Week 8

Additional Topics to Know

CSE 492 J




Dynamic Programming

- A method for optimizing algorithms over recursion
- Store subproblems of the main problem so we don't have to re-compute them when we need them later on in solving the main problem
- Google New Grad/FB FTE (not new grad) interviews

A little confusing? Don't worry, you are not alone!

Dynamic Programming

Take a recursive algorithm and find the overlapping subproblems, then cache the results for future recursive calls.

Brute Force solution is almost always recursive
Seems like you are making repetitive recursive calls
Interviewer asks you "how can this be better"

Run time is bad, sometimes even exponential

Let's Practice with CEAO-IT!

Question

Given a number `n`, print `n`-th Fibonacci Number.

How can we solve this problem in the most optimized way?




Clarify

1. Can `n` be a non-positive number?
   a. Depends, `n` can be 0, but not negative.
2. Can we use additional data structures?
   a. Yes, assume we want the fastest overall runtime.
3. What should be the result when `n = 0`?
   a. The result should be 0, before the first 1 in the sequence 1, 1, 2, ...,Fib(`n`)

Example

Edge Case 1:

`n = 0; Fib = None`

`Output = 0`

Edge Case 2:

`n = 1; Fib = 1`

`Output = 1`




Example

Middle Case 1:

```
n = 2; Fib = 1, 1
Output = 1
```

Middle Case 2:

```
n = 9; Fib = 1, 1, 2, 3, 5, 8,
                13, 21, 34
Output = 34
```




Approach

Brute Force:

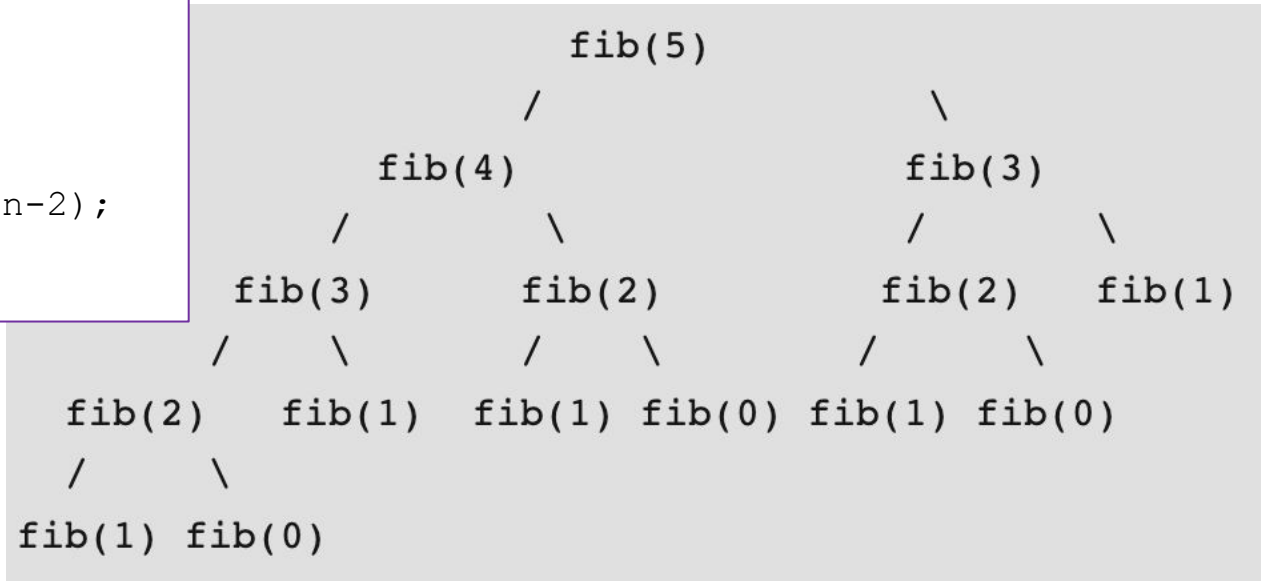
1. Use a recursive function to solve
2. Starting with `n` and descending down, recursively return the addition of the last and second last numbers of our sequence
3. End our recursion when we hit our base case `n = 1`

Has O(2^n) runtime with O(n) space complexity...

There is a faster way using Dynamic Programming...

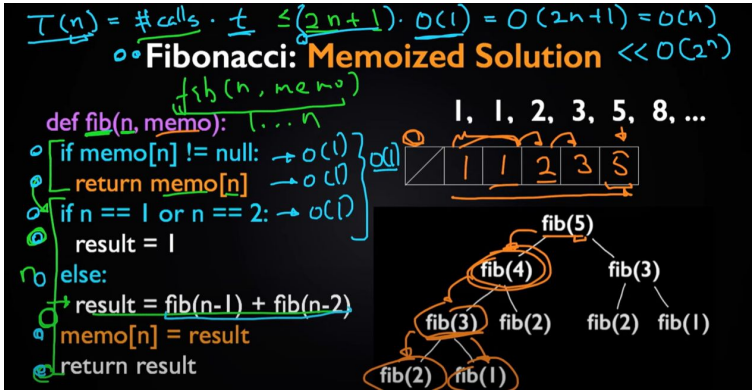
Recursive Solution

```
public static int fib(int n) {
    if (n <= 1) {
        return n;
    }
    return fib(n-1) + fib(n-2);
}
```



Dynamic Programming Solution

```
public int fib(int n) {
    int f[] = new int[n+1];
    f[0] = 0;
    f[1] = 1;
    for (int i = 2; i <= n; i++) {
        f[i] = f[i-1] + f[i-2];
    }
    return f[n];
}
```



https://www.youtube.com/watch?v=vYquumk4nWw

Optimize

- Optimized: Use Dynamic Programming to pre-compute Fib sequence up until `n` and return. Runs in O(N) runtime.
- Optimized no additional data structure: We compute the value of our current term with a fixed number of elements. O(1)
- Note: When you are computing a value in a sequence in an interview, think about using DP if applicable.

Implement

- Create 3 variables to hold our second last value, our last value, and our current value with respect to our current term in the sequence when iterating.
- Iterate in a for-loop until we hit term `n` and add our last and second last values and set them equal to our current value.
- When we exit the for-loop, we will have computed the Fibonacci value at `n`.

Implement - Java Code

```
static int fib(int n) {
    int a = 0, b = 1, c;
    if (n == 0)
        return a;
    for (int i = 2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}
```

Test

Test with Middle Case 2:

```
n = 9
Fib = 1, 1, 2, 3, 5, 8, 13, 21, 34
```

Resulting variable values after for loop ends:

```
a = 21, b = 34, c = 34
Return b = 34
```




Question

You are climbing a staircase. It takes `n` steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

How can we solve this problem in the most optimized way?




Clarify

1. Can `n` be a non-positive number?
    a. No, `n` must be equal to or greater than 1.
2. Can we use additional data structures?
    a. Yes, assume we want the fastest overall runtime.
3. Can we climb the same number of steps in a row?
    a. Yes, you can climb in any combination of 1 or 2 steps.




Example

Edge Case 1:

n = 1; 1. Take one step forward.

Output = 1

Edge Case 2:

n = 2; 1) 1 + 1 step 2) 2 steps

Output = 2

Example

Middle Case:

```
n = 3;
1) 1 + 1 + 1 steps
2) 1 + 2 steps
3) 2 + 1 steps
Output = 3
```




Approach

Brute Force:

1. Use a recursive function to solve
2. Starting with `n` and descending down, recursively return the sum of the combinations it took to get to the last and second last steps from our current step
3. End our recursion when we hit our base cases `n = 1, n = 0`

Has O(2^n) runtime with O(n) space complexity...unless...dynamic programming...

Optimize

- Optimized: Use Dynamic Programming to pre-compute the combinations of steps it takes to get to `n` and return. Runs in O(N) runtime.
- Optimized no additional data structure: We compute the value of our current step with a fixed number of elements (our last and second last step it took to get to our current step). O(1)
- Note: We are computing a value in a sequence, just like the Fibonacci problem...

Implement

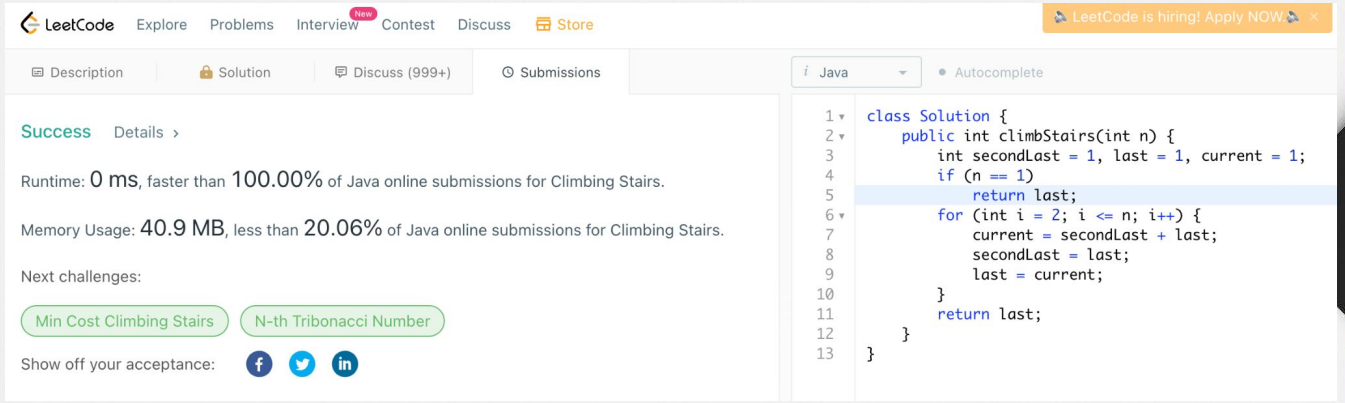
- Create 3 variables to hold the number of combinations it took to get to our second last step, our last step, and our current step with respect to our current step in the sequence when iterating.
- Iterate in a for-loop until we hit term `n` and add our last and second last steps' combinations and set them equal to the number of combinations to get to our current step.
- When we exit the for-loop, we will have the number of combinations to get to step `n`.

Implement - Java Code

```
public int climbStairs(int n) {
    int secondLast = 1, last = 1, current = 1;
    if (n == 1)
        return last;
    for (int i = 2; i <= n; i++) {
        current = secondLast + last;
        secondLast = last;
        last = current;
    }
    return last;
}
```

Fun Fact

The previous solution runs faster than 100% of leetcode solutions for the problem...which is the first time I have ever gotten a 100%...if you haven't been convinced of the power of DP yet...you should be now...



Test

Test with Middle Case:

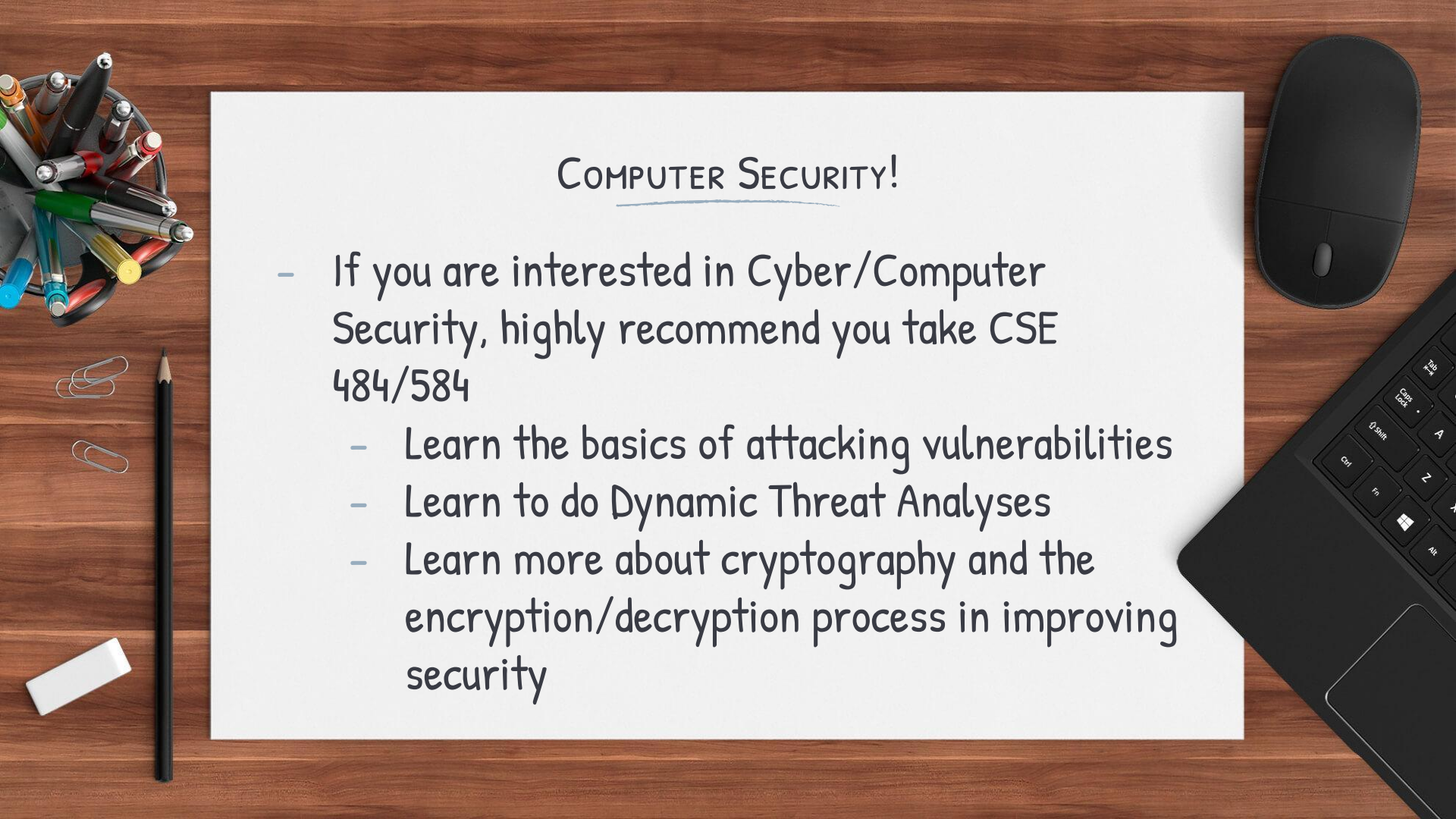
`n = 3`

`3 different ways to get to step 3`

Resulting variable values after for loop ends:

`secondLast = 2, last = 3, current = 3`

`Return` **`last = 3`**



Computer Security!

- If you are interested in Cyber/Computer Security, highly recommend you take CSE 484/584
  - Learn the basics of attacking vulnerabilities
  - Learn to do Dynamic Threat Analyses
  - Learn more about cryptography and the encryption/decryption process in improving security



General Interview Questions I Got in Cyber Security Interviews

- Where do you get your security news from? Can you tell me about a recent incident that you found interesting?
- What is MFA?
- If a friend asked you for advice on how to improve their cyber hygiene, what advice would you give them, and how would you walk them through the process?
- What are some of your goals as a cybersecurity professional?

Technical Interview Questions

Offense

Tell me about SQLi, and how to remediate it

Tell me about XSS, and how to remediate it

What's your favorite terminal command? What have you used it for?

Defense

How can you tell if a computer has malware on it?

What does a phishing email look like?

What are some of the scripts that you have been working on?

Tell me about a time where you automated a manual task.

Technical Interview Questions

Offense

Have you used Burpsuite before? If so, how have you used it?

How would you go about scanning a host for the services it is running/open ports?

What are some tools you've used to do reconnaissance on a webpage?

Defense

Incredibly simple coding questions (nothing harder than leetcode easy)

Have you ever worked with querying an API?

What's your favorite programming language, and why?

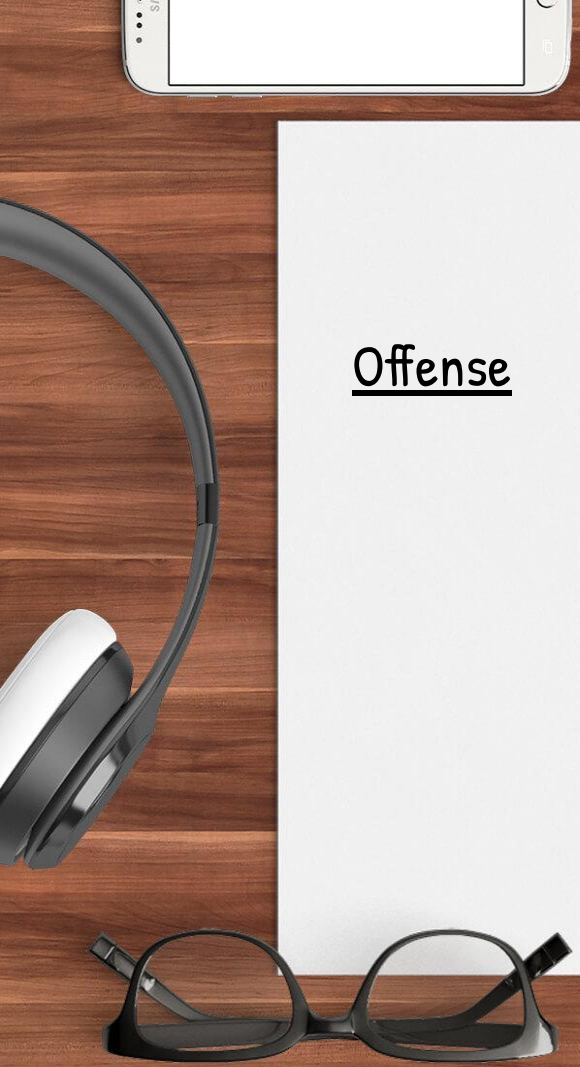
Technical Interview Questions

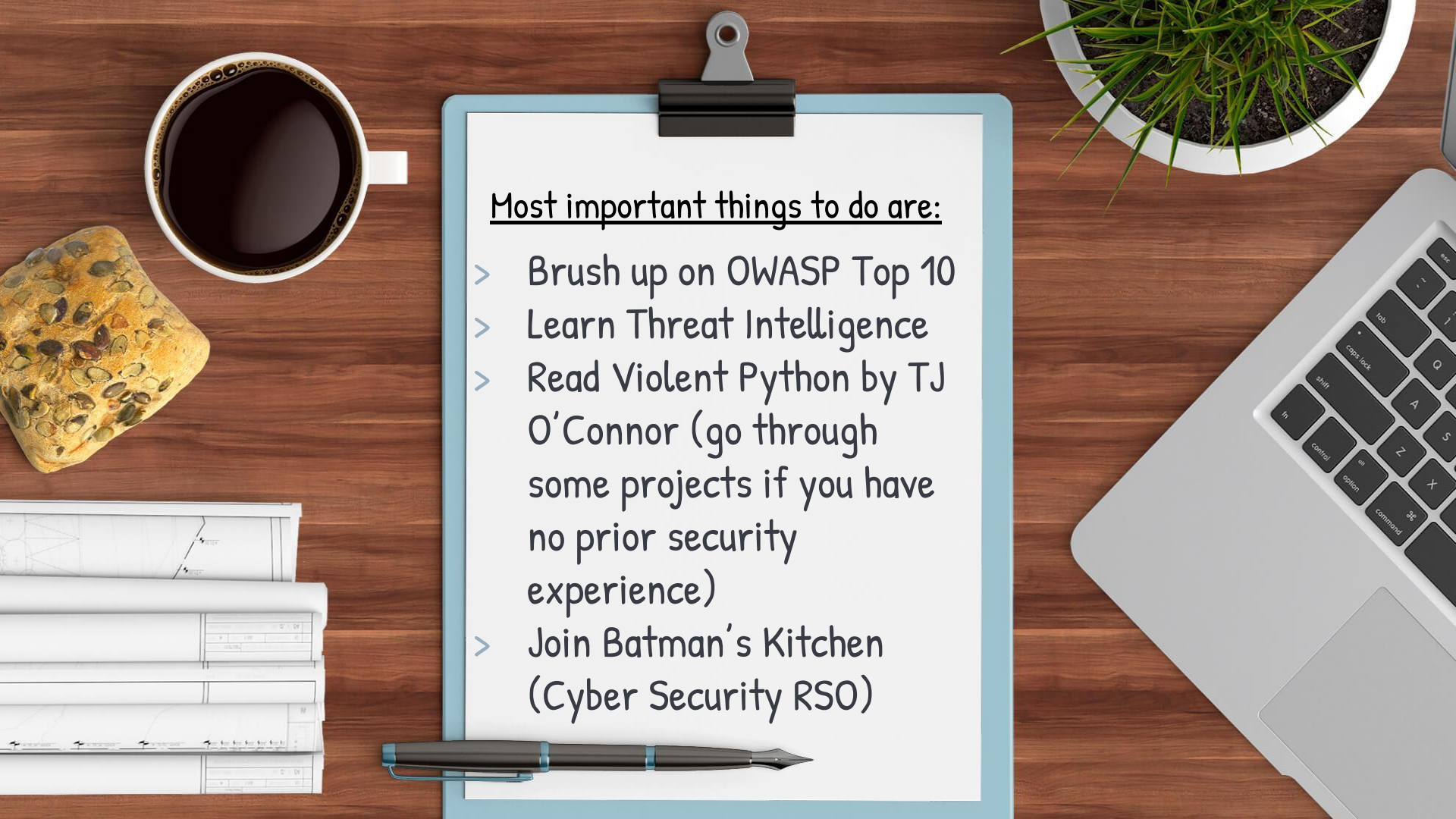
Offense

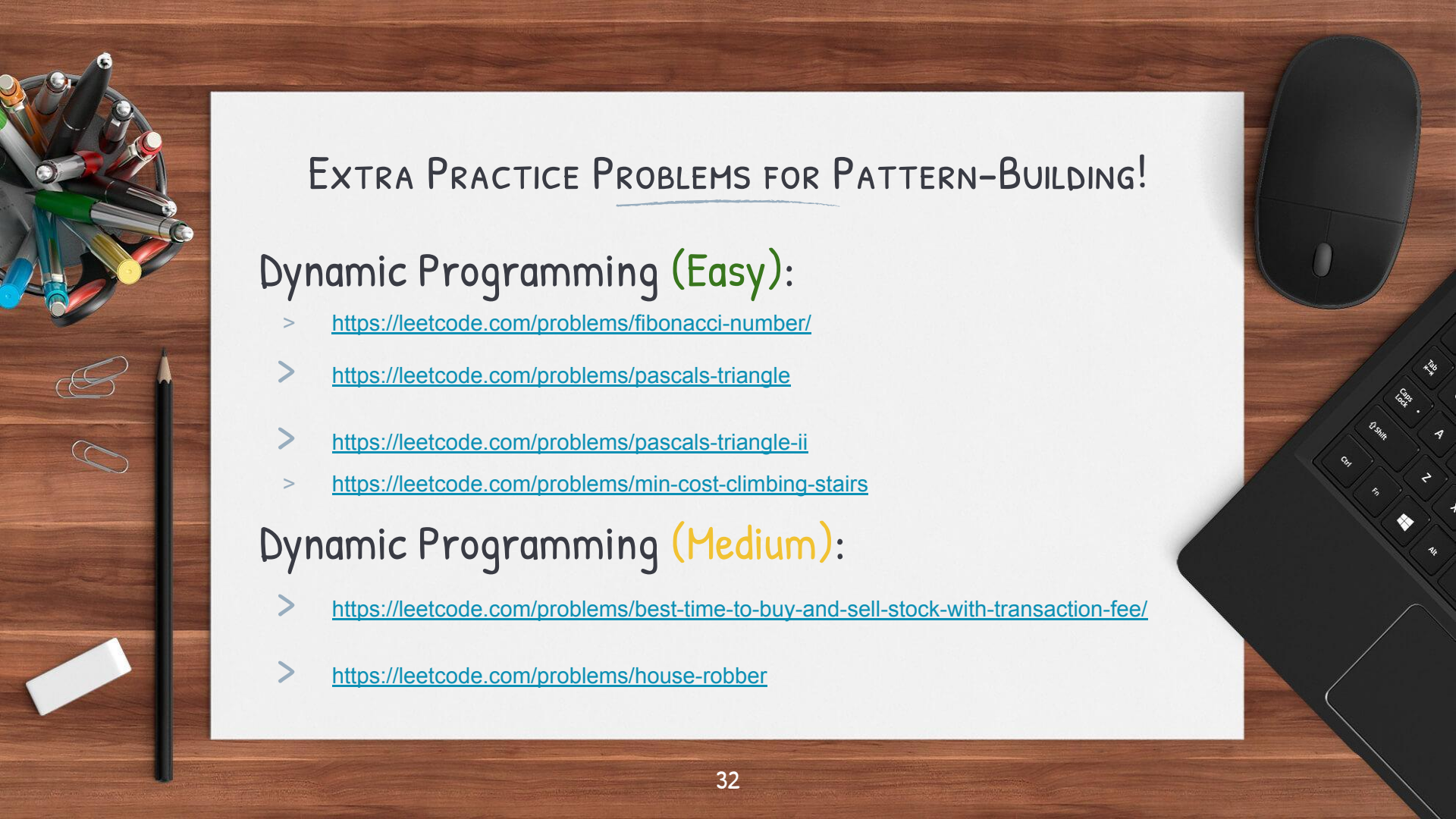
Defense

Do you know what a SIEM is? Have you worked with one before?

Do you have any experience with setting up logging on a network?




Most important things to do are:
> Brush up on OWASP Top 10
> Learn Threat Intelligence
> Read Violent Python by TJ O'Connor (go through some projects if you have no prior security experience)
> Join Batman's Kitchen (Cyber Security RSO)



Extra Practice Problems for Pattern-Building!

Dynamic Programming (Easy):

- https://leetcode.com/problems/fibonacci-number/
- https://leetcode.com/problems/pascals-triangle
- https://leetcode.com/problems/pascals-triangle-ii
- https://leetcode.com/problems/min-cost-climbing-stairs

Dynamic Programming (Medium):

- https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/
- https://leetcode.com/problems/house-robber

Bitshifting / Bitwise operations

Bit shifting is an operation done on all the bits of a binary value in which they are moved by a determined number of places to either the left or right.

- value "0001" or "1" is shifted left, it becomes "0010" or "2,"
- shifted to the left again it becomes "0100," or "4.
- Logical bit shifting may be useful for multiplying or dividing unsigned integers by powers of two

Java API on Bitshifting

Logical Shifts

A ***Left Logical Shift*** of one position moves each bit to the left by one. The vacant least significant bit (LSB) is filled with zero and the most significant bit (MSB) is discarded.

A ***Right Logical Shift*** of one position moves each bit to the right by one. The least significant bit is discarded and the vacant MSB is filled with zero.

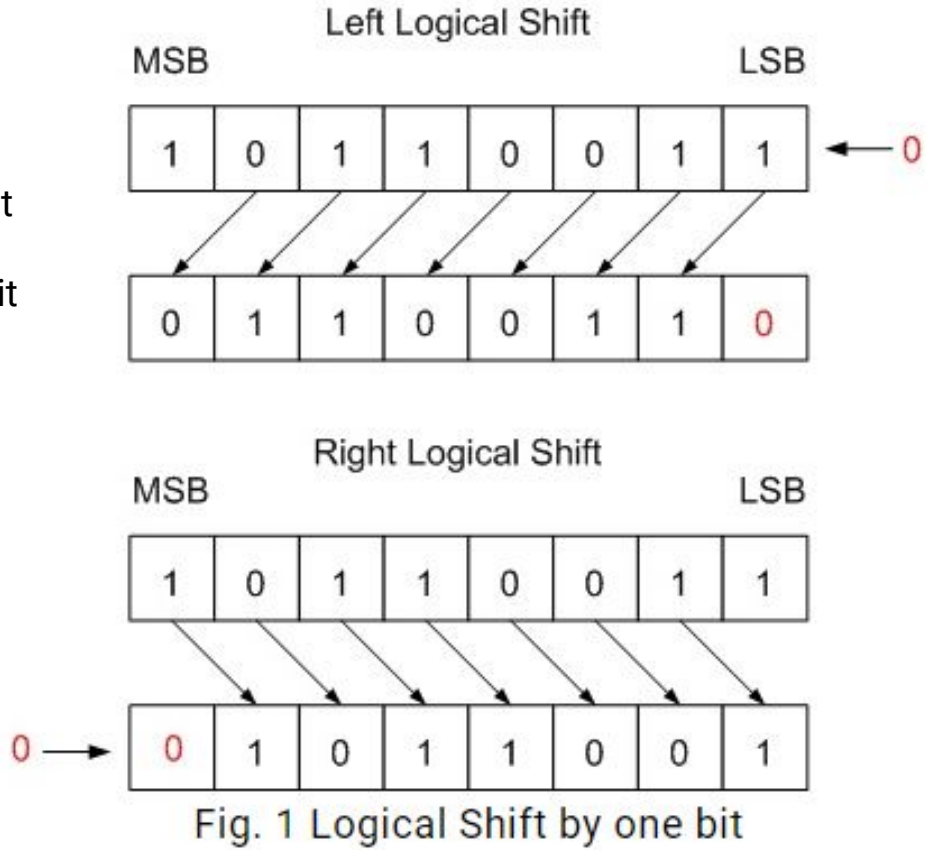
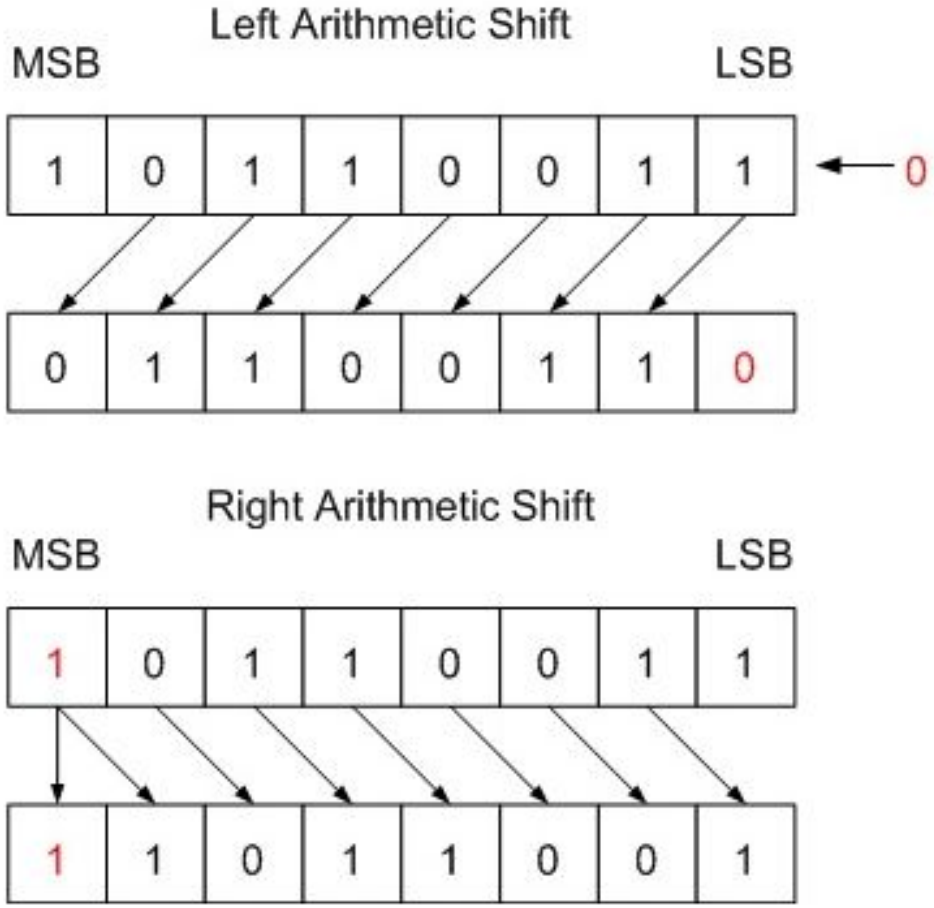


Fig. 1 Logical Shift by one bit

Arithmetic Shifts

A ***Left Arithmetic Shift*** of one position moves each bit to the left by one. The vacant least significant bit (LSB) is filled with zero and the most significant bit (MSB) is discarded. It is identical to Left Logical Shift.

A ***Right Arithmetic Shift*** of one position moves each bit to the right by one. The least significant bit is discarded and the vacant MSB is filled with the value of the previous (now shifted one position to the right) MSB.



Left Logical Shifts

When shifting left, the most-significant bit is lost, and a 00 bit is inserted on the other end

<< is the left shift operator

- 0010 << 1 → 0100
- 0010 << 2 → 1000

A single left shift multiplies a binary number by 2:

- 0010 << 1 → 0100
  - 0010 is 2
  - 0100 is 4

Right Shifts

Java provides two right shift operators:

>> does an **arithmetic** right shift

>>> does a **logical** right shift

Arithmetic shift

When shifting right with an **arithmetic right shift**, the least-significant bit is lost and the most-significant bit is *copied*.

- 1011 >> 1 → 1101
- 1011 is -5
- 1101 is -3
- 1011 >> 3 → 1111
- 0011 >> 1 → 0001
- 0011 >> 2 → 0000

Logical shift

When shifting right with an **logical right shift**, the least-significant bit is lost and the most-significant bit is *set to 0*.

- 1111 >>> 1 → 0111
- 1111 is -1
- 0111 is 7

Practice Problems