

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”.
 $0.5 N \lg N$: 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 a **single merge operation** use in tilde?
 $5N \lg N$. May as well use this space to remind you that $\lg N$ is the same thing as $\log_2 N$.
4. Why does the Mergesort code use `less(a[i], a[j])` instead of `a[i] < a[j]`?
It uses `Comparables`. `<` is not defined for comparables.
5. Tough problem: Which of the following two loops is more likely to be faster? Why?

```
for (int i = 0; i < N/4096; i++)      for (int i = 0; i < N; i += 4096)
    sum += a[i]                      sum += a[i];
```

The one on the left. Memory locality speeds up code. See Wikipedia for more.
6. What was the point of teaching you bottom-up mergesort if it is always worse?
To give an alternate perspective on how things could be done (bottom vs. top, iterative vs. recursive).
7. Is comb sort (http://en.wikipedia.org/wiki/Comb_sort) stable? If so, why? If not, give a counterexample.
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?
3: You want multiple ways to compare. You want dynamic comparison. There is no natural order.
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** have the second static keyword?
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?
Example: [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.
Selection sort. Only items out of place are moved, and they move to a target position only once.

12. Give an example of a non-sorted array that causes quadratic behavior for Quicksort with no shuffle.
A sorted array with two items out of place.
13. Given an array of size N , do you expect a partition operation or a merge operation on that array to be faster? Why?
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. Why not just check to see if the array is sorted before quicksorting rather than shuffling?
You might still go quadratic (see #12 or precept questions from this week).
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?
Yes. You might have a partition that would have done one exchange, and you might instead do more. This is not necessarily a bad thing however, as it might mean your pivot ends up in the middle. For example, partitioning a sorted array involves no exchanges, but results in bad pivot behavior!
16. For an array, provide a simple description of the worst possible pivot. The best possible pivot.
Best pivot: Median. Worst pivot: Largest or smallest item.
17. Why don't we always just choose the median as our pivot?
Finding the median is too slow. There are three basic ways: Sorting (not a good step for a sorting algorithm). Tarjan's algorithm (OK from an order of growth point of view, but it is an expensive linear time algorithm that makes Quicksort overall slower than Mergesort). Quick select (partitioning the array is also a bad step for deciding how to partition).
18. Which sort is likely to use more compares when sorting: quicksort or mergesort?
Quicksort. In fact, it will always use more compares.
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.
Not good from a performance perspective. I'm going to avoid elaborating here since I really like this problem. This is a fine thing to do I suppose if you are in a hurry when programming.
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?
They perform the same number of compares. This tells us nothing. We need to consider things like the exchanges. Or even better, just run experiments.
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>
It has a bunch of janky looking special purpose signatures for each primitive type.

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);
```

Haven't tried it myself.

23. A modified version of quicksort that counts the number of compares can be found at <http://joshh.ug/misc/QuickStats.java>. You may find it interesting to run experiments using this code.

This was pretty fun last spring.

24. Extra tricky: Prove that Quicksort without shuffling takes quadratic time on an almost sorted array.

Will do later. TBA: Unknown. Feel free to write a proof and I'll use yours.