Linked List Cheat Sheet

Definition

- A linked list is a sequence of data structures that are all connected together. Each link contains a connection to another link.
- A linked list can only be accessed from the head

What are some common questions we should ask our interviewer?

- Is it a singly linked list or a doubly linked list?
- If the example given is sorted, is the list always sorted?
- Can I create a new list?
- What should the method return?
    - A pointer to a head? A brand new list? Nothing?
- What kind of data does the linked list hold?
    - Might affect how you check for equality between strings, integers, etc
- Does the linked list have a maximum (or minimum) number of nodes?
- Could there be cycles in the linked list?

Are there any special techniques that we can use to help make this easier?

- Multiple passes
    - To find the length, or save other information about the contents
- Two pointers
    - ‘Race car’ strategy with one regular pointer, and one fast pointer
- Dummy node
    - Helpful for preventing errors when returning ‘head’ if merging lists, deleting from lists

Pseudocode

- Can you create any **magic** helper methods that would simplify the solution? (ie `getLength()`, `reverse()`)
- Talk through different approaches you can take, and their tradeoffs
- Be able to verbally describe your approach and explain how an example input would produce the desired output

Tips:

- Draw out node references
    - Helps to keep track of pointer and make sure you still have the pointer references you need
- Handle edge cases
    - Are there cases when you are having to return a null head?
    - Are there cases when you're not handling null pointers?

Questions to consider before being "done"

- Will my code handle a totally empty linked list?
- If there is a really high number of nodes, am I loading them all into memory?
- Will even vs. odd number of nodes affect my algorithm?
    - If there is an even number, where is the middle?

Time Complexity

	Best Case	**Worst Case**
Accessing/Searching	O(1)	O(N)
Inserting	O(1)	O(N)

Deleting	O(1)	O(N)

Note: Best cases occur when the node is at the head of the list, and worst cases occur when the node is at the end of the list