    private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {
        /* as before */
    }

    private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
        if (hi <= lo) return;
        int mid = lo + (hi - lo) / 2;
        sort(a, aux, lo, mid);
        sort(a, aux, mid+1, hi);
        merge(a, aux, lo, mid, hi);
    }

    public static void sort(Comparable[] a) {
        aux = new Comparable[a.length];
        sort(a, aux, 0, a.length - 1);
    }
}
```
Assertions

**Assertion.** Statement to test assumptions about your program.
- Helps detect logic bugs.
- Documents code.

**Java assert statement.** Throws an exception unless boolean condition is true.

```
assert isSorted(a, lo, hi);
```

**Can enable or disable at runtime.**  ⇒  No cost in production code.

```
java -ea MyProgram     // enable assertions
java -da MyProgram     // disable assertions (default)
```

**Best practices.** Use to check internal invariants. Assume assertions will be disabled in production code (e.g., don't use for external argument-checking).
First base case encountered
Return, and continue.
Merge
Mergesort: visualization
Mergesort demo

• Animation

• German folk dance
  – http://www.youtube.com/watch?v=XaqR3G_NVoo
Merge
Bottom-up MergeSort

1. Every element itself is trivially sorted;
2. Start by merging every two adjacent elements;
3. Then merge every four;
4. Then merge every eight;
5. ... 
6. Done.
Bottom-up mergesort

Basic plan.
- Pass through array, merging subarrays of size 1.
- Repeat for subarrays of size 2, 4, 8, 16, ....

<table>
<thead>
<tr>
<th>sz = 1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>a[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M E R G E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 0, 0, 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E M R G E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 2, 2, 3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E M R G E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 4, 4, 5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E M R G E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 6, 6, 7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E M R G E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 8, 8, 9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E M R G E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 10, 10, 11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E M R G E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 12, 12, 13)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E M R G E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 14, 14, 15)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E M R G E S O R T E X A M P L E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sz = 2</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>a[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M E R G E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 0, 0, 1, 3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E G M R E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 4, 5, 7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E G M R E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 8, 9, 11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E G M R E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 12, 13, 15)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E G M R E S O R T E X A M P L E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sz = 4</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>a[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M E R G E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 0, 3, 7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E G M R E S O R T E X A M P L E</td>
</tr>
<tr>
<td>merge(a, 8, 11, 15)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E G M R E S O R T E X A M P L E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sz = 8</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>a[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>merge(a, 0, 7, 15)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A E E E E E G L M M O P R R S T X</td>
</tr>
</tbody>
</table>
public class MergeBU
{
    private static Comparable[] aux;

    private static void merge(Comparable[] a, int lo, int mid, int hi)
    { /* as before */ }

    public static void sort(Comparable[] a)
    {
        int N = a.length;
        aux = new Comparable[N];
        for (int sz = 1; sz < N; sz = sz+sz)
        {
            for (int lo = 0; lo < N-sz; lo += sz+sz)
                merge(a, lo, lo+sz-1, Math.min(lo+sz+sz-1, N-1));
        }
    }
}