Insertion and Selection Sort

Section 2

John Kulczak and Alex Andrews

Sorting

- Sorts from smallest to largest.
- Must be comparable objects (compareTo() method must be implemented).
- Selection and Insertion sort are inefficient for larger lists, but useful for smaller lists.
- Why sort a list? Sorted lists can use certain search algorithms.

Selection Sort

- Searches the unsorted part of the list for the smallest element.
- Swaps the smallest element from the unsorted list with the first element in the unsorted list.
- Repeats this process for next element, all the way until the last element.
- $O(n^2)$ efficiency.

Selection Sort Example

Green = Sorted part of the list

8 5 1 2 7	unsorted list
1 5 8 2 7	after 1st pass
1 2 8 5 7	after 2nd pass
1 2 5 8 7	after 3rd pass
1 2 5 7 8	after 4th pass

Insertion Sort

- Divides the list into two parts, a sorted list followed by an unsorted list. (At the start, the first element in the list is considered sorted with respect to itself).
- Moves elements from the unsorted list into the sorted list, one at a time.
- Inserts elements by shifting elements of higher value to the right.

- Efficiency ranges from $O(n^2)$ to $O(n)$, depending on how sorted the list is beforehand.
- Worst case for insertion sort efficiency would be an already sorted list, but in reverse order.

Insertion Sort Example

Green = Sorted part of the list
Underlined = next element to be inserted

8 5 1 2 7	unsorted list
5 8 1 2 7	after 1st pass
1 5 8 2 7	after 2nd pass
1 2 5 8 7	after 3rd pass
1 2 5 7 8	after 4th pass