<u>CSCI 112</u>
<u>Friday, July 16, 2010</u>

<u>Sorting Objects</u>

Let's say we create some Dogs:

```
class Dog{
	private String name;
	private int age;

	public Dog( String name, int age ){
		this.name = name;
		this.age = age;
	}
}
```

We create two dogs (incomplete code):

```
Dog a = new Dog( "Fido", 4 );
Dog b = new Dog( "Odie", 2 );
```

We can't sort the dogs by age the way it is now, because age is private, so we'd have to have getter methods, or we can use the Java libraries. So, let's rewrite *Dog*:

```
class Dog implements Comparable{
	private String name;
	private int age;

	public Dog( String name, int age ){
		this.name = name;
		this.age = age;
	}
	public int compareTo( Object other ){
		Dog d = (Dog)other;
		return this.age - d.age;
	}
}
```

<u>Three Comparable States</u>

With the *compareTo* object, we need to return one of three things:

- if the objects are equal:
  - *return 0;*
- if <u>this</u> is greater than other:
  - return a positive number
- if <u>this</u> is less than other:

- return a negative number

Now that we have this in our class, we can do this:

```
Dog a = new Dog( "Fido", 4 );
Dog b = new Dog( "Odie", 2 );
System.out.println( a.compareTo(b) );
//this will return 2, as it is 4 - 2
```

Or, in a larger example:

```
import java.util.Arrays;

Dog[] myDogs = new Dogs[5];

myDogs[0] = new Dog( "Fido", 4 );
myDogs[1] = new Dog( "Odie", 2 );
myDogs[2] = new Dog( "Lassie", 3 );
myDogs[3] = new Dog( "Spot", 1 );
myDogs[4] = new Dog( "Buddy", 5 );

Arrays.sort(myDogs);
//uses quick sort, able to do because our class Dog implements Comparable
```

Bubble Sort

Algorithm

```
void bubble_sort( int[] a ){
        while( true ){
                boolean swapped = fale;
                for( int i = 0; i < a.length - 1; i++ ){
                        if( a[i] > a[i+1] ){
                                int temp = a[ii];
                                a[i] = a[i+1];
                                a[i+1] = temp;
                                swapped = true;
                        }
                }
                if( swapped == false )
                        break;
        }
}
```

Gap Sort

- Based on the bubble sort

Algorithm:

1. Start with a large gap

```
gap = array.length -1;
```

2. Compare all elements “gap” units apart
3. Reduce 'gap' by 1
4. go back to beginning
5. gap of 1 means one pass of the bubble sort

Code:

```
void swap( int[] a, int i, int j){
	int temp = a[i];
	a[i] = a[j];
	a[j] = temp;
}
void gap_sort( int[] a ){
	for( int gap = a.length - 1; gap >= 1; gap-- ){
		for int i = 0; i + gap < a.length; i++ ){
			if( a[i] > a[gap + i] ){
				swap( a, i, gap+i );
			}
		}
	}
}
```

Drunk Sort

- The most useless of sorts, but it's fun to know

```
void swap( int[] a, int i, int j){
	int temp = a[i];
	a[i] = a[j];
	a[j] = temp;
}
void drunk_sort( int[] a ){
	Random rng = new Random();
	boolean sorted = true;

	while( true ){
		for( int i = 0; i < a.length; i++ ){
```

```
                swap( a, i, rng.nextInt( a.length ) );
            }
            sorted = true;
            for( int i = 0; i < a.length - 1; i++ ){
                if( a[i] > a[i+1] ){
                    sorted = false;
                }
            }
            if( sorted )
                break;
        }
}
```