Outline

- Quadratic-Time Sorting
  - Bubble/Shaker Sort
  - Insertion Sort
  - Odd-Even Sort

- Linearithmic-Time Sorting
  - Heap Sort
  - Merge Sort
  - Quick Sort

- Conclusion

Check out this link for animation of various sorting algorithms:
http://cs.pugetsound.edu/~aasmith/sorters/
Heap Sort

- Already saw this: (Refer back to Lec 8 notes)
  - Insert all elements in the list into a min-heap
  - Repeat until heap is empty
    - Remove smallest element from the heap
    - Put it in tail of new list

```java
public static int[] heapSort(int[] list) {
    Heap<Integer> heap = new Heap<>();

    // build min-heap -- add all items from the list
    for (int i = 0; i < list.length; i++) {
        heap.add(list[i]);
    }

    // re-build the list by continuously pulling the min from the heap
    for (int i = 0; i < list.length; i++) {
        list[i] = heap.remove(0);
    }
    return list;
}
```
Heap Sort Analysis

- **Pros:**
  - It's fast: $O(n \log n)$ time in best/worst/average case

- **Cons:**
  - *Our* version uses up $n$ space (need to create a heap of size $n$)
  - But there is a better version that does not create auxiliary space
Outline

- Quadratic-Time Sorting
  - Bubble/Shaker Sort
  - Insertion Sort
  - Odd-Even Sort

- Linearithmic-Time Sorting
  - Heap Sort
  - Merge Sort
  - Quick Sort

- Conclusion

Check out this link for animation of various sorting algorithms:
http://cs.pugetsound.edu/~aasmith/sorters/
Merge Sort (Demo)

- Let's think recursively about sorting a list

- Let's decompose the problem into smaller subproblems
  - Split the list into two sublists
    - Recursively, so keep doing it...
    - When do we get to stop?
  - Merge the sorted sublists together
Insight: Merging Two Sorted Arrays Is Fast

- To merge two sorted arrays into a new sorted array:
  - Repeatedly compare the two smallest items in A[.] and B[.]
  - Copy the smaller of the two to the new array
Merging Two Sorted Arrays

- To merge two sorted arrays into a new sorted array:
  - Repeatedly compare the two smallest items (tiebreak: take from first)
  - Copy the smaller of the two to the new array

- Let's write `merge()`
  - Complexity: $O(m + n)$, where $m$, $n$ refer to sizes of the 2 arrays
Merge Sort

- **Observations:**
  - Just saw: Merging two *sorted lists* into a sorted list is linear time
  - Insight: A list containing 0 or 1 element is already sorted

- *Merge Sort* exploits the above two observations
  - Input an unsorted list
  - Recursively split the list in two unsorted halves
    - Until each of the halves contains one element
    - Merge the sorted halves together
Merge Sort (Cont.)

Unsorted

Sorted
Complexity of Merge

- First, determine the time complexity of \texttt{merge()}
  - Just need to examine each item once
  - As we know from before, \( T_{\text{merge}}(m, n) = m + n \)

- But we're always merging two roughly-equal halves:
  - So, in our case, \( T_{\text{merge}}(n/2, n/2) = n \)
Complexity of MergeSort

- Visually...

- MergeSort's running time:

\[ T(n) = \begin{cases} 
1, & \text{if } n \leq 1 \\
2T(n/2) + 2n + 1, & \text{otherwise}
\end{cases} \]

\[ T(n) = O(n \log_2 n) \]
Analysis of MergeSort

- **Pros:**
  - Log-linear time complexity
  - Time complexity is consistent (best/avg/worst) and predictable

- **Cons:**
  - Space complexity can be a problem
    - Requires 2 additional copies of the list each recursive call
      - One created when splitting, another created when merging
  - Semi-sorted lists don't help, unlike the linear-time sorts