When analyzing the time complexity of our methods, we:

1. Identify the relevant input(s) to the method.
2. Identify the critical operation(s) that the method perform.
3. Give a mathematical model (i.e., a mathematical function) that relates the size of the input to the number of critical operations the method performs.

In this lab, we’ll gain practice going through this three-step process on a variety of examples.

## 1 Counting Crows

For each of the methods below, go through the three step process described above to develop a model of the program’s time complexity. Explicitly list the relevant inputs, critical operations, and the model itself.

```java
public static boolean boundedBy(int lo, int x, int hi) {
    return lo <= x && x <= hi;
}

public static long factorial(int n) {
    long result = 1;
    for (int i = n; i > 1; i--) {
        result = result * n;
    }
    return result;
}

public static void doubleEveryOther(int[] arr1) {
    for (int i = 0; i < arr.length; i += 2) {
        arr[i] = arr[i] * 2;
    }
}

public static int[] sumPairs(int[] arr1, int[] arr2) {
    if (arr1.length != arr2.length) { return null; }
    int[] ret = new int[arr1.length];
    for (int i = 0; i < arr1.length; i++) {
        ret[i] = arr1[i] + arr2[i];
    }
}
public static int[] allPairsProduct(int[] arr) {
    int[] ret = new int[arr.length * arr.length];
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr.length; j++) {
            ret[i * arr.length + j] = arr[i] * arr[j];
        }
    }
    return ret;
}

(Write down your answers to part 1 here.)
2 MCSS Revisited

Next go through the process of applying this three step process to build mathematical models for the maximum contiguous subsequence methods you explored in the previous lab. For convenience, here are the methods as they were originally presented:

```java
public static int compute1(int[] arr) {
    int max = 0;
    for (int i = 0; i < arr.length; i++) {
        for (int j = i; j < arr.length; j++) {
            int sum = 0;
            for (int k = i; k <= j ; k++) {
                sum += arr[k];
            }
            max = Math.max(max, sum);
        }
    }
    return max;
}

public static int compute2(int[] arr) {
    int max = 0;
    for (int i = 0; i < arr.length; i++) {
        int sum = 0;
        for (int j = i; j < arr.length; j++) {
            sum += arr[j];
            max = Math.max(max, sum);
        }
    }
    return max;
}

public static int compute3(int[] arr) {
    int max = 0;
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        sum = Math.max(0, arr[i] + sum);
        max = Math.max(sum, max);
    }
    return max;
}
```
Verify that the models that you develop for compute1, compute2, and compute3 are consistent with the data that you previously obtained.

(Write down your answers to part 2 here.)