Finish the Runtimes

Here is a video walkthrough of the solutions.

Below we see the standard nested for loop, but with missing pieces!

```
for (int i = 1; i < ______; i = ______) {
    for (int j = 1; j < ______; j = ______) {
        System.out.println("We will miss you next semester Akshit :(");
    }
}
```

For each part below, **some** of the blanks will be filled in, and a desired runtime will be given. Fill in the remaining blanks to achieve the desired runtime! There may be more than one correct answer.
Hint: You may find `Math.pow` helpful.

(a) Desired runtime: $\Theta(N^2)$

```
for (int i = 1; i < N; i = i + 1) {
    for (int j = 1; j < i; j = ______) {
        System.out.println("This is one is low key hard");
    }
}
```

```
for (int i = 1; i < N; i = i + 1) {
    for (int j = 1; j < i; j = j + 1) {
        System.out.println("This is one is low key hard");
    }
}
```

Explanation: Remember the arithmetic series $1+2+3+4+\ldots+N = \Theta(N^2)$. We get this series by incrementing j by 1 per inner loop.

(b) Desired runtime: $\Theta(log(N))$

```
for (int i = 1; i < N; i = i * 2) {
    for (int j = 1; j < ______; j = j * 2) {
        System.out.println("This is one is mid key hard");
    }
}
```

Any constant would work here, 2 was chosen arbitrarily.

```
for (int i = 1; i < N; i = i * 2) {
    for (int j = 1; j < 2; j = j * 2) {
        System.out.println("This is one is mid key hard");
    }
}
```

Explanation: The outer loop already runs $\log n$ times, since i doubles each time. This means the inner loop must do constant work (so any constant `j <`

k would work).

(c) Desired runtime: $\Theta(2^N)$

```
for (int i = 1; i < N; i = i + 1) {
    for (int j = 1; j < ______; j = j + 1) {
        System.out.println("This is one is high key hard");
    }
}
```

```
for (int i = 1; i < N; i = i + 1) {
    for (int j = 1; j < Math.pow(2, i); j = j + 1) {
        System.out.println("This is one is high key hard");
    }
}
```

Explanation: Remember the geometric series $1 + 2 + 4 + ... + 2^N = \Theta(2^N)$. We notice that i increments by 1 each time, so in order to achieve this 2^N runtime, we must run the inner loop 2^i times per outer loop iteration.

(d) Desired runtime: $\Theta(N^3)$

```
for (int i = 1; i < ______; i = i * 2) {
    for (int j = 1; j < N * N; j = ______) {
        System.out.println("yikes");
    }
}
```

```
for (int i = 1; i < Math.pow(2, N); i = i * 2) {
    for (int j = 1; j < N * N; j = j + 1) {
        System.out.println("yikes");
    }
}
```

Explanation: One way to get N^3 runtime is to have the outer loop run N times, and the inner loop run N^2 times per outer loop iteration. To make the outer loop run N times, we need stop after multiplying `i = i * 2` N times, giving us the condition `i < Math.pow(2, N)`. To make the inner loop run N^2 times, we can simply increment by 1 each time.