Go to Moodle and open Analysis Lab for instructions

Exercise 1

Count:

```
for (x: Int in 1..4) {
   println(x)
}
```

4

```
for (x: Int in 1..<20) {
   if (x % 2 == 0) {
       println(x)
   }
}
```

9

1 ⇒ 9

```
for (x: Int in 0..<10) {
   for (y: Int in 1..x) {
       println("$x $y")
   }
}
```

Throws an error | 45

Exercise 2

Count and Function:

```
for (x: Int in 0..<n) {
   println(x)
}
```

$f(n) = n$

```
for (x: Int in 0..<n) {
   for (y: Int in 0..<n) {
       if (x != y) {
           println("$x $y")
       }
   }
}
```

$f(n) = n^2 - n$

```
for (x: Int in 0..<n) {
   for (y: Int in x..<n) {
       println("$x $y")
   }
}
```

(n)

(n-1)

~~$f(n) = n^2$~~

$f(n) = n^2 - n$

Exercise 3

```
fun pairStudents(students:List<String>){
   for (student1 in students){
        for(student2 in students){
             if (student1 != student2){
                  println("Pair: $student1 and $student2")
             }
        }
   }
}
```

a) $f(n) = n^2 - n$

b) $O(n) = n^2$, but it doesnt print n^2 amount of times, but as you get bigger its easier to "lose" the n

Exercise 4

```
public boolean binarySearch(int[] list, int item) {
   int first = 0;
   int last = list.length - 1;
   while (first <= last) {
        int midpoint = (first + last) / 2;
        if (list[midpoint] == item) {
            return true;
        } else if (list[midpoint] < item) {
            last = midpoint - 1;
        } else {
            first = midpoint + 1;
        }
   }
   return false;
}
```

~~1 or n~~

$f(n) = \log_2 n$ $(\log_2 n)$

It doesn't work → it has to be sorted to work

Extra

```
fun mergesort(A: MutableList<Int>, temp: MutableList<Int>, left: Int,
right: Int) {

   if (left == right) {          // List has one record
       return
   }

   val mid = (left+right)/2           // Select midpoint
   mergesort(A, temp, left, mid)      // Mergesort first half
   mergesort(A, temp, mid+1, right)   // Mergesort second half

   for (i in left..right) {    // Copy subarray to temp
       temp[i] = A[i]
   }

   // Do the merge operation back to A
   var i1 = left
   var i2 = mid + 1
   for (curr in left..right) {
       if (i1 == mid+1) {                      // Left sublist exhausted
           A[curr] = temp[i2++]
       }
       else if (i2 > right) {                  // Right sublist exhausted
           A[curr] = temp[i1++]
       }
       else if (temp[i1] < temp[i2]) {          // Get smaller value
           A[curr] = temp[i1++]
       }
       else{
           A[curr] = temp[i2++]
       }
   }
}
```

1) $f(n) = n \log n$

2) half the work → recursively

3) They are going to be "left" anyway