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 4 Discussion Prompt Questions" above to submit.

The prompt questions for this module are:

1. A lot of things have a FIFO characteristic. What is something that does not have a FIFO characteristic that you could still effectively manage with a queue?

If order does not matter, a queue might still be fine to use. For example, e-commerce order processing for Amazon. It doesn't matter in what order the orders are processed as long as the orders get processed. There are rare edge cases in which perhaps the last item is shipped and there isn't an item left for a subsequent order… but these cases are rare and the customer ultimately doesn't know that they placed their order prior to another. So orders can be placed in a queue and processed and it's a more than adequate solution.

2. What are some of the key distinguishing features of a list? Under what conditions would these be useful?

- all locations are available for insertion and deletion, unlike stacks and queues which limit where you can insert and delete.
- ordered collection
- no limit on number of items
- no limit on nature of items.

A list is useful when needing to access any location and this is what distinguishes a list from a queue or stack which have access restrictions. If all items need to be accessed and not just the first or the last, a list might be a useful data structure over a stack or a queue. For example a menu would lend well to a list. We don't want to just access the first or last items of a menu but all of them since they all need equal access.

3. Should a list ADT include hints or references to the list implementation?

No. That would be strange? I think the ADT supports inserts and deletes and optionally isEmpty and search but the underlying implementation (array or linked list) shouldn't be necessary.

4. What might you consider to be a nice-to-have in a list ADT (not necessary, but potentially useful)?

Optional useful methods would be reverse (reverses the order of the list) and empty (empties out a list) and sort (and pass it some sort logic code so it can sort by that logic like longest or shortest length).

5. When might the implementation details of a list be an important consideration?

Not sure. I guess a key difference between array and linkedList is the size limitations. Linked can be dynamically allocated whereas array has a static size. It'd be important to know how a list is implemented if size considerations are important to the user of the list.

6. List some pros and cons of an array implementation of a list.
Cons:

- statically allocated.
- for the standard implementation of the list, inserts requires elements to be shifted. Same with deletes. This takes O(n/2) on average.
- If you use an alternative array implementation of a list, you can at least make deletes O(1) time by just marking a spot deleted with a flag or using a bit to denote that the space is empty.

Pros:
- you can exploit random access because it is an array. 4th item in list is index 3.

7. List some pros and cons of a linked-list implementation of a list.
pros: no size limits because dynamic allocation.

cons: sequential access for search so searching is O(n)