Algorithms & Data Structures
Part 2 (weeks 6–10)
Set 1: Asymptotics

Konrad Dabrowski
konrad.dabrowski@durham.ac.uk

Online Office Hour:
Mondays 13:30–14:30
See Duo for the Zoom link

Table of Contents

Elementary functions

Should be comfortable with polynomials, exponential functions, logarithmic functions

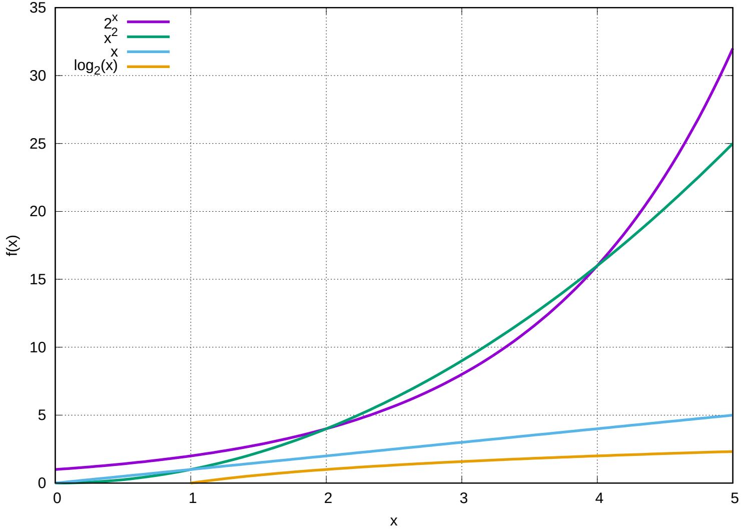
Should know, and be able to derive/prove, e.g.

- $\log_b(xy) = \log_b(x) + \log_b(y)$
- $\log_b(x/y) = \log_b(x) - \log_b(y)$
- $\log_b(x^y) = y \cdot \log_b(x)$
- $\log_b(b^x) = x$, e.g. $\ln(e^4) = 4$
- $b^{a \cdot \log_b x} = x^a$, e.g. $2^{7 \cdot \log_2 x} = x^7$
- $\log_b x = \frac{\log_a x}{\log_a b}$, e.g. $\log_{19} 257 = \frac{\ln 257}{\ln 19}$

Things like this will be on next week's practical assignment

Table of Contents

Growth of functions



Growth of functions

Example: We want to compare the growth rate of $f(x) = x^2$ and $g(x) = 2^x$.

x	$f(x) = x^2$	$g(x) = 2^x$
0	0	**1**
1	1	**2**
2	4	4
3	**9**	8
4	16	16
5	25	**32**
10	100	**1024**
20	400	**1,048,576**

Growth of functions

Example: We want to compare the growth rate of $f(x) = x^{100}$ and $g(x) = 2^x$.

x	$f(x) = x^{100}$	$g(x) = 2^x$
0	0	1
1	1	2
2	2^{100}	4
3	3^{100}	8

But

$$\begin{aligned} \mathrm{g}(1000) &= 2^{1000} = 2^{10\cdot 100} = 1024^{100} \\ &> 1000^{100} = \mathrm{f}(1000) \end{aligned}$$

$$\begin{aligned} \mathrm{g}(10,000) &= 2^{10,000} = 2^{10\cdot 1000} = 1024^{1000} \\ &\gg 1000^{133} \approx 10,000^{100} = \mathrm{f}(10,000) \end{aligned}$$

Time complexity

Definition

The **time complexity** of an algorithm can be expressed in terms of the number of basic operations used by the algorithm when the input has a particular size.

Examples of basic operations are

- additions,
- multiplications,
- comparisons of two numbers,
- swaps,
- assignments of values to variables,. . .

The **space complexity** of an algorithm is expressed in terms of the memory required by the algorithm for an input of a particular size. We will mainly be concerned with time complexity.

Example: evaluation of polynomials

To evaluate the polynomial

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$$

at a fixed value x_0, we can use different approaches yielding different numbers of multiplications and additions.

How many operations of both types do we need in the straightforward way?

Is there a smarter way (in terms of the number of operations)?

Example: evaluation of polynomials

The polynomial

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$$

at a fixed value x_0 can be evaluated as follows:

Polynomial($x_0, a_0, \ldots, a_n$: real numbers)

```
1: $power = 1$
2: $y = a_0$
3: for $i = 1$ to $n$ do
4:     $power = power * x_0$
5:     $y = y + a_i * power$
6: end for
```

$y = a_n x_0^n + a_{n-1} x_0^{n-1} + \ldots + a_2 x_0^2 + a_1 x_0 + a_0$

If we use this procedure, then we will need $2n$ multiplications and n additions to evaluate a polynomial of degree n at $x = x_0$.

You may know Horner scheme; needs n multiplications and n additions.

Example: sorting

The real numbers $a_1, \ldots a_n$, $n \geq 2$, can be **sorted** (i.e. arranged in ascending order) by the **insertion sort** algorithm:

Insertion($a_1, \ldots, a_n$: real numbers with $n \geq 2$)

```
for $j = 2$ to $n$ do
    $x = a_j$
    $i = j - 1$
    while $i > 0$ and $\boxed{a_i > x}$ do
        $a_{i+1} = a_i$
        $i = i - 1$
    end while
    $a_{i+1} = x$
end for
```

Example: sorting

The real numbers $a_1, \ldots a_n$, $n \geq 2$, can be **sorted** (i.e. arranged in ascending order) by the **insertion sort** algorithm:

Insertion($a_1, \ldots, a_n$: real numbers with $n \geq 2$)

```
for $j = 2$ to $n$ do
    $x = a_j$
    $i = j - 1$
    while $i > 0$ and $\boxed{a_i > x}$ do
        $a_{i+1} = a_i$
        $i = i - 1$
    end while
    $a_{i+1} = x$
end for
```

The **worst case** number of comparisons $a_i > x$ is
$1 + 2 + \ldots + (n-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$.
If $a_1 \leq a_2 \leq \ldots \leq a_n$, then the number of comparisons $a_j > x$ is $n - 1$.

Worst-case time complexity

Definition

The **worst-case time complexity** of an algorithm can be expressed in terms of the largest number of basic operations used by the algorithm when the input has a particular size.

Worst-case analysis tells us how many operations an algorithm requires to guarantee that it will produce a solution.

The worst-case time analysis is a standard way to estimate the efficiency of algorithms.
Usually **time complexity** means **worst-case time complexity**.

Another important type of complexity analysis is called **average-case** analysis. Here we are interested in the average number of operations over all inputs of a given size.

Worst-case time complexity

It is difficult to compute the **exact** number of operations.

Usually we don't need it. It is sufficient to **estimate** this number, i.e. give **bounds**.

We are more interested in **upper bounds** for the worst-case analysis.

These bounds should give us the possibility to estimate **growth** of the number of operations when the input size increases.

It is important to estimate the number of operations when the input size is **large**.

Big-O notation

Definition

Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that f is $O(g)$ if there are constants C and k such that

$$|f(x)| \leq C \cdot |g(x)|$$

whenever $x \geq k$.

This is read as "f is big-O of g".

This definition is introduced for general functions. If we consider time complexity functions all functions will have positive values and we do not have to be concerned about the absolute value signs.

Big-O notation

The definition says that after a certain point, namely after k, the absolute value of $f(x)$ is bounded from above by C times the absolute value of $g(x)$.

In terms of time complexities $f(x)$ is no worse than $C \cdot g(x)$ for all relatively large input sizes x.

C is a fixed constant usually depending on the choice of k. We are not allowed to increase C as x increases.

Big-O notation

The constants C and k in the definition of big-O notation are called **witnesses** to the relationship $f(x)$ is $O(g(x))$.

If there is a pair of witnesses to the relationship $f(x)$ is $O(g(x))$, then there are infinitely many pairs of witnesses to that relationship.
Indeed, if C and k are one pair of witnesses, then any pair C' and k', where $C \leq C'$ and $k \leq k'$, is also a pair of witnesses.

To establish that $f(x)$ is $O(g(x))$ we need only one pair of witnesses to this relationship. (We can be "generous", i.e. we do not have to look for the best values of C and k.)

Definition ("Real" definition)

Let $g : \mathbb{N} \to \mathbb{N}$.

$$O(g) = \{f : \mathbb{N} \to \mathbb{N} \mid \exists C, k > 0,\ f(x) \leq C \cdot g(x)\ \forall x \geq k\}$$

- Will often read something like $T(n) = O(n^2)$, with equals sign.

  Is of course horrible abuse of notation, as one is a value and the other a set of functions, but it's absolutely standard ($\ldots T = O(n^2)$ isn't much better.)
- Correct would be $T \in O(n^2)$ but hardly anyone uses that.
- You should **say** "is" ("is big-O of")

  some say "is (in the) order of"
- You should **write** "$=$" (or, more informally, "is (in)")

Example

Let $f(x) = x^2 + 2x + 1$. Then $f(x) = O(x^2)$.

Proof.

For $x \geq 1$, we have $1 \leq x \leq x^2$. That gives

$$f(x) = x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$$

for $x \geq 1$. Because the above inequality holds for every positive $x \geq 1$, using $k = 1$ and $C = 4$ as witnesses, we get

$$f(x) \leq C \cdot x^2,$$

for every $x \geq k$. □

Example

Let $f(x) = 3x^3 - 7x^2 - 4x + 2$. Then $f(x) = O(x^3)$.

Proof.

For $x \geq 1$, we have $1 \leq x \leq x^2 \leq x^3$. That gives

$$\begin{aligned} |f(x)| = |3x^3 - 7x^2 - 4x + 2| &\leq 3x^3 + 7x^2 + 4x + 2 \\ &\leq 3x^3 + 7x^3 + 4x^3 + 2x^3 = 16x^3 \end{aligned}$$

for $x \geq 1$. Because the above inequality holds for every positive $x \geq 1$, using $k = 1$ and $C = 16$ as witnesses, we get

$$|f(x)| \leq C \cdot |x^3|,$$

for every $x \geq k$. □

Example

Let $f(x) = 3^x$. Then $f(x)$ **is not** $O(2^x)$.

Proof.

Assume that there are constants k and C such that $3^x \leq C \cdot 2^x$ when $x \geq k$. Then

$$\left(\frac{3}{2}\right)^x \leq C,$$

when $x \geq k$.

But any exponential function a^x grows arbitrarily large whenever $a > 1$; a contradiction. □

Example

The polynomial

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

at a fixed value x_0 can be evaluated as follows:

Polynomial($x_0, a_0, \ldots, a_n$: real numbers)

```
1: $power = 1$
2: $y = a_0$
3: **for** $i = 1$ to $n$ **do**
4:     $power = power * x_0$
5:     $y = y + a_i * power$
6: **end for**
```

$$y = a_n x_0^n + a_{n-1} x_0^{n-1} + \ldots + a_2 x_0^2 + a_1 x_0 + a_0$$

The time complexity of the procedure is $O(n)$.

Example

The real numbers $a_1, \ldots a_n$, $n \geq 2$, can be **sorted** (i.e. arranged in ascending order) by the **insertion sort** algorithm:

Insertion($a_1, \ldots, a_n$:real numbers with $n \geq 2$)

```
for $j = 2$ to $n$ do
    $x = a_j$
    $i = j - 1$
    while $i > 0$ and $a_i > x$ do
        $a_{i+1} = a_i$
        $i = i - 1$
    end while
    $a_{i+1} = x$
end for
```

The time complexity of the procedure is $O(n^2)$.

Big-O notation

Some orders that often occur in practice are:

Big-O form	Name
$O(1)$	constant
$O(\log n)$	logarithmic
$O(n)$	linear
$O(n \log n)$	$n \log n$
$O(n^2)$	quadratic
$O(n^3)$	cubic
$O(n^k)$ for $k = 0, 1, 2, \ldots$	polynomial
$O(c^n)$ for $c > 1$	exponential

Big-O notation

Example

Let $1 < a < b$. Then $a^x = O(b^x)$ but b^x **is not** $O(a^x)$.

Proof.

For any $x \geq 0$, $a^x \leq b^x$. Hence, we can take $C = 1$ and $k = 0$.

Assume that there are constants k and C such that $b^x \leq C \cdot a^x$ when $x \geq k$. Then

$$\left(\frac{b}{a}\right)^x \leq C,$$

when $x \geq k$.

Observe that $c = \frac{b}{a} > 1$. Any exponential function c^x grows arbitrarily large whenever $c > 1$; a contradiction. □

Big-O notation

Example

Let $a, b > 1$. Then $\log_a x = O(\log_b x)$.

Proof.

We know that $\log_a x = \frac{\log_b x}{\log_b a}$. Hence, we can take $C = \frac{1}{\log_b a}$ and any $k > 0$. □

Big-O notation

Example

Let $0 < p < q$. Then $x^p = O(x^q)$ but x^q **is not** $O(x^p)$.

Proof.

For any $x \geq 1$, $x^p \leq x^q$. Hence, we can take $C = 1$ and $k = 1$.

Assume that there are constants k and C such that $x^q \leq C \cdot x^p$ when $x \geq k$. Then

$$x^{q-p} \leq C.$$

Observe that $r = q - p > 0$. Any function x^r grows arbitrarily large whenever $r > 0$; a contradiction. $\square$

Big-O notation

Example

Let $a > 1$ and let $0 < p$. Then $x^p = O(a^x)$ but a^x **is not** $O(x^p)$.

Example

Let $a > 1$ and let $0 < p$. Then $\log_a x = O(x^p)$ but x^p **is not** $O(\log_a x)$.

Sum and product rules

Theorem (The sum rule)

If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $f_1(x) + f_2(x)$ is $O(\max\{|g_1(x)|, |g_2(x)|\})$.

Proof.

Let C_i and k_i be witness pairs for $f_i(x)$ is $O(g_i(x))$, for $i = 1, 2$. Let $k = \max\{k_1, k_2\}$ and $C = C_1 + C_2$. Then for $x > k$ we have $|f_1(x) + f_2(x)| \leq |f_1(x)| + |f_2(x)| \leq C_1 \cdot |g_1(x)| + C_2 \cdot |g_2(x)| \leq C \cdot \max\{|g_1(x)|, |g_2(x)|\}$.

The last inequality is true because $C_1 y_1 + C_2 y_2 \leq C_1 \max\{y_1, y_2\} + C_2 \max\{y_1, y_2\} = (C_1 + C_2) \max\{y_1, y_2\} = C \max\{y_1, y_2\}$. $\square$

Corollary: If $f_1(x)$ is $O(g(x))$ and $f_2(x)$ is $O(g(x))$, then $f_1(x) + f_2(x)$ is also $O(g(x))$.

Sum and product rules

Theorem (The product rule)

If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $f_1(x) \cdot f_2(x)$ is $O(g_1(x) \cdot g_2(x))$.

Proof.

Let C_i and k_i be witness pairs for $f_i(x)$ is $O(g_i(x))$, for $i = 1, 2$.
Let $k = \max\{k_1, k_2\}$ and $C = C_1 \cdot C_2$. Then for $x > k$ we have
$|f_1(x) \cdot f_2(x)| = |f_1(x)| \cdot |f_2(x)| \leq C_1 \cdot |g_1(x)| \cdot C_2 \cdot |g_2(x)| = C \cdot |g_1(x) \cdot g_2(x)|$.
This proves the theorem. □

Example

Proposition

Let $a_0, a_1, \ldots, a_n$ *be real numbers,*
$f(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_1 \cdot x + a_0$*. Then*

$$f(x) = O(x^n).$$

Proof.

For each $0 \leq i \leq n$, $x^i = O(x^n)$.
Then we observe that for any constant a, $a \cdot x^i = O(x^n)$.
By the sum rule $f(x) = O(x^n)$. $\square$

Lower bounds: Big-Omega notation

The Big-O notation is very useful to find reasonable upper bounds for growth rates, but does not really help much if we are interested in the best function that **matches the growth rate**.
As a first step in this direction, we introduce a similar definition for lower bounds which is called **Big-Omega notation**.

Definition

Let $f(x)$ and $g(x)$ be functions from the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Omega(g(x))$ if there are positive constants C and k such that

$$|f(x)| \geq C \cdot |g(x)|$$

whenever $x \geq k$. Note that this implies that $f(x)$ is $\Omega(g(x))$ if and only if $g(x)$ is $O(f(x))$.

Same order growth rates

Definition

Let $f(x)$ and $g(x)$ be functions from the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Theta(g(x))$ if $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$.

This is equivalent to saying that $f(x)$ is $\Theta(g(x))$ if $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$.

And this is equivalent to saying that there are constants C_1, C_2 and k such that $|f(x)| \leq C_1 \cdot |g(x)|$ and $|g(x)| \leq C_2 \cdot |f(x)|$ whenever $x \geq k$.

Examples: All polynomials $a_n x^n + \ldots + a_1 x + a_0$ with $a_n \neq 0$ are $\Theta(x^n)$.

Little-o notation

We would like to have a tool for disregarding or neglecting "small order" terms.

Little-o notation gives us such a tool.

It is based on the concept of limits, so working with this notation requires some background in Calculus.

Definition

Let $f(x)$ and $g(x)$ be functions from the set of real numbers to the set of real numbers. We say that $f(x)$ is $o(g(x))$ ("little-o" of $g(x)$) when

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0.$$

Without going into details and definitions related to limits this says that the fraction $\frac{f(x)}{g(x)}$ gets arbitrarily close to 0 when x gets larger.

Without limit:

Definition

$$o(g) = \{f : \mathbb{N} \to \mathbb{N} \mid \forall C > 0 \ \exists k > 0 : \ f(n) < C \cdot g(n) \ \forall n \geq k\}$$

Little-o notation

This clearly shows that $f(x)$ is $o(g(x))$ implies $f(x)$ is $O(g(x))$.

If we suppose $f(x)$ **is not** $O(g(x))$, then for all positive constants C and k, there exists a value of $x > k$ such that $|f(x)| > C \cdot |g(x)|$, and then clearly either $\lim_{x\to\infty} \frac{f(x)}{g(x)}$ does not exist or it is not 0. Then $f(x)$ is **not** $o(g(x))$.

Special case: sublinear functions

Intuitively, a function is called **sublinear** if it grows slower than any linear function. With **Little-o notation** we can make this very precise.

Definition

A function $f(x)$ is called **sublinear** if $f(x)$ is $o(x)$, so if $\lim_{x\to\infty} \frac{f(x)}{x} = 0$.

Examples: The function $f(x) = 100/\log(x)$ is sublinear, since

$$\lim_{x\to\infty} \frac{f(x)}{x} = \lim_{x\to\infty} \frac{100}{x \cdot \log x} = 0$$

but $f(x) = \frac{1}{2}x$ is not because

$$\lim_{x\to\infty} \frac{f(x)}{x} = \lim_{x\to\infty} \frac{\frac{1}{2}x}{x} = \frac{1}{2}$$

The function $f(x) = \sqrt[3]{x^2}$ is sublinear, since

$$\lim_{x\to\infty} \frac{f(x)}{x} = \lim_{x\to\infty} \frac{x^{\frac{2}{3}}}{x} = \lim_{x\to\infty} x^{-\frac{1}{3}} = 0.$$

Little-omega

ω is to o what Ω is to O:

$$f = \omega(g) \quad \Leftrightarrow \quad g = o(f)$$

Or:

Definition

$$\omega(g) = \{f : \mathbb{N} \to \mathbb{N} \mid \forall C > 0 \; \exists k > 0 : \; f(n) > C \cdot g(n) \; \forall n \geq k\}$$

General rules

The following results show how to apply asymptotic notation more generally. You can use the rules without proving them.

Theorem

If $f_1(x)$ is $o(g(x))$ and $f_2(x)$ is $o(g(x))$, then $f_1(x) + f_2(x)$ is $o(g(x))$.

Theorem

If $f_1(x)$ is $O(g(x))$ and $f_2(x)$ is $o(g(x))$, then $f_1(x) + f_2(x)$ is $O(g(x))$.

Theorem

If $f_1(x)$ is $\Theta(g(x))$ and $f_2(x)$ is $o(g(x))$, then $f_1(x) + f_2(x)$ is $\Theta(g(x))$.

Summary: for $g : \mathbb{N} \to \mathbb{N}$,

"$\leq$": $\mathcal{O}(g) = \{f : \mathbb{N} \to \mathbb{N} \mid \exists C, k > 0 : \; f(n) \leq C \cdot g(n) \; \forall n \geq k\}$

"$\geq$": $\Omega(g) = \{f : \mathbb{N} \to \mathbb{N} \mid \exists C, k > 0 : \; f(n) \geq C \cdot g(n) \; \forall n \geq k\}$

"$=$": $\Theta(g) = O(g) \cap \Omega(g)$

"$<$": $o(g) = \{f : \mathbb{N} \to \mathbb{N} \mid \forall C > 0 \; \exists k > 0 : \; f(n) < C \cdot g(n) \; \forall n \geq k\}$

"$>$": $\omega(g) = \{f : \mathbb{N} \to \mathbb{N} \mid \forall C > 0 \; \exists k > 0 : \; f(n) > C \cdot g(n) \; \forall n \geq k\}$

Table of Contents

Table of Contents