```java
/**
 * TimeComplexity.java
 * Illustrates basic approach to characterizing a method's
 * time complexity.
 *
 */
public class TimeComplexity {

    // number of timing runs to make
    private static final int NUM_RUNS = 5;

    // 1.0E9 ns per second
    private static final double SECONDS = 1_000_000_000d;

    /** Drives execution. */
    public static void main(String[] args) {
        long start;
        long elapsedTime;
        double avgTime = 0d;
        int n = 4;
        System.out.printf("%4s%8s\n", "N", "Time");
        for (int i = 0; i < NUM_RUNS; i++) {
            start = System.nanoTime();
            methodToTime(n);
            elapsedTime = System.nanoTime() - start;
            System.out.printf("%4d %8.3f\n", n, (elapsedTime / SECONDS));
            n = n * 2;
        }
    }


    /** Method to be timed. */
    private static void methodToTime(int n) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    foo();
                }
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    foo();
                }
            }
        }
    }


    /**
     * Something that (hopefully) takes time >= 0.001 seconds
     * so that the program output looks better.
     */
    private static int foo() {
        int[] a = new int[100];
        java.util.Random rng = new java.util.Random();
        for (int i = 0; i < a.length; i++) {
            a[i] = rng.nextInt();
        }
        int sum = 0;
        for (int i = 0; i < 100_000; i++) {
            for (int val : a) {
                sum = sum + val;
            }
        }
        return sum;
    }
```

```
}
```