Module 4 Workbook

This week we are focusing on hard interview questions. The fact of the matter is that if you understand the strategies you need to understand these problems, you will be able to solve any problem.

Not to mention, just picture how much more confident you'll feel this week after solving all of these hard problems!

Remember from the lessons this week the 3 reasons why problems are hard:

1. Weird edge cases
2. Non-obvious optimizations
3. Lots of moving parts

Probably the biggest win that you will find this week is just focusing on not getting overwhelmed. When a problem is difficult, you just need to break it down into manageable chunks.

We'll practice this in the exercises this week.

Table of Contents

Day 1

Today, we are going to practice breaking down problems into more manageable chunks. We covered several examples of this in Module 4 Video 2.

For each of the following problems, your goal is simply to break it down into a series of simpler problems. There's no absolute right answer here, but you want to define the problem as a series of simpler functions whose solutions are trivial.

Tips:

1. You might want to think of this as writing pseudocode
2. We're looking for a working solution, not optimal solution
3. Make sure that you easily understand how to implement all of the sub-functions. In the example below if you're not sure how to reverse a linked list, you may want to break that itself down into a series of smaller functions.

Example:

Q: Write a function to print a linked list in reverse order
A:

```
func printReverse(list) {
      reverseList(list)
      printList(list)
      reverseList(list)
}
```

In this example, we're just defining two simpler functions, one to reverse the list and one to print the list.

Problems:

1. Justify text (Leetcode)
2. Calculator (Leetcode)
3. Word Ladder (Leetcode)

Day 2

Today we're going to take the problems from yesterday and go through the entire interview framework. This will give you an opportunity to work through the whole process in its entirety!

It is super important that you actually follow the steps in order here. Remember that once you get good at the framework you can start making adjustments, but until that point you really want to follow it step by step.

As a quick reminder here is the process:

1. Understand the problem [3-5 minutes]
2. Find a brute force solution [5 minutes]
3. Optimize your solution [15 minutes]
4. Code your solution [15 minutes]
5. Test your solution [5 minutes]

I would encourage you to do this for as many of the problems from yesterday that you have time for. Do at least one problem (pick the one you found hardest) and do the others if you have time.

Problems:

1. Justify text (Leetcode)
2. Calculator (Leetcode)
3. Word Ladder (Leetcode)

Day 3

There are a handful of “tricks” out there that can get you to optimal solutions that aren’t necessarily obvious. Today, we’re going to go over some of these techniques. While these problems aren’t super likely to come up, it’s good to just make sure that you have a grasp on the strategies.

For each of these problems do the following:

1. Take 5 minutes to find a brute force solution and think about if there are any more optimal approaches you can come up with
2. Flip to the solution. Read through and understand as best as you can
3. Close the solution and attempt to code it up WITHOUT cheating and looking back at the solution
4. If you are able to do it, you’re good. That means you have at least a basic understanding of the strategy
5. If you aren’t able to code it up on you own, repeat steps 2-5.

Problems:

1. Ones in Binary (Leetcode)
2. First Missing Positive (Leetcode)
3. Find the Missing Number (Leetcode)
4. Random Linked List Node (Leetcode)
5. Median of Stream (Leetcode)

Day 4

Today, we're going to look at a couple of problems that have some tricky edge cases. With all of these problems, you'll need to make sure that you clearly understand exactly what is being asked and take the time to ensure that you're handling all the edge cases.

For each problem, you should work through the entire problem solving framework. Once you're done and have tested your code by hand, copy it verbatim into Leetcode and test your solution. If it fails any of the test cases, you want to be sure you understand where the mistake was and avoid making that in the future.

As a quick reminder here is the problem solving framework:

1. Understand the problem [3-5 minutes]
2. Find a brute force solution [5 minutes]
3. Optimize your solution [15 minutes]
4. Code your solution [15 minutes]
5. Test your solution [5 minutes]

Problems:

1. Search in a Rotated Array (Leetcode)
2. Sort a List (Leetcode)

Day 5

Today and tomorrow, we're going to practice working through a handful of problems from start to finish. However, unlike the beginning of this week, I'm not going to give you any more guidance of what to look for.

One of the biggest problems in interviews is that we're used to having a lot of information at our fingertips. We say "I'm going to practice graph problems" and when we actually go to solve the problem, we know to use a graph because that's why we're doing the problem.

However, in the actual interview, you won't have any guidance around how to approach the problem. You need to be prepared to figure out how to solve it and identify the challenges yourself. That's why we're starting to practice in a slightly more "random" way.

For each of the problems, work through the entire framework. Make sure not to skip steps and don't give up before you give it a really solid try.

As a quick reminder here is the problem solving framework:

1. Understand the problem [3-5 minutes]
2. Find a brute force solution [5 minutes]
3. Optimize your solution [15 minutes]
4. Code your solution [15 minutes]
5. Test your solution [5 minutes]

Problems:

1. Split Array (Leetcode)
2. Add 2 Numbers (Leetcode)

Day 6

Today we're going to do some more of what we were doing yesterday.

For each of the problems, work through the entire framework. Make sure not to skip steps and don't give up before you give it a really solid try.

As a quick reminder here is the problem solving framework:

1. Understand the problem [3-5 minutes]
2. Find a brute force solution [5 minutes]
3. Optimize your solution [15 minutes]
4. Code your solution [15 minutes]
5. Test your solution [5 minutes]

Problems:

1. Longest Substring Without Repeating Characters (Leetcode)
2. 3 Sum (Leetcode)
3. K Closest Points (Leetcode)

Day 7

Rest day!

We've done a lot this week so I didn't schedule anything for you on this last day. Use this time to take a little break or catch up on anything you may have gotten behind on :)

Workbook Solutions

Day 1

1. Justify text (Leetcode, Video Walkthrough, Java Code, Python Code)
    a. Define the following functions:
        i. `getNextLine(words, startIndex, width)`
        This function gets as many words as it can from the array without exceeding the width
        ii. `justifyLine(words)`
        This function justifies a single line. It's easy to take a greedy approach here where we simply keep adding spaces between words from left to right until we reach the desired width
    b. With these functions, we just loop through the entire input and justify each line before combining them

2. Calculator (Leetcode, Video Walkthrough, Java Code, Python Code)
    a. Define the following functions:
        i. `findMatchingParen(expression, openParen)`
        This function finds the matching closed parentheses for any given open
        ii. `evaluateSubExpression(expression, start, end)`
        This function recursively evaluates the expression from start to end. Each time we find parentheses, we call findMatchingParen to find the close parenthesis and then recursively evaluate the subexpression
    b. We just iterate over our string. As we find literals, we keep a running sum, and if we find parentheses, we evaluate that subexpression first and then treat the result as a literal

3. Word Ladder (Leetcode, Video Walkthrough, Java Code, Python Code)
    a. Define the following functions:
        i. `validSteps(word, dict)`
        `validSteps` takes the word and finds all words in the dictionary that are exactly one word different from the starting word
    b. With `validSteps`, we can just use DFS to identify the path from our word to the desired result. `validSteps` gives us all the adjacent nodes in our "graph"

Day 2

Copy your code verbatim into Leetcode and keep track of any bugs.

1. Justify text (Leetcode, Video Walkthrough, Java Code, Python Code)
2. Calculator (Leetcode, Video Walkthrough, Java Code, Python Code)
3. Word Ladder (Leetcode, Video Walkthrough, Java Code, Python Code)

Day 3

1. Ones in Binary (Leetcode, Video Walkthrough, Java Code, Python Code)

   The key insight for this problem is that `n & (n-1)` removes the highest-order 1 bit from a number. What this allows us to do is solve the problem in time proportional to the number of 1s in the binary number.

   The code looks like this:

```
public int hammingWeight(int n) {
    int sum = 0;

    // Good to do != 0 because our number might be negative
    while (n != 0) {
        n = n & (n-1);
        sum++;
    }
    return sum;
}
```

2. First Missing Positive (Leetcode, Video Walkthrough, Java Code, Python Code)

   The basic idea here is that we want to identify whether a number is in the array by flipping the value at that index from positive to negative.

   To simplify, consider if the input only had positive numbers in it (remember how one of our strategies was to solve a simpler problem?). This will give us the same result anyway since we're only looking for the smallest *positive* integer.

   Now, we know that the missing number must be between `1` and N+1 (the length of the array). Therefore, let's go through our array and negate the indices from `0-N` for which that number is present in the array. Then we can quickly see what the missing number is.

Here's the code:

```
public int firstMissingPositive(int[] nums) {
    if (nums.length == 0) return 1;

    for (int i = 0; i < nums.length; i++) {
        if (nums[i] > nums.length || nums[i] <= 0)
            nums[i] = Integer.MAX_VALUE;
    }
    for (int i = 0; i < nums.length; i++) {
        int val = Math.abs(nums[i]);
        if (val <= nums.length) {
            if (nums[val-1] > 0) nums[val-1] *= -1;
        }
    }
    for (int i = 0; i < nums.length; i++) {
        if (nums[i] > 0) return i+1;
    }
    return nums.length+1;
}
```

3. Find the Missing Number (Leetcode, Video Walkthrough, Java Code, Python Code)

   While superficially, this looks similar to the previous problem, we will take a different approach because the numbers are consecutive.

   In this problem, we realize that the properties of XOR are that `x ^ x = 0` and `x ^ 0 = x`. This allows us to XOR a bunch of stuff together and come up with a result.

   Simply, we can XOR all of the values in our actual input and XOR that with what we would expect if there were no missing number. Everything will cancel out except the single missing number.

Here's the code:

```
public int missingNumber(int[] nums) {
    int expected = 0;
    int actual = 0;
    for (int i = 0; i < nums.length; i++) {
        actual ^= nums[i];
    }
    for (int i = 0; i < nums.length+1; i++) {
        expected ^= i;
    }
    return actual ^ expected;
}
```

4. Random Linked List Node (Leetcode, Video Walkthrough, Java Code, Python Code)

   The most efficient way to approach this problem is to use Reservoir Sampling. The basic concept is pretty simple. Each time we visit a node, we will select that node with the probability `1/N` where `N` is the number of nodes we've seen so far. Otherwise, we will keep the value from before.

   That means that for `N=1`, we set our random node with probability `1/1`. For our second node, we select that to replace the first node with probability `1/2` and so on.

   Here's the code:

```
public int getRandom() {
    Random rand = new Random();

    ListNode curr = list;
    int result = 0;
    int count = 1;
    while(curr != null) {
        if (rand.nextInt(count) < 1) result = curr.val;
        curr = curr.next;
        count++;
    }
    return result;
}
```

5. Median of Stream (Leetcode, Video Walkthrough, Java Code, Python Code)

   We need to keep track of all of the items that we've seen, but there is a relatively efficient way to do this.

   If we maintain a min and max priority queue, we can keep track of the middle of our stream at all times, which allows us to quickly determine the median.

   Here's the code:

```
public MedianFinder() {
    lowerHalf = new PriorityQueue<>(Collections.reverseOrder());
    upperHalf = new PriorityQueue<>();
}

public void addNum(int num) {
    lowerHalf.add(num);
    upperHalf.add(lowerHalf.poll());

    if (lowerHalf.size() < upperHalf.size()) {
        lowerHalf.add(upperHalf.poll());
    }
}

public double findMedian() {
    if (lowerHalf.size() > upperHalf.size())
        return (double) lowerHalf.peek();
    return (lowerHalf.peek() + upperHalf.peek())/2.0;
}
```

Day 4

Copy your code verbatim into Leetcode and keep track of any bugs.

1. Search in a Rotated Array (Leetcode, Video Walkthrough, Java Code, Python Code)
2. Sort a List (Leetcode, Video Walkthrough, Java Code, Python Code)

Day 5

Copy your code verbatim into Leetcode and keep track of any bugs.

1. Split Array (Leetcode, Video Walkthrough, Java Code, Python Code)
2. Add 2 Numbers (Leetcode, Video Walkthrough, Java Code, Python Code)

Day 6

Copy your code verbatim into Leetcode and keep track of any bugs.

1. Longest Substring Without Repeating Characters (Leetcode, Video Walkthrough, Java Code, Python Code)
2. 3 Sum (Leetcode, Video Walkthrough, Java Code, Python Code)
3. K Closest Points (Leetcode, Video Walkthrough, Java Code, Python Code)