Topics of Mathematics in Cryptology

Asymptotic Notation

Dominique Unruh

March 30, 2017

1 The need for asymptotic notation

In this lecture, we will introduce the concept of asymptotic notation. First, we answer the question what an asymptotic notation is and what it is for. Assume, for example, that you are analyzing an algorithm. You are interested in the running time of this algorithm. Your analysis reveals:

- The subroutine $f(a)$ does at most a^2 multiplications of integers of length a.
- Besides that, $f(a)$ does some additional stuff. This additional stuff takes less time than a multiplications, except if $a < a_0$ (due to some fixed overhead). a_0 is approximately 1000, but you are not sure because computing the precise number would require much more work.
- The main program calls $f(a)$ for each $a = 1, \ldots, n^3$ and performs at most n^5 additional "elementary operations".

And now you ask: What is the running time of the main program in terms of n? Given the above, this is difficult to answer. $f(a)$ makes a^2 multiplications. How much time does a multiplication take? That depends strongly on the multiplication algorithm, let's call the time needed for a multiplication of length-a integers $m(a)$. So the time spent by $f(a)$ is at most "$a^2 m(a)$ + additional stuff". For $a \geq a_0$, the additional stuff takes less time than the multiplications. So for $a \geq a_0$, the running time of $f(a)$ is at most $2a^2 m(a)$.

The running time spent by all calls to $f(a)$ is then at most $\sum_{a=0}^{n^3} time(f(a))$. For $a \geq a_0$ we know a bound on $time(f(a))$. Thus $\sum_{a=0}^{n^3} time(f(a)) = \sum_{a=0}^{a_0-1} time(f(a)) + \sum_{a=a_0}^{n^3} time(f(a)) \leq \sum_{a=0}^{a_0-1} time(f(a)) + \sum_{a=a_0}^{n^3} 2a^2 m(a) =: A$

Now we can continue to compute the running time of the main program. It is A steps plus the running time of the "elementary operations". We don't really know how long an "elementary operation" takes, except that there is a constant bound e on that time (the precise value of e depends on the machine model). So $time(main) \leq A + n^5 e$.

Altogether a relatively complex analysis, with various parameters (e and m) that depend on the specific machine and implementation details, and with a complex summation

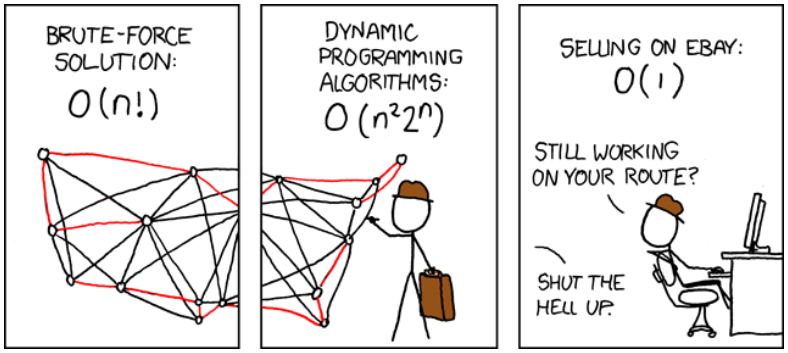

Figure 1: xkcd comic #399 by Randall Munroe, Creative Commons by-nc 2.5.

formula $\sum_{a=a_0}^{n^3} 2a^2 m(a)$ and with the annoying $\sum_{a=0}^{a_0-1} time(f(a))$ that is only relevant for small a's anyway.

Although such a detailed analysis may be required in some cases, in many cases we would be happy with an estimate that shows the essential complexity of the algorithm, without getting into too many details of the formula. This is possible using asymptotic notation. Anticipating the notation introduced later, the above analysis can be done as follows: For most (reasonable) multiplication algorithms, multiplying integers of length a takes time that is at most quadratic in a, up to some constant factor. (I.e., it is bounded by something like $23a^2$ or similar.) We say, multiplication has running time in $O(a^2)$. $f(a)$ does a^2 multiplications, so the running time of those is in $O(a^4)$. The "additional stuff" takes less than that, except for $a < a_0$. The asymptotic notation ignores finitely many exceptions ($a = 1, \ldots, a_0 - 1$), so the additional stuff also takes time $O(a^4)$. Hence $f(a)$ takes time $O(a^4)$. (No need to write $O(2a^4)$ since $O(a^4)$ means a^4 up to some factor.) The main program calls $f(a)$ for $a = 1, \ldots, n^3$. The running time of these calls is then $O(\sum_{a=1}^{n^3} a^4)$. Looking it up in a list of summation formulas, we see that $\sum_{a=1}^{n^3} a^4 = \frac{n^{15}}{5} + \frac{n^{12}}{2} + \frac{n^9}{3} - \frac{n^3}{30}$. But we do not even need that level of detail. The powers smaller than n^{15} can be ignored asymptotically, and the factor $\frac{1}{5}$, too. So $O(\sum_{a=1}^{n^3} a^4) = O(n^{15})$. Finally, the main program runs n^5 elementary operations. This takes time $O(n^5)$ (no need to know the time for an elementary operation, as long as it is bounded by a constant). So the total running time is $O(n^{15} + n^5)$. Since n^5 is irrelevant in comparison to n^{15}, we have that the total running time is in $O(n^{15})$. (Which, by definition of $O(\ldots)$, means that for some constant c, the running time is at most $c \cdot n^{15}$.)

We see that an analysis using asymptotic notation gives a much simpler result, and that the analysis itself needs to keep track of much less details. (Whenever some small terms occurred, we could just throw them away.)

Of course, for some applications we might need the precise number of steps. In this case, we cannot use asymptotic notation. But otherwise, asymptotics make life easier.

We stress that, given the definition of $O(\ldots)$ below, the above analysis is rigorous. All the "throwing away" of "irrelevant terms" is fully justified by the definition of O.

2 Big Oh

We now formally introduce the notion $O(\dots)$ used in the section before, called the "big Oh"-notation.[1]

In this and the following sections, we will only consider non-negative functions ($\forall x. f(x) \geq 0$). Definitions in the literature do not agree on the treatment of negative functions, so we avoid the issue here. (They are not often needed anyway.)

Definition 1 *Let $f, g : \mathbb{N} \to \mathbb{N}$ be non-negative functions. We say that $f \in O(g)$ if there is a constant $c \geq 0$ such that for all sufficiently large[2] n, we have that $f(n) \leq c \cdot g(n)$.* **O; big Oh**

In slight abuse of notation, the functions f, g are very often written as implicit functions of some variable, e.g., n. For example, instead of writing "$f \in O(g)$ where $f(n) := n^2$ and $g(n) := n^3$", we just write "$n^2 \in O(n^3)$". (That is, n^2 stands for the function that maps n to n^2.) If more than one variable is involved, one needs to pay attention to make clear which variable is the function argument.

Another common abuse of notation is to write, e.g., $n^2 = O(n^3)$ instead of $n^2 \in O(n^3)$. Personally, I do not like this notation. For example, in this notation both $n = O(n^3)$ and $n^2 = O(n^3)$ are true, but $n \neq n^2$. Formally, $O(g)$ is a set of functions (namely of all functions f that satisfy Definition 1), so it should be treated as a set.

Exercise 1 *Which of the following statements is true? (Justify your answer.)*

(a) $\log n \in O(n)$.

(b) $O(n) \in \log n$.

(c) $2 \in O(1)$.

(d) $1 \in O(2)$.

(e) $n^{23} + 10 + n^3 \in O(n^{23})$.

(f) $n^{23} \in O(n^{24})$.

(g) $12034273842374923 + n^{23} + 10 + n^3 \in O(12034273842374923)$. *(The long number is the same on both sides.)*

(h) $2^n \in O(n^{23})$.

(i) $O(n^{23}) \in O(n^{22})$.

[1] Originally, O actually referred to a big omicron (and should thus be written O, not O), not to a Latin O [Knu76]. However, due to the similarity of the Latin O and the Greek Omicron, the name "big Oh" now seems generally accepted.

[2] When we say, "for all sufficiently large n, $P(n)$ holds", this means "there exists an n_0 such that for all $n \geq n_0$, $P(n)$ holds".

An example on how to operate with big oh notation has been given in the preceding section. However, to make the argumentation given there formal, we need to know some important properties of $O(\dots)$.

Lemma 1 (Some properties of big Oh)

(i) *If* $f \in O(g)$ *and* $c \geq 0$ *then* $c \cdot f \in O(g)$.

(ii) *If* $f_1 \in O(g_1)$ *and* $f_2 \in O(g_2)$ *then* $f_1 + f_2 \in O(g_1 + g_2)$.

(iii) *If* $f_1 \in O(g_1)$ *and* $f_2 \in O(g_2)$ *then* $f_1 \cdot f_2 \in O(g_1 \cdot g_2)$.

(iv) *If* $f_1, f_2 \in O(g)$ *then* $f_1 + f_2 \in O(g)$.

(v) *If* $f_1 \leq f_2$ *(or at least: for all sufficiently large* n, $f_1(n) \leq f_2(n)$*) and* $f_2 \in O(g)$, *then* $f_1 \in O(g)$.

(vi) *If* $g_1 \leq g_2$ *(or at least: for all sufficiently large* n, $g_1(n) \leq g_2(n)$*), then* $O(g_1 + g_2) = O(g_2)$.

(vii) *If* $f \in O(g)$, *then* $\sum_{i=1}^{b} f(n_i) \in O(\sum_{i=1} g(n_i))$ *for any* b *and* n_i *(even if these values depend on* n*).*

Exercise 2 *Make the asymptotic analysis in the preceding section rigorous. That is, for each step in the argumentation, say which of the properties from Lemma 1 is used.*

Exercise 3 *Show Lemma 1* (i), (ii), (iv), (v).

Big Oh notation can also be used to easily define some important classes of functions. For example, $O(1)$ is the set of all bounded functions, i.e., of all functions f such that there is a c such that $\forall n. f(n) \leq c$.

And the set of all polynomially bounded functions (i.e., the set of all functions f such that $f \leq p$ for some polynomial p) is $\bigcup_i O(n^i)$.

Often, a notation like the following is used: $f \in n + O(\sqrt{n})$. (Or more generally, $f \in T$ where T is a term involving some normal computations on reals and occurrences of big O notation.) What is the meaning of that? It means that f is equal to $n + f'$ where $f' \in O(\sqrt{n})$. More formally: If $T(O(g))$ is a term involving an occurrence of $O(g)$, then $f \in T(O(g))$ means "there exists an $f' \in O(g)$ such that $f = T(f')$. Thus $f \in n + O(\sqrt{n})$ holds iff there exists a constant c such that for all sufficiently large n, we have $f \leq n + c\sqrt{n}$.

Exercise 4 *Which of the following holds?*

(a) $n + 2 \in n + O(1)$.

(b) $2n \in n + O(\log n)$.

(c) $n + 10 \log n \in n + O(\log n)$.

3 Big Omega

The big Oh notation above allows to show that a function is *at most as* big as a certain value. There is an analogous notation to show that a function is *at least as* big as a certain value:

Definition 2 *Let $f, g : \mathbb{N} \to \mathbb{N}$ be non-negative functions. We say that $f \in \Omega(g)$ if there is a constant $c \geq 0$ such that for all sufficiently large n, we have that $f(n) \geq c \cdot g(n)$. (Note: It says $\geq$ here, not $\leq$.)* **Ω; big Omega**

An example of a statement involving Ω is about the complexity of sorting functions. It can be shown that any sorting algorithm (that makes black-box comparisons) needs to perform at least $\log n!$ comparisons for sorting a list of n elements.[3] Using big Omega notation, we can just write: any sorting algorithm has a worst case number of comparisons of $\Omega(\log n!) = \Omega(n \log n)$. (The last equality can be shown using the Stirling formula.) We cannot write $O(n \log n)$ here, because there are some sorting algorithms that take much more than $n \log n$ comparisons (e.g., bubble sort).

Exercise 5 *Which of the statements in Exercise 1 are true if we replace O by Ω?*

Exercise 6 *All of the properties listed in Lemma 1 also hold for Ω instead of O with small modifications. What are the modifications that need to be done in Lemma 1 to make it hold for Ω?*

Exercise 7 *Alice tells Bob: I developed a new sorting algorithm with running time $\Omega(\sqrt{n})$. Bob is not impressed. Why?*

4 Asymptotics in crypto

In cryptography, another asymptotic class of functions is of great importance:

Definition 3 (Negligible functions) *A non-negative function μ is called* negligible*, if for any positive polynomial p, it holds that for all sufficiently large η we have $\mu(\eta) < 1/p(\eta)$.* **negligible**

A different way to express this is: A function μ is negligible iff $\mu \in \bigcap_i O(1/\eta^i)$. Or: $\mu \in \bigcap 1/\Omega(\eta^i)$.

What is the meaning of this definition? Typically, the term negligible is applied to probabilities. In crypto, we consider probabilities that are negligible to be too small to matter. In other words, if an attack works only with negligible probability, then we consider that attack to be unsuccessful.

Exercise 8 *Which of the following statements is true? (Justify your answer.)*

[3] The argument is actually very simple: Each comparison gives at most one bit of information. There are $n!$ possible permutations of the list. To identify which of the permutations needs to be applied, we thus need $\log n!$ bits, hence $\log n!$ comparisons.

(a) 2^{-n} *is negligible.*

(b) n^{-100} *is negligible.*

(c) 2^n *is negligible.*

(d) Consider a cryptosystem. On security parameter η*, that scheme uses a block cipher with key length* $\log \eta$*. The adversary can (only) break the cryptosystem by guessing the key of the block cipher. Then the probability that the adversary breaks the cryptosystem is negligible in* η*.*

(e) The same, but with key length η *(instead of* $\log \eta$*).*

(f) Optional (more tricky): The same, but with key length $(\log \eta)^2$*.*

Of course, it is somewhat arbitrary to say that negligible functions are too small to matter, while, e.g., n^{-80} (which is very very small) is considered to matter. The reasons for using this particular class of functions in crypto is that it behaves quite nicely under various operations, this makes things easier in many situations. The following lemma lists some of these nice properties of negligible functions:

Lemma 2 (Some properties of negligible functions)

(i) If f_1 *and* f_2 *are negligible, then* $f_1 + f_2$ *is negligible.*

(ii) If f *is negligible and* $c > 0$ *is a constant, then* $c \cdot f$ *is negligible.*

(iii) If f *is negligible and* p *is polynomially bounded (i.e., there is a polynomial that is bigger than* p *for sufficiently large* n*), then* $p \cdot f$ *is negligible.*

(iv) If f *is negligible, then for any* $c > 0$*,* f^c *is negligible. (This also includes cases like* $f^{1/2} = \sqrt{f}$*.)*

Exercise 9 *An adversary tries to attack a protocol. He can do two things: First, he can try* once *to guess the master key* k*. Second, he can try to guess as often as he wishes to guess a key* k'*. You may assume that* $2^{-|k|}$ *and* $2^{-|k'|}$ *are negligible (i.e., a single try at guessing* k *or* k' *succeeds only with negligible probability). The adversary wins if he guesses* k *(in the first guess) or* k' *(in any of the guesses).*

Show that the probability that the adversary wins is negligible.

(Use Lemma 2; remember also that $\Pr[A_1 \vee \cdots \vee A_n] \leq \Pr[A_1] + \cdots + \Pr[A_n]$*.)*

The next important class of functions in cryptography is that of *overwhelming* functions. **overwhelming**
Intuitively, a probability is overwhelming, if it is almost 1. Formally:

Definition 4 (Overwhelming) *A function* f *is overwhelming if* $1 - f$ *is negligible.*

For example: If we wish to state that a certain message delivery algorithm is secure against denial-of-service attacks, in a formal definition we might require that the probability that the message is delivered is overwhelming. (Which is the same as requiring that the message is lost with negligible probability.)

Exercise 10 *In a good cryptosystem, which of the following is overwhelming? (a) the probability that no attack takes place, (b) the probability that an attack takes place.*

Finally, a third class of functions is often used in crypto, that of noticeable functions.

Definition 5 *A non-negative function f is* noticeable *if there is a positive polynomial p such that for all sufficiently large n, we have $f(n) \geq 1/p(n)$.* **noticeable**

In many situations in crypto, the fact that a probability is noticeable implies that by repetition (trying again until it works), you can make the probability go up almost 1 (overwhelming).

Exercise 11 *Consider the function $f(n) := 1$ for even n and $f(n) := 0$ for odd n. Is f negligible? overwhelming? noticeable?*

A warning: the word *non-negligible* is sometimes used as a synonym for noticeable, and sometimes to mean a function that is not negligible. Be aware of this when you see the word non-negligible! **non-negligible**

References

[Knu76] Donald E. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 8(2):18–24, April 1976.