

### Flipped Session Short Answer – Week 3

Instructions: Work with a group (2-6 people) to answer as many questions as possible.

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”.
2. In the worst case, how many compares will Mergesort use in tilde notation?
3. In the best case, how many array accesses does Mergesort use in tilde? (The answer is less than  $\sim 6N$ )
4. Why does the Mergesort code use `less(a[i], a[j])` instead of `a[i] < a[j]`?
5. Tough problem: Which of the following two loops is more likely to be faster? Why?  

<code>for (int i = 0; i &lt; N/4096; i++)</code>	<code>for (int i = 0; i &lt; N; i += 4096)</code>
<code>sum += a[i]</code>	<code>sum += a[i];</code>
6. What was the point of teaching you bottom-up mergesort if it is always worse?
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.
8. Give two reasons why you might use a comparator instead of a built-in `compareTo` method?
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?
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?
11. Give an exchange-based sort algorithm that uses the minimum number of exchanges.
12. Give an example of a non-sorted array that causes quadratic behavior for Quicksort with no shuffle.

13. Given an array of size  $N$ , do you expect a partition operation or a merge operation on that array to be faster? Why?
14. Why not just check to see if the array is sorted before quicksorting rather than shuffling?
15. Suppose that we're about to partition an array. Is it possible that shuffling the array before partitioning will actually make the partitioning process slower than it would have been without the shuffle? If so, give an example. If not, why?
16. For an array, provide a simple description of the worst possible pivot. The best possible pivot.
17. Why don't we always just choose the median as our pivot?
18. Which sort is likely to use more compares when sorting: quicksort or mergesort?
19. Suppose we have an array of size  $N$  with  $k$  unique items. Assume  $k$  is a constant relative to  $N$ . One way to find the median of such an array is to 3-way quicksort the array, then return the middle item. Critique this idea.
20. Compare the number of compares used by 2-way and 3-way quicksort (page 301). Assume all unique keys. What can we infer about their relative performance on such arrays from this information?
21. Java does NOT autobox arrays. How does the built in `Arrays.sort` deal with sorting of primitive arrays? Equivalently, how does it decide whether to use Mergesort or Quicksort?  
<http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html>
22. Run an experiment to determine if 3-way quicksort or standard 2-way quicksort is faster on random inputs. For some extra insight, edit the sorting algorithms to collect information on the number of compares.  
  

```
int N = 10000000;  
Double[] nums = new Double[N];  
for (int i = 0; i < N; i++)  
    nums[i] = (Double) StdRandom.uniform(0.0, 1.0);  
  
Quick3way.sort(nums);
```
23. A modified version of quicksort that counts the number of compares can be found at <http://joshhug/misc/QuickStats.java>. You may find it interesting to run experiments using this code.
24. Extra tricky: Prove that Quicksort without shuffling takes quadratic time on an almost sorted array.