Why sort
- It’s nice to see data in sorted order
- Easier to search (binary search)
- Sorting used as a step in many algorithms

Sorting algorithms
- There are many algorithms for sorting:
  - Selection sort
  - Insertion sort
  - Bubble sort
  - Merge sort
  - Heap sort
  - Radix sort
  - Quick sort
  - Stooge sort
- Each has its advantages and disadvantages

Selection Sort
- Find the smallest item
- Put it in the first position
  - Find the 2\textsuperscript{nd} smallest item
  - Put it in the 2\textsuperscript{nd} position
  - Find the 3\textsuperscript{rd} smallest item
  - Put it in the 3\textsuperscript{rd} position
  
Selection Sort code

```java
public static void selectionSort(Comparable[] array){
    int min;
    for (int i = 0; i < array.length-1; i++) {
        min = i;
        for (int j = i+1; j < array.length; j++)
            if (array[j].compareTo(array[min]) < 0)
                min = j;
        swap (array, min, i);
    }
}

private static void swap(Comparable[] array, int i, int j){
    Comparable temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}
```
Insertion sort

- Works the same way you arrange your hand when playing cards.
  - Pick up a card and place it in your hand in the correct position relative to the cards you're already holding.

Arranging a hand of cards

Insertion Sort (cont.)
Insertion sort – more formally

- Insertion sort partitions the array into two regions: sorted, and unsorted.
- Each iteration the sorted part grows by 1.

Insertion Sort Algorithm

```java
public static void insertionSort(Comparable[] array) {
    for (int i = 1; i < array.length; i++) {
        Comparable temp = array[i];
        int position = i;
        // shift larger values to the right
        while (position > 0 && array[position-1].compareTo(temp) > 0) {
            array[position] = array[position-1];
            position--;
        }
        // insert the current item
        array[position] = temp;
    }
}
```

With a for loop

```java
public static void insertionSort(Comparable[] array) {
    for (int i = 1; i < array.length; i++) {
        Comparable temp = array[i];
        int position = i;
        // shift larger values to the right
        while (position > 0 && array[position-1].compareTo(temp) > 0) {
            array[position] = array[position-1];
            position--;
        }
        // insert the current item
        array[position] = temp;
    }
}
```
Sorting Linked Lists
- Accessing an element in a linked list takes time.
- Can you sort a linked list with Selection Sort or Insertion Sort maintaining the same level of efficiency as using arrays?

Bubble Sort
- Compares neighboring elements, and swaps them if they are not in order
  - Effect: the largest value will "bubble" to the last position in the array.
  - Repeating the process will bubble the 2nd largest value to the 2nd last position in the array.

Merge Sort
- Basic idea
  - Divide data into two (smaller) parts
  - Sort the parts
  - Merge the sorted parts

```
public static void bubbleSort (Comparable [] array) {
    for (int position = array.length-1; position>=0; position--) {
        for (int i = 0; i < position; i++) {
            if (array[i].compareTo(array[i+1]) > 0) {
                swap(array, i, i+1);
            }
        }
    }
}
```

```
public static void bubbleSort (Comparable [] array) {
    for (int position = array.length-1; position>=0; position--) {
        for (int i = 0; i < position; i++) {
            if (array[i].compareTo(array[i+1]) > 0) {
                swap(array, i, i+1);
            }
        }
    }
}
```
Merge Sort - Merge

```
{1,2,3,4,5,6,7,9}
    
{2,3,7,9}  
    (3,7)  
  (2,9)   (1,9) 
      (1)   (8) 
            (4,5)  
                     (5) 
```

MergeSort

```
public static void mergesort(Comparable[] array, int first, int last){
// Sorts array[first...last] in ascending order
if (first < last) {
    int mid = (first + last) / 2; // midpoint of the array
    mergesort(array, first, mid);
    mergesort(array, mid + 1, last);
    merge(array, first, mid, last);
} // if first >= last, there is nothing to do
}
```

```
why does it work?
```

Merge

```
Data: 2 3 7 9 1 4 5 6
Temp:   
Step 1:
  2 3 7 9 1 4 5 6
    1
Step 2:
  2 3 7 9 1 4 5 6
    1 2
Step 3:
  2 3 7 9 1 4 5 6
    1 2 3
Step 4:
  2 3 7 9 1 4 5 6
    1 2 3 4
```

Stooge Sort

```
public static void stoogeSort(Comparable[] array, int i, int j)
{
    if (array[i].compareTo(array[j]) > 0) {
        swap(array, i, j);
    }
    if (j - i > 1) {
        int third = (j - i + 1) / 3;
        stoogeSort(array, i, i + third);
        stoogeSort(array, i + third, j);
        stoogeSort(array, i, j - third);
    }
}
```

```
public static void stoogeSort(Comparable[] array) {
    stoogeSort(array, 0, array.length - 1);
}
```