The Big-O Notation

Franck van Breugel

November 15, 2010

Mathematical Preliminaries

We use $\mathbb{N}$ to denote the set $\{0, 1, 2, \ldots\}$ of natural numbers and $f : \mathbb{N} \to \mathbb{N}$ to denote a function from natural numbers to natural numbers.

The symbol $\exists$ denotes existential quantification, that is, it expresses "there exists." The symbol $\forall$ denotes universal quantification, that is, it expresses "for all."

Estimating the Number of Semicolons

To estimate the complexity of a `main` method, we will associate a function $f : \mathbb{N} \to \mathbb{N}$ with that piece of Java code. This function f maps a natural number n to a natural number $f(n)$. The natural number $f(n)$ will roughly capture the answer to the question "How many elementary operations are performed at most when executing the `main` method for an input of size n?" Rather than defining what we mean by elementary operations, we will simply count the number of semicolons that we encounter when executing the code. Below, we will also make several simplifying assumptions. In other courses, we will be more precise.

Let us start with a very simple example.

```
1 public static void main(String[] arguments)
2 {
3    PrintStream output = System.out;
4    int number = arguments.length;
5    output.println(number);
6 }
```

In this example, the input consists of the command-line arguments. No matter how many command-line arguments we provide, the execution will always encounter four semicolons (to simplify matters, we will assume here that the code of the `println` method contains a single semicolon). Hence, in this case, the function $f_1 : \mathbb{N} \to \mathbb{N}$ is defined as

$$f_1(n) = 4.$$

As a second example, we consider the following `main` method.

```
1 public static void main(String[] arguments)
2 {
3    PrintStream output = System.out;
4    int number = arguments.length;
5    if (number > 0)
6    {
7       output.println(number);
8    }
9 }
```

In this case, the function $f_2 : \mathbb{N} \to \mathbb{N}$ is defined as

$$f_2(n) = \begin{cases} 2 & \text{if } n = 0, \\ 4 & \text{otherwise.} \end{cases}$$

As a third example, we consider the following `main` method.

```
1 public static void main(String[] arguments)
2 {
3    PrintStream output = System.out;
4    int number = arguments.length;
5    if (number > 0 && !arguments[0].equals(""))
6    {
7       output.println(number);
8    }
9 }
```

Again, to simplify matters, let us assume that the code of the `equals` method only contains a single semicolon. Note that in this case the number of semicolons encountered during the execution not only depends on the number of command-line arguments, but also on the fact whether the first command-line argument is the empty string. In such a case, we report an upperbound for the number of semicolons encountered, that is, the maximum number of semicolons encountered for a particular input size. In this case, the function $f_3 : \mathbb{N} \to \mathbb{N}$ is defined as

$$f_3(n) = \begin{cases} 2 & \text{if } n = 0, \\ 5 & \text{otherwise.} \end{cases}$$

Let us now consider an example that contains a loop.

```
1 public static void main(String[] arguments)
2 {
3    PrintStream output = System.out;
4    int number = arguments.length;
5    for (int i = 0; i < number; i++)
6    {
7       output.println(arguments[i]);
8    }
9 }
```

Let us first determine the number of semicolons for a number of different values for n, the size of the input (in this case the number of command-line arguments).

n	number of semicolons
0	4
1	7
2	10
3	13

From the above table we can conclude that in this case the function $f_4 : \mathbb{N} \to \mathbb{N}$ is defined as

$$f_4(n) = 3n + 4.$$

The Big-O Notation

Rather than determining the exact number of semicolons encountered during the execution when the input has a particular size (which in some cases is very difficult), we only approximate the number of semicolons and we classify the obtained functions into different classes. For that purpose, we introduce the so-called big-O notation.

Definition 1 *Let* $f : \mathbb{N} \to \mathbb{N}$ *and* $g : \mathbb{N} \to \mathbb{N}$ *be functions. Then* $f \in O(g)$ *if*

$$\exists M \in \mathbb{N} : \exists F \in \mathbb{N} : \forall n \geq M : f(n) \leq F \times g(n),$$

The above formula states that "there exist natural numbers M and F such that for all $n \geq M$, we have that $f(n) \leq F \times g(n)$." That is, we can find a particular "minimal size" M and a particular "factor" F such that for all inputs of size n that are greater than or equal to the "minimal size," we have that $f(n) \leq F \times g(n)$.

In the notation $O(1)$, the 1 denotes the constant function that maps each natural number n to one.

Proposition 2 $f_1 \in O(1)$.

Proof To prove that $f_1 \in O(1)$ we need to pick a particular M and a particular F. We pick $M = 0$ and $F = 4$. Then it remains to show that

$$\forall n \geq 0 : 4 \leq 4 \times 1,$$

that is, for all $n \geq 0$, we should prove that $4 \leq 4 \times 1$. This is obviously true. □

Similarly, we can show that $f_2 \in O(1)$ and $f_3 \in O(1)$. All those that belong to the class $O(1)$ are called constant time.

In the notation $O(n)$, the n denotes the identity function that maps each natural number n to itself.

Proposition 3 $f_4 \in O(n)$.

Proof To prove that $f_4 \in O(n)$ we need to pick a particular M and a particular F. We pick $M = 3$ and $F = 4$. Then it remains to show that

$$\forall n \geq 6 : 3n + 4 \leq 4n,$$

that is, $3n + 4 \leq 4n$ for all $n \geq 3$. Let n be an arbitrary natural number with $n \geq 3$. Then

$$3n + 4 \leq 3n + n = 4n.$$

Hence, the above is true. □

All those functions that belong to the class $O(n)$ are called linear time.

Comparing Methods

Assume that the user of the following `main` method always provides one or more integers as command-line arguments (this can be considered a precondition of the method).

```
1  public static void main(String[] arguments)
2  {
3     PrintStream output = System.out;
4
5     int smallest = 0;
6     int number = arguments.length;
7     for (int i = 0; i < number; i++)
8     {
9        boolean isSmallest = true;
10       for (int j = 0; j < number; j++)
11       {
12          isSmallest = isSmallest &&
13             (i == j ||
14             Integer.parseInt(arguments[i]) < Integer.parseInt(arguments[j]));
15       }
```

```
16          if (isSmallest)
17          {
18             smallest = i;
19          }
20       }
21       output.println(arguments[i]);
22    }
```

The `main` method prints the smallest command-line argument. The number of semicolons is roughly captured by the function $f_5 : \mathbb{N} \to \mathbb{N}$ is defined as

$$f_5(n) = 4n^2 + 5n + 5.$$

In the notation $O(n^2)$, the n denotes the identity function that maps each natural number n^2 to itself.

Proposition 4 $f_5 \in O(n^2)$.

Proof To show that $f_5 \in O(n^2)$, we pick $M = 6$ and $F = 5$. It remains to prove that

$$\forall n \geq 6 : 4n^2 + 5n + 5 \leq 5n^2,$$

that is, $4n^2 + 5n + 5 \leq 5n^2$ for all $n \geq 6$. Let $n \geq 6$. Then

$$4n^2 + 5n + 5 \leq 4n^2 + 5n + n = 4n^2 + 6n \leq 4n^2 + n^2 = 5n^2.$$

That concludes the proof. □

All those functions that belong to the class $O(n)$ are called quadratic.

An alternative way to determine the smallest command-line argument is implemented by the following `main` method.

```
1  public static void main(String[] arguments)
2  {
3     PrintStream output = System.out;
4
5     String smallest = 0;
6     for (int i = 1; i < arguments.length; i++)
7     {
8        if (Integer.parseInt(arguments[i]) < Integer.parseInt(smallest))
9        {
10          smallest = arguments[i];
11       }
12    }
13    output.println(smallest);
14 }
```

In this case, the number of semicolons can be estimated by the function $f_6 : \mathbb{N} \to \mathbb{N}$ is defined as

$$f_6(n) = 4n + 5.$$

Proposition 5 $f_6 \in O(n)$.

Proof To prove that $f_6 \in O(n)$ we need to pick a particular M and a particular F. We pick $M = 5$ and $F = 5$. Then it remains to show that

$$\forall n \geq 5 : 4n + 5 \leq 5n,$$

that is, $4n + 5 \leq 5n$ for all $n \geq 5$. Let n be an arbitrary natural number with $n \geq 5$. Then

$$4n + 5 \leq 4n + n = 5n.$$

Hence, the above is true. □

Since the first `main` method is quadratic time and the second `main` method is linear time, the second one will be more efficient for inputs of large size.