Custom Sorting

Arrays

Most of you already know how to use arrays. One helpful piece in contest programming is the Arrays class. It contains many static methods easing programming of arrays and ultimately makes your life easier. Here we will look at a few useful methods for getting the most out of arrays.

```
Arrays.fill(array, someValue);
Arrays.fill(array, fromInclusive, toExclusive, value);
```

Fill is a useful method in the arrays class. It, like many of the static methods in Arrays, is implemented several times in java to be useful on all its primitives. (e.g. long, int, char, byte, float, double, boolean)

I find this method particularly useful when filling in a memo table with -1 for dynamic programming solutions or filling distance arrays with large values for BFS or dijkstra's algorithm. The method simply sets all values in the array to a new value. The runtime is the same as a for loop over the array filling with values.

```
Arrays.copyOf(array, length);
Arrays.copyOfRange(array, fromInclusive, toExclusive);
```

These two methods allow you to copy each array or a portion of each array. If length is longer than the original array, the copy will be padded with zeros.

```
Arrays.toString(array);
```

This method is useful for debugging your program. Sometime it is useful to know the values of one of your arrays in the middle of your program. This handy function gets a string representation of the array for printing. All the primitives work great. If you have an array of objects then each object must have a toString method. You can implement that in the following way:

```
class MyClass
{
   public String toString()
   {
      return x+", "+y; // if x and y are instance vars in MyClass
   }
}
```

```
Arrays.sort(array);
Arrays.sort(array, fromInclusive, toExclusive);
```

This method is useful when trying to sort an array of data. For primitives, the array will be put in increasing order of value. Unfortunately, you cannot change the order arrays of primitives are sorted. Sorts in Java run in $O(n \log n)$ runtime. Sorting a large number elements will run in time for most problems.

For arrays of objects you can specify a comparator. This comparator will allow you to sort the array in a custom way ignoring the objects compareTo method.

This can be done in line in java, in this example we will reverse the order of how Strings are normally sorted.

```
String[] array = new String[n];
// Read stuff into array.
Arrays.sort(array, new Comparator<String>(){
      public int compare(String a, String b){
         return b.compareTo(a);
      }
   });
```

If you are building a custom object, it is also possible to implement your own compareTo function in the object. To properly do so, your class must implement the Comparable interface. The syntax for doing this is shown below in the first line of the class.

For example:

```
class Point implements Comparable<Point>
{
   int x, y;

   Point(int x, int y)
   {
      this.x = x;
      this.y = y;
   }

   // This method sorts points by x values and, if there is a tie, y
   // values.
   public int compareTo(Point rhs)
   {
      if (x == rhs.x)
         return Integer.compare(y, rhs.y);
      return Integer.compare(x, rhs.x);
   }
}
```

Now, once we've created our Point class. If we happen to have an Array of points, we can sort them as follows:

```
Point[] myPts = new Point[n];
// Read pts in.
Arrays.sort(myPts);
```

The cool thing is that if we need to sort the same data in two different ways, we can implement Comparable for one method of sorting and then pass in a Comparator (page 1) for the second sort. Or. We could simply do two separate Comparators as well.