Module 10 - Discussion Prompt Questions
Jot down short answers for each prompt question as you go along. Review and update your answers after finishing and submit before Tuesday midnight. Click on "Module 10 - Discussion Prompt Questions" above to submit.

The prompt questions for this module are:

1. What are some key factors that differentiate the various sorts you have seen?

- Whether the order of the file affects the sort or not (insertion sort, bubble)?
- If a pivot is used, how is the pivot selected (quicksort)?
- does it require extra space? (merge sort, index sequential sort)
- time complexity avges. counting sort is O(n)
- Is it stable? Do tied items change order on the pass or not?
- does it require an array or linked list?

2. What would you describe as the core idea behind merge sort?

Combine separate pieces into one piece. That one piece is going to require extra space, however. For example, if you had a single file of size 16 records, we can pretend it's 16 files of size 1 piece. We do this because merging needs the pieces to be sorted and a piece of size 1 is trivially sorted so we pretend that there are 16 trivially sorted pieces. We then merge to 8 pieces of size 2. merge again so 4 pieces of size 4, etc. finally 1 pieces of size 16 which is totally sorted.

so pieces from 1 to 2 to 4 to 8 to n grows exponentially so there are log(n) levels/passes. That's why there are log(n) for the cost where k is the number of levels. Cost of single pass is n. That's why the avg cost is n * log(n) and order does not matter because we're still going to do the same amount of work.

3. What are some good methods for choosing a quicksort pivot, and why?

Pick 3 numbers, take the median. That way you don't wind up with the totally worst case scenario of an unbalanced piece with 0 data.

4. How are subfiles created for shellsort?

So variable k is the skip factor or a jump factor that tells us how far to move through the file to get the next item in the same subfile. Need an array for this for better performance.

For example in file: 5,4,3,6,7,8,9,2,0,1,3,4,5 and if k is 3, one subfile will contain 5,6, 9,1,5.

Then we run insertion sort on the subfiles created. k represents the jumps to get next item in subfile but also the number of the subfiles.

After a pass, we take the next lowest k in the recurrence relation. The recurrence relation is an ideal way to pick k values.

5. In what way does a natural merge exploit existing order in the dataset?
The algorithm for picking subfiles involves adding numbers to the subfile if it is increasing. A new subfile begins when the next record decreases in value. So if the file is in order, the cost of

sorting is just n… no merging is required. A single pass through the data results in 1 subfile and that's exactly when you stop merging.

6. Describe in your own words the notion of a stable sort.

A stable sort is one that does not reorder duplicates or equal values in the output array. In the output, duplicate items stay in the same order as they were in the input array.

7. What difference might it make if the records you are sorting are very large or very small?

If the number of objects is small enough to fits into the main memory, this type of sorting is called internal sorting. If the number of objects is so large that some of them reside on external storage during the sort, the sorting would have to be done externally. How would external sort be done? Break them into pieces, bring them into internal (main memory), sort them, then consolidate them back together probably using a merge sort.

8. In considering a sorting problem, how important is it to you that some of the data is likely to be ordered?

It is important if the algorithm benefits from ordered data. Insertion sort comes to mind and so does natural merge sort. Some algorithms actually get worse from ordered data. For some algos, it doesn't matter like shell sort and straight 2 way merge; the works needs to be done the same way regardless of the ordering.

9. Have you been in a pre-demo development crunch? What were the key decisions you faced, and how did you address them?

A common question that comes up is if the extra complexity is worth the time? A lot of the decisions come to how long something will take (development time). Also, how much something will cost (development time, server cost, database cost) I assume sorting problems come with the same set of issues: cost and time. Some sorts like external sorts with merge sort require extra space… can we manage the cost?

10. What are the risks associated with having the wrong algorithm in place as a long-term solution?

If the sorting algorithm is time complexity of n^2, then the time to sort will grow exponentially larger as n grows. That's one risk. It may not be a problem when the data is small, but the penalty grows exponentially as the data grows. It may not scale well as the data grows. Some sorts are only good on small files as well because of the extra cost of the auxiliary space like quadratic selection sort

To grader: Is this right?