UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

ANALYSIS OF ALGORITHMS
PRACTICE 1

José Carlos Gualo Cejudo

Subject: Programming methodology
Docent: Jesús Fontecha Diezma

Index

First Function: Iterative

$$H(n) = n(2n - 1) = 2n^2 - n$$

Our solution in code for this method was:

```
public static void first_equation_iterative(int n) {
    int result = 0;
    result = n * (2 * n - 1);
}
```

In this equation, as we can see, the value of the function increments when the value of n increments, (always talking about positive values of n).

For our code, the time that takes to execute the method is more or less the same for all the values of n, so we can say that the complexity of this algorithm will be constant, so in Big Oh notation it will be represented as O(1).

Second Function: Iterative

$$\mathrm{H(n)} = \sum_{i=0}^{n-1} (4i + 1)$$

Our solution in code for this method was:

```
public static void second_equation_iterative(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        result += 4 * i + 1;
    }
}
```

When running this method, we could appreciate that the time that takes to execute it, was bigger when the value of is bigger.

Now, having a look to the code, we can see that there is a for loop which will be always executed n times. So, according to this information, we can define that the complexity of this algorithm in Big Oh notation will be O(n).

Second Function: Recursive

$$\mathrm{H(n)} = \sum_{i=0}^{n-1} (4i + 1)$$

As we can see, the function is the same as in the previous section, but in this case, the code to solve it is made in a recursive way:

```
public static int second_equation_recursive(int n) {
    int result = 0;
    if (n == 0) {
        return result;
    } else {
        result = ((4 * (n - 1) + 1)) + second_equation_recursive(n - 1);
    }
    return result;
}
```

As we can see looking at code, we have a base case which is reached when the value of n is equal to zero, otherwise, (if the value of n is not zero, and remember, it has to be always positive), we assign a new value to the variable result.
For calculating the complexity of this algorithm, we thought that the better way to do it was by expansion of the recurrence:

$$(4 \cdot (n-1)+1) + second\_equation\_recursive(n-1)$$

$$C(n) = C(n-1) + 3$$
$$C(n) = [C(n-2)+3] + 3 = C(n-2) + 6$$
$$C(n) = C(n-3) + 9$$
…
…
…
…
$$[C(n-n)+3] + 3(n-1) = C(0) + 3n = n$$

So finally, we would get that the complexity represented in Big Oh notation is O(n).