```java
public static void twoForLoops(int n) {
    for (int i = 0; i < n; i++) {
        System.out.println("Printing: " + i);
    }
    for (int i = 0; i < 100; i++) {
        System.out.println("Printing: " + i);
    }
}
```

HINT: TO EXPRESS COMPLEXITY WE ASSUME N IS VERY LARGE

THE COMPLEXITY OF THIS OPERATION IS O(N), FOR VERY LARGE "N" THE CONSTANT 100 ASSOCIATED WITH THE SECOND FOR LOOP CAN BE IGNORED

O(N + 100) ~ O(N)

```java
public static void twoForLoopsNM(int n, int m) {
    for (int i = 0; i < n; i++) {
        System.out.println("Printing: " + i);
    }
    for (int i = 0; i < m; i++) {
        System.out.println("Printing: " + i);
    }
}
```

HINT: THERE ARE 2 LOOPS HERE, ONE OF LENGTH N AND THE OTHER OF LENGTH M, WHERE BOTH N AND M CAN BE VERY LARGE

THE COMPLEXITY OF THIS OPERATION IS O(N+M)

EACH FOR LOOP IS INDEPENDENT AND TAKES A DIFFERENT INPUT, SO THE COMPLEXITY IS ADDITIVE

```java
public static void twoForLoopsNAndM(int n, int m) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; i < m; i++) {
            System.out.println("Printing: " + (i * j));
        }
    }
}
```

HINT: NESTED LOOPS ONCE AGAIN BUT WORKING ON DIFFERENT INPUTS EACH OF WHICH CAN BE LARGE

THE COMPLEXITY OF THIS OPERATION IS O(N*M)

FOR EACH ITERATION OF THE LOOP GOING TO N, THE SECOND LOOP ITERATES M TIMES. NUMBER OF OPERATIONS = N*M

```java
public static void twoForLoopsNestedAndNonNested(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            System.out.println(String.format("Product of %s and %s is %s", i, j, i * j));
        }
    }
    for (int i = 0; i < n; i++) {
        System.out.println("Printing: " + i);
    }
}
```

HINT: THERE IS A NESTED AS WELL AS A SINGLE FOR LOOP HERE, REMEMBER WE ONLY CARE ABOUT THE HIGHEST ORDER TERM

THE COMPLEXITY OF THIS OPERATION IS $O(N^2)$

SINCE THERE ARE 2 LOOPS THE COMPLEXITY IS ACTUALLY $O(N^2 + N)$, LOWER ORDER TERMS ARE IGNORED

```java
public static void twoForLoopsNotCompletelyStraightforward(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = n; j > n / 2; j--) {
            System.out.println(String.format("Value of i: %s j: %s", i, j));
        }
    }

}
```

HINT: WORK OUT FOR EACH ITERATION OF THE OUTER LOOP, HOW OFTEN THE INNER LOOP WILL BE CALLED

Outer Loop	Inner Loop
0	0
1	1
2	2
3	2
N	N/2

WE CAN SEE THAT FOR EVERY N THE INNER LOOP EXECUTES N/2 TIMES

THE COMPLEXITY OF THIS CODE IS $N*(N/2) = O(N^2)$

```java
public static void doublingLoopVariable(int n) {
    for (int i = 1; i < n;) {
        System.out.println("Value of i is: " + i);
        i = i * 2;
    }
}
```

HINT: THIS IS NOT A SIMPLE INCREMENT, WHICH MEANS WE BREAK OUT OF THE LOOP MUCH FASTER THAN INCREMENTING BY A CONSTANT SUCH AS 1,2 ETC

LET'S TAKE THIS STEP BY STEP - WHAT IS THE VALUE OF i AT THE START OF EACH ITERATION?

FIRST ITERATION STARTS WITH THE VALUE OF $i = 1 = 2^0$

SECOND ITERATION $i = 1 \times 2 = 2 = 2^1$

THIRD ITERATION $i = 2 \times 2 = 4 = 2^2$

FOURTH ITERATION $i = 4 \times 2 = 8 = 2^3$

...

$(K + 1)^{TH}$ ITERATION $i = 2^{k-1} * 2 = 2^k$

THERE IS SOME NUMBER K FOR WHICH $2^K = N$

THIS K IS THE VALUE AT WHICH WE BREAK OUT OF THE LOOP

THE LOOP RUNS K TIMES

```java
public static void doublingLoopVariable(int n) {
    for (int i = 1; i < n;) {
        System.out.println("Value of i is: " + i);
        i = i * 2;
    }
}
```

HINT: THIS IS NOT A SIMPLE INCREMENT, WHICH MEANS WE BREAK OUT OF THE LOOP MUCH FASTER THAN INCREMENTING BY A CONSTANT SUCH AS 1,2 ETC

LET'S DERIVE THIS, THE VALUE OF `i` DOUBLES AT EVERY ITERATION

THERE IS SOME NUMBER K FOR WHICH

$$2^K = N$$

$$2^K = N$$

$$\log_2(2^K) = \log_2 N$$

$$K \log_2 2 = \log_2 N$$

$\log_2 2 = 1$, FOR BASE-2 LOGS

$$K = \log_2 N$$

REMEMBER THE LOOP RUNS K TIMES

I.E. THE LOOP RUNS LOG(N) TIMES

THE COMPLEXITY OF THIS CODE IS O(K) WHICH IS EQUIVALENT TO O(LOG N)

```java
public static void halvingLoopVariable(int n) {
    for (int i = n; i > 0;) {
        System.out.println("Value of i is: " + i);
        i = i / 2;
    }
}
```

HINT: ONCE AGAIN THE LOOP DECREMENT IS NOT A SIMPLE DECREMENT, WE HALVE THE LOOP VARIABLE AT EVERY ITERATION

THERE IS A SIMPLE RULE OF THUMB - IF N IS THE LENGTH OF THE INPUT

AND THE INPUT SPACE IS HALVED IN SOME WAY FOR EVERY ITERATION

THEN THE DERIVATION WE SAW EARLIER APPLIES

THE COMPLEXITY OF THIS CODE IS ONCE AGAIN O(K) WHICH IS EQUIVALENT TO O(LOG N)