PRINCETON UNIVERSITY

COS 226–Algorithms and Data Structures
Week 3: *Comparators & Sorting (Algorithms §2.1 and §2.2)*

Version: February 18, 2018

Exercise 1 – Comparables and Comparators

In the sorting algorithms seen in lecture, we have been using *static* comparators: given any two elements, such comparators will always provide the same answer. But comparators do not need to be static, and can instead depend on additional information. This exercise illustrates this notion by looking at a comparator that compares the distance of two given points p and q each to a third point w.

Recall that a comparator returns a negative value if the first parameter is smaller than the second, positive if the first parameter is greater, and zero if the values are equal.

A. Suppose we want to create a Comparator that compares two points based on their distance from some third point. Fill in the code below. You may find the back of the other page of this handout useful.

```
public static class DistanceComparator implements Comparator<______________> {
    Point2D ________;

    public DistanceComparator(___________) {
        _________ = __________;
    }

    public int compare(_______ p, _______ q) {
        double distToP = p.distanceTo(_______);
        double distToQ = q.distanceTo(_______);
        if (distToP < distToQ) return ____;
        if (distToP > distToQ) return ____;
        return ____;
    }
}
```

B. Now suppose we want to use our comparator to sort a list of Points called by their distance from the origin. Fill in the code below to accomplish this task.

```
Point2D[] points = getRandomPoints();
Point2D origin = ___________________________________;

Comparator<Point2D> originDistanceComparator = ______________________________;
Arrays.sort(points, ___________________________);
```

C. Summary: what method must a Comparable have? What method must a Comparator have? Above is an example of calling sort with a Comparator. If I want to sort with the Point2D compareTo method, how do I call Arrays.sort()?

Exercise 2 –

The column on the left is the original input of strings to be sorted; the column on the right are the strings in sorted order; the other columns are the contents at some intermediate step during one of the **6** sorting algorithms listed below. Match up each algorithm by writing its number under the corresponding column. Use each number exactly once.

nite	deni	deni	deni	deni	dint	dine	deni
rein	dent	dent	dent	dent	dine	deni	dent
deni	nite	ding	ding	diet	deni	dent	diet
dent	rein	grin	nite	dine	dent	edit	dine
rent	ding	nite	rein	ding	edit	ding	ding
ding	grin	rein	rent	rent	ding	grin	dint
grin	rent	rent	grin	grin	grin	dog	dire
ride	ride	ride	ride	ride	dog	dire	dog
rind	diet	diet	rind	rind	dire	diet	edit
diet	dint	dint	diet	nite	diet	dint	grin
dint	rind	rind	dint	dint	nite	nite	nite
ring	ring	ring	ring	ring	ring	ring	rein
dire	dine	dire	dire	dire	rind	rind	rent
dog	dire	dog	dog	dog	ride	ride	ride
edit	dog	edit	edit	edit	rent	rent	rind
dine	edit	dine	dine	rein	rein	rein	ring

(1) Original input

(2) Sorted

(3) Selection sort

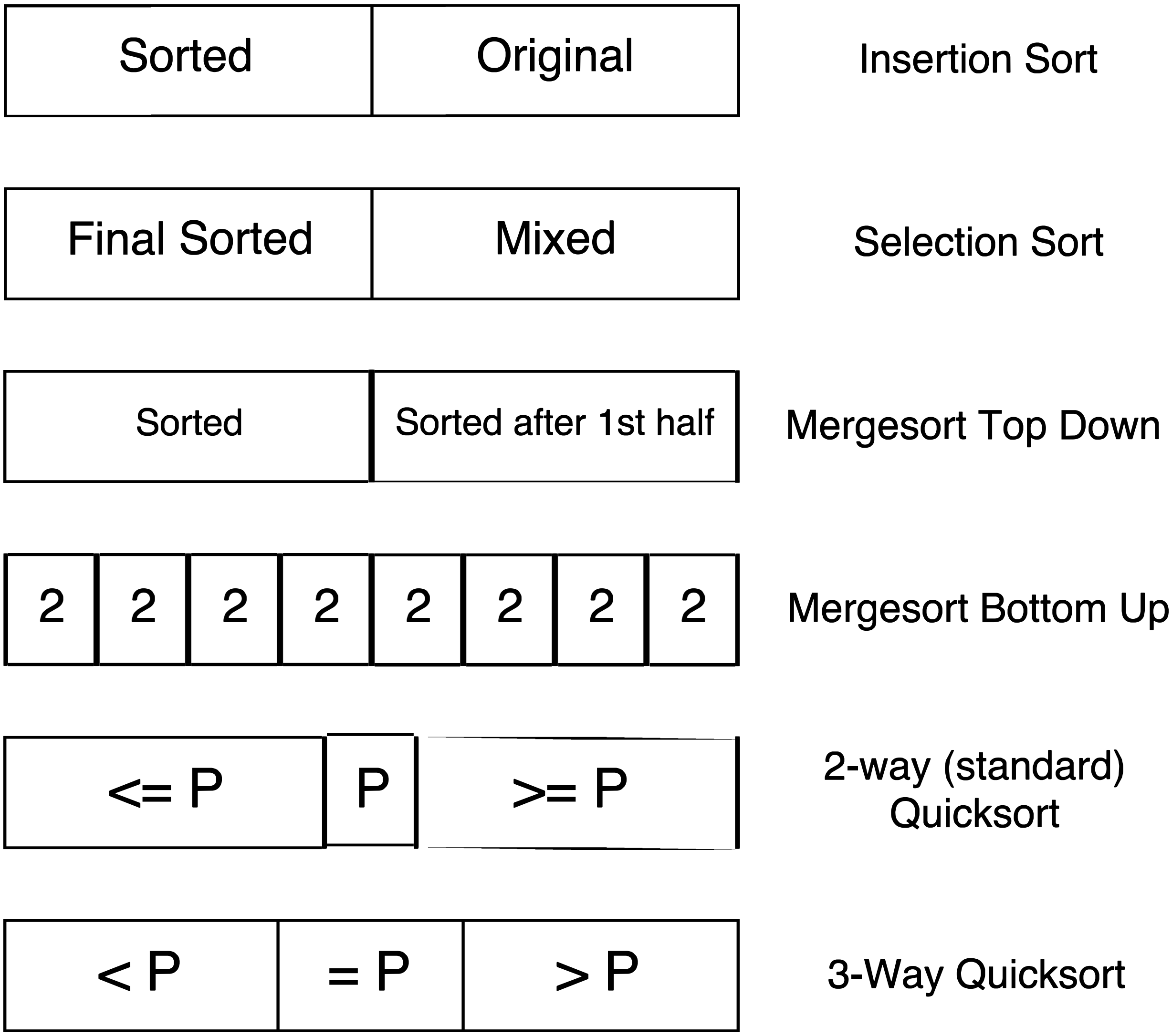
(4) Insertion sort

(5) Mergesort (*top-down*)

(6) Mergesort (*bottom-up*)

(7) Quicksort (*standard, no shuffle*)

(8) Quicksort (*3-way, no shuffle*)



Sorting Invariants

Exercise 3 – Counting Compares (Bonus)

Suppose that you have an array of length 2n consisting of n B's followed by n A's. Below is the array when n = 10.

B B B B B B B B B B A A A A A A A A A A

A. How many compares does it take to merge sort (ascending order) the array, as a function of n? Use tilde notation to simplify your answer.

B. The number of compares to 2-way quicksort is the same as if the elements were not sorted: $\sim (n(log_2 n))$

How many compares does it take to (3-way) quick sort (ascending order) the array, as a function of n? Use tilde notation to simplify your answer.