Phase 1: First Attempt

- Identify the problem
    - How many parameters are there?
    - What type of input values? What type of output values?
        - Numbers, characters, strings, sets, sequence, tuple, graph
    - Create a clear example of the inputs and outputs
        - Small example inputs
        - Boundary values (e.g., super large integers)
        - Uncommon values
            - Negative numbers, prime numbers
            - Whitespace characters, symbols and digits in strings
    - If it's strings
        - Am I allowed to treat strings as primitives, or will I have to consider string comparisons (which are linear time)
    - If it is a collection
        - Is it sorted/ordered?
        - Are duplicates allowed?
        - Is there a minimum size?
        - What is the element type?
    - Clearly restate the problem in my own words
    - Make a graphical representation of the problem
    - Is it a decision problem, search problem, or optimization problem?
        - Keep an eye out for the exact kind of output - often decision problems are easier than search problems.
- Define the context
    - What runtime are they expecting?
        - If it is constant time, I should think about:
            - Positional access of an array (all cases)
            - Accessing an element in a Hash Table (expected case)
            - Adding/Popping an element to a linked list
            - Check if an element is in a Hash Set
            - Checking the maximum/minimum of a heap
        - If it logarithmic time, maybe I need:
            - Self-balancing Binary Search Tree lookup/removal/addition
            - Removing/Adding an element to a Heap
            - Binary search of a linear data structure
            - Traversing to the bottom of a tree
        - If it is linear, I can probably use:
            - Linear data structures, like a list
            - Non-comparison based sorting algorithms
        - If it is linearithmic, I should think about:
            - Sorting the input
            - Constructing a balanced tree
        - If it is quadratic, I should think about:

            - Crosswise comparison of two linear structures
    - What variables is the expected runtime expressed in?
        - Do I know the type of the variable?
            - Is it the size of a collection (n)?
            - Is it the number of bits in a number?
            - Is it the size of the number itself?
            - Is it the number of nodes in a tree?
        - Are there more than one variables?
        - Do any of the variables have any relationships?
    - Are they telling me anything about the best/expected/worst case?
        - If the worst case is different than the expected case, I should think about data structures with different behavior in those cases:
            - Amortized structures (e.g., vectors)
            - Hash maps
    - Do they demand a specific time complexity, or are they using Big Oh?
        - If it's a specific time complexity, then I have to use very precise methods and really keep an eye on my operations.
        - If it's Big Oh, then I should keep in mind that I can let bigger operations dominate
            - If I sort, then I can do any number of linear operations as long as there are not a linear number of them.
    - What about space complexity?
        - If it's constant space, probably have to do swaps and work with the data structures they already give me
        - If it's logarithmic space, then I can think about something like a callstack or a tree exploration that doesn't require much additional space
        - If it's linear space, I can have a linear data structure, but I can also fit them into a tree.
        - If it's quadratic, I think about tables
    - What other constraints does the problem impose?
        - Do they limit the number of comparisons I should do?
            - Start thinking about trees and tournaments
            - Use a Decision tree to enumerate the possible universes of input
        - Do they reference specific data structures I can key in on?
- Explore possible strategies
    - What is the dumbest possible brute-force solution that might work?
        - Could I generate every possible candidate solution and test if it works?
    - Is the data unsorted? What happens if I sort it?
        - Is the collection just integers? Can I use a non-comparison based sorting algorithm?
    - What happens if I put things into a Hash Map?
        - Can I have integer indices and use an array instead?
        - If I need to have multiple things tied to a key, should I use a set or a list?
    - What happens if I put things into a tree?

        - I can often get a win if I have to end up traversing to the bottom of a tree instead of across all elements.
        - If I have to traverse, what order should I do it in?
    - Can I describe the problem with a graph?
        - What are the vertices and the edges?
        - Can I flip things around so that what I thought of as edges are now vertices?
    - Do I need to search a collection?
        - Is there any way I can build up the collection to have the value that I want, in a convenient location (e.g., the front or back).
        - What happens if I keep pointers between elements, can I short circuit having to do repeated traversals?
    - If I'm being restricted by time complexity, can I flatten operations to not happen inside of loops?
    - Can I swap out for a more/less restricted data structure?
        - E.g., can I replace a list with a Stack/Queue/Deque?
        - Does a Tree have to be self-balancing, or can I build it to be balanced as I go?
    - What if I reframe the problem recursively?
        - What are subproblems?
        - What is the base case?
        - How do I split up the problem?
        - How do I recombine the problem?
        - Where is the recursion?
- Act on the best solution
    - Let me try my solution on the example inputs I made before, carefully walking through and keeping mental track of the correctness and efficiency.
- Reflection
    - I found a solution, but was it fast enough?
    - Are any steps unnecessary?
    - Can I find a counterexample that proves my solution is wrong?

Phase 2: Take a Break

- Ideally sleep for 8 hours

Phase 3: Second Attempt

- Sit down and repeat Attempt 1 on all of this, limited to at least 30 minutes.
- Google the problem description that I have.
- Crack open the textbook and see if they have a relevant algorithm for this problem.

Phase 4: Reflection

- Ask myself:
    - What worked well?
    - What failed?