

**COS 226 – Data Structures and Algorithms**  
**Fall 2014 – Flipped Lecture Section**  
**Small Group Worksheet – 09.25.14**  
**25 minutes**

*Instructions: You can work individually or as a small group (2-3) to solve these problems.*

1. For an array of  $N$  items, in the best case, how many compares will Mergesort use in tilde notation? Give a very succinct explanation for what constitutes the “best case”.

**$0.5 N \lg N$ : Every item on one side is smaller than every item on the other side.**

2. In the worst case, how many compares will Mergesort use in tilde notation?

**$N \lg N$ . It's important to note that we only count `compareTo` calls and not calls to `<`.**

3. In the best case, how many array accesses does Mergesort use in tilde? (The answer is less than  $\sim 6N$ )

**Somewhere between  $4N \lg N$  and  $5N \lg N$ .**

4. Why does the Mergesort code use `less(a[i], a[j])` instead of `a[i] < a[j]`?

**Because `<` may not be defined for `A[i]` type data.**

5. Tough problem: Which of the following two loops is more likely to be faster? Why?
- |  |  |
|--|--|
| <pre>for (int i = 0; i &lt; N/4096; i++)<br/>    sum += a[i]</pre> | <pre>for (int i = 0; i &lt; N; i += 4096)<br/>    sum += a[i];</pre> |
|--|--|

**left is better. Since computers are good at accessing adjacent array entries using caching**

6. What was the point of teaching you bottom-up mergesort if it is always worse?

**no real reason, other than to annoy you ☹. But seriously, don't you want to know how the same thing can be done differently, iterative vs recursive, processing things from bottom of the tree vs top of the tree etc..**

7. Is comb sort ([http://en.wikipedia.org/wiki/Comb\\_sort](http://en.wikipedia.org/wiki/Comb_sort)) stable? If so, why? If not, give a counterexample.

**ANS: if you had a chance to see what it is, Nope. It has long distance exchanges; always dangerous. Consider [2 3 3 4] with comb size 2 then 1.**

8. Give two reasons why you might use a comparator instead of a built-in `compareTo` method?

**ANS: `compareTo` can work with natural order. Any comparable object must have `compareTo` method implemented and it is “the” way to compare. But `Comparable` gives us `compare` method to define more than one way to compare the same object.**

9. Consider the following Comparator, which is an inner class of a class called `student`:

```
private STATIC? class ByName implements Comparator<Student> {  
    public STATIC? int compare(Student v, Student w)  
    { return v.name.compareTo(w.name); }  
}
```

Under what circumstance would you want to remove the first static keyword? You might find <http://algs4.cs.princeton.edu/25applications/Student.java> useful. Why can you NOT remove the second static keyword?

**ANS:** This question is a bit annoying and stupid too. You'd remove the first if you wanted to save memory and did not need access to a calling object's state. We won't discuss dynamic comparators in this class, but they are a thing. The original version of this question was busted. You cannot have the second static keyword because you won't obey the Comparator API. In principle such a compare method could work if it didn't need any Comparator instance variables. See week3 precept for an example of a Comparator with instance variables.

10. Let's consider the problem of lower bounding the number of exchanges for exchange-based sorting: Given a single exchange, what is the maximum number of inversions that we can fix at once?

**ANS:** [9 8 7 6 5 4 3 2 1]. Swap 1 and 9. This fixes 16 inversions. Generally:  $2N-3$

11. Give an exchange-based sort algorithm that uses the minimum number of exchanges.

**ANS:** Selection sort. Only items out of place are moved, and they move to a target position only once

12. Is it possible for Merge Sort to behave as a quadratic algorithm for some bad set of inputs?

**ANS:** no way. Mathematically, it can never behave quadratic for any set of inputs. So it is consistently linearithmic

13. Given an array of size  $N$ , do you expect a partition operation or a merge operation on that array to be faster? Why?

**ANS:** Partition. Why? Look at the code! Merge is a rabid beast, writing everything three times! Partition barely does anything most of the time: A pointer just happily zips along.

14. Rewrite the insertion sort code below w/o using any exchanges

```
public static void sort(Comparable[] a)
{
    int N = a.length;
    for (int i = 0; i < N; i++)
        for (int j = i; j > 0; j--)
            if (less(a[j], a[j-1]))
                exch(a, j, j-1);
            else break;
}
```

```
public static void sort(Comparable[] a)
{
    int N = a.length;
    for (int i = 0; i < N; i++) {
        int temp = a[i];
        for (int j=i; less(temp,a[j-1]) && j > 0 ; j--);
        A[j] = temp;
    }
}
```