Last Time

• Shell Sort
  – Insertion sort, but allowing for bigger “jumps” in the exchanges
  – Need to pick a good increment sequence, of which there is no proven best sequence
• Basic idea: Split an array into two halves, sort them, and then merge them back into a single sorted array.
  – (How do we sort each half?)
Naïve Merge
Abstract In-Place Merge

```java
public void merge(Comparable[] a, int lo, int mid, int hi) {
    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++) {
        aux[k] = a[k];
    } //for
    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++];
        } //if-else
    } //for
} //merge
```
public void merge(Comparable[] a, int lo, int mid, int hi) {
    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++) {
        aux[k] = a[k];
    } //for
    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++];
        } //if-else
    } //for
} //merge
Abstract In-Place Merge Visual

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

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aux[]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| merged result | A | C | E | E | G | M | R | R | T |

02/08/2016
Merge Sort
Allegheny College
Abstract In-Place Merge

• Still uses extra space, but not a substantial amount more.

• We only need to allocate memory for aux once; then we can continually overwrite it as we work through the data.

• The merge function actually handles the sort! Now we just need to structure our merge() calls:
  – Top-down mergesort
  – Bottom-up mergesort
Top-Down Mergesort

private static Comparable[] aux;

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

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

• This is called a divide-and-conquer algorithm – it recursively breaks down the problem into 2+ subproblems of the same (or related) type, until they become simple enough to solve directly.

• Think of it as a proof by induction in code form – we can sort a simple array, and then we can sort a complex array from a previous pair of arrays.
Top-Down Mergesort Evaluation

• How many compares does this algorithm perform?
  – Let $C(N)$ be the number of compares needed to sort an array of length $N$.
    • $C(0) = C(1) = 0$
  – Upper bound for the number of compares required:
    • $C(N) \leq C(N/2) + C(N/2) + N$
  – Lower bound for the number of compares required:
    • $C(N) \leq C(N/2) + C(N/2) + N/2$
  – Can we get an exact solution?
Top-Down Mergesort Evaluation

• To make things easy, let’s say that \( N \) is a power of 2 (say \( N = 2^n \)). This tells us that \( \frac{N}{2} = 2^{n-1} \).

\[
\begin{align*}
C(2^n) &= C(2^{n-1}) + C(2^{n-1}) + 2^n \\
C(2^n) &= 2C(2^{n-1}) + 2^n \\
C(2^n)/2^n &= 2C(2^{n-1})/2^n + 2^n/2^n \\
C(2^n)/2^n &= C(2^{n-1})/2^{n-1} + 1 \\
C(2^n)/2^n &= C(2^{n-2})/2^{n-2} + 1 + 1 \\
\ldots \ldots \\
C(2^n)/2^n &= C(0)/2^0 + n \\
C(2^n)/2^n &= 0/1 + n \\
C(2^n)/2^n &= n \\
C(2^n) &= n \times 2^n \\
C(2^n) &= C(N) = n \times 2^n = N \times \log(N)
\end{align*}
\]

- Divide both sides by \( 2^n \)
- Apply same equation to \( 2^{n-1} \)
- Repeat n-1 times
- Multiply by \( 2^n \)
- Convert back from \( n \) to \( N \)
Top-Down Mergesort Evaluation

• How many compares does this algorithm perform?
  – $N \times \log(N)$

• How many exchanges does this algorithm perform?
  – Well… we don’t actually do exchanges. Instead, we do array accesses.
  – Each merge uses $2N$ for the copy, $2N$ for the move back, and at most $2N$ for compares.
  – $6N \times \log(N)$
Top-Down Mergesort Improvements

• Now we can sort in \( N \times \log(N) \) time – a substantial improvement over the \( N^2 \) time of Insertion and Selection Sorts.

• We know that Insertion Sort is efficient for small arrays, so if we switch to Insertion Sort once the problem is broken down below some threshold, we can improve Mergesort by 10-15%.

• We can reduce the run time to be linear for arrays that are already sorted, by adding a test to skip the call to `merge()` if \( a[mid] \leq a[mid+1] \).
Top-Down Mergesort Improvements

![Diagram of Mergesort process]

- first subarray
- second subarray
- first merge
- first half sorted
- second half sorted
- result
private static Comparable[] aux;

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);
        }
    }
}

Bottom-Up Mergesort

• Rather than follow a tree-like structure, instead let’s merge all of the 1-length arrays into 2s in one pass, then all of the 2-length arrays into 4s, and repeat.
  – The second subarray may be smaller than the first in the last pass, but that’s no problem for merge() as we designed it.
### Bottom-Up Mergesort Visual

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

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

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

| sz = 8 | merge(a, 0, 7, 15) | A E E E E E G L M M O P R R S T X |
Bottom-Up Mergesort Evaluation

• Did our number of compares and array accesses change with this new implementation?
  – Well, we still have $N \times \log(N)$ compares ($\log(N)$ passes through as we increase $sz$, and between $N/2$ and $N$ items compared on each pass) and $6N$ array accesses per compare.
  – The exact number may have changed, but the order of growth has not.

• Bottom-up is better for linked lists. (why?)
Any Questions?