Asymptotics

1 Growth of functions

The more logs, the slower it grows
Large constants before the terms will cause the crossover point to be pushed far into larger digits

1.1 Examples

1.1.1 Example 1

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	2	25
5	25	32
10	100	$1048,576$

1.1.2 Example 2

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:

$$\begin{aligned} g(1000) = 2^{1000} = 2^{10 \cdot 100} &= 1024^{100} \\ &> 1000^{100} = \mathbf{f}(1000) \\ \mathbf{g}(10,000) = 2^{10,000} = 2^{10.1000} &= 1024^{1000} \\ &\gg 1000^{133} \approx 10,000^{100} = \mathbf{f}(10,000) \end{aligned}$$

- Subbing in values will allow you to find the crossover point

2 Time complexity

2.1 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 (tends to be the slowest of the basic operations)
- 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. Will mainly be concerned with time complexity

2.2 Example - Evaluation of polynomials

To evaluate 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,, 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?

- n operations 1st term, n-1 2nd term etc etc
- Roughly n^2 operations to perform

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

It can be evaluated as follows

```
Polynomial(x_0, a_0, ..., a_n: real numbers)
power = 1
y=a_0
for i=1 to n do
    power=power×x_0
    y = y + a_i× power
end for
```

- This builds the power up as you go along so vastly reduces re calculating

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

However there is an alternate method

```
Horner(a_0, a_0, ..., a_n:real numbers)
y = a_n
for i=n-1 down to 0 do
    y = y × x_0 + a_i
end for
```

If we use this procedure, then we will need n multiplications (1 multiplication in loop, loop n times) and n additions to evaluate a polynomial of degree n at $x = x_0$

2.3 Example - Sorting

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

```
Insertion(a_1, ..., a_n: real numbers with n ⩾ 2)
for j=2 to n do
    x = x_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 **worst case** for number of comparisons $a_i > x$ is:

$$1 + 2 + \ldots + (n-1) = \frac{n(n-1)}{2}$$

If $a_1 \leqslant a_2 \leqslant ... \leqslant a_n$ then the number of comparisons $a_j > x$ is n-1

2.4 Worst case time complexity

2.4.1 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

2.4.2 More on worst case time complexity

- 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
- 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 then the input size is **large**

3 Big O

3.1 Definition

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

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

whenever $x \geqslant k$

3.2 More on Big O

- The 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.
- The definition says that after a certain point, namely after k, the absolute value of $f(x)$ is bounded 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
- The constants C and k in the definition of big-O are called the **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 $C \leqslant C'$ and $k \leqslant k'$, 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)

3.3 Examples

3.3.1 Example 1

- Here we are replacing all x terms with x^2, then the numbers will come out easily.

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

For $x \geqslant 1$ we have $1 \leqslant x \leqslant x^2$. That gives

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

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

$$f(x) \leqslant C \cdot x^2$$

for every $x \geqslant k$

3.3.2 Example 2

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

For $x \geqslant 1$, we have $1 \leqslant x \leqslant x^2 \leqslant x^3$. That gives

$$|f(x)| = \left|3x^3 - 7x^2 - 4x + 2\right| \leq 3x^3 + 7x^2 + 4x + 2 \quad \leq 3x^3 + 7x^3 + 4x^3 + 2x^3 = 16x^3$$

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

$$|f(x)| \leqslant C \cdot |x^3|$$

for every $x \geqslant k$

3.3.3 Example 3

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

Assume that there are constants k and C such that $3^x \leqslant C \cdot 2^x$ when $x \geqslant k$ Then

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

when $x \geqslant k$

But any exponential function a^x grows monotonically whenever $a \geqslant 1$; a contradiction

3.3.4 Example 4

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

```
Insertion(a_1,...,a_n: real numbers with n ⩾ 2)
for j=2 to n do
x = x_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
```

$$y = a_n x_0^n + a_{n-1} x_0^{n-1} + ... + a_1 x_0 + a_0$$

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

3.3.5 Example 5

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

```
Insertion(a_1,...,a_n: real numbers with n ⩾ 2)
for j=2 to n do
x = x_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 $O(n^2)$

3.4 Summary

Big-O form	Name
$O(1)$	constant
$O(logn)$	logarithmic
$O(n)$	linear
$O(n \log n)$	n log n
$O(n^2)$	quadratic
$O(n^3)$	cubic
$O(n^k)$	polynomial
$O(c^n)$	exponential

3.5 More examples

3.5.1 Example 1

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

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

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

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

when $x \geqslant k$

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

3.5.2 Example 2

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

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 $k > 0$

$$\log_a x = \frac{1}{\log_b a} \cdot \log_b x$$

$$f(x) = \frac{1}{C} \cdot g(x)$$

3.5.3 Example 3

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

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

Assume that there are constants k and C such that $x^q \leqslant C \cdot x^p$ when $x \geqslant k$ Then

$$x^{q-p} \leqslant C$$

Observe that $r = q - p > 0$ Any function x^r grows monotonically whenever $r > 0$; a contradiction

3.5.4 Example 4

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

$$\lim_{x \to \infty} \frac{x^p}{a^x} = 0$$

3.5.5 Example 5

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

$$\lim_{x \to \infty} \frac{\log_a x}{x^p} = 0$$

3.6 Sum and Product rules

3.6.1 The sum rule

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

Two functions $f_1(x)$ and $f_2(x)$ for each we know what their O are. Adding these together, the big o of the sum of them is the maximum of their Os

3.6.1.1 Example

$$f_1(x) = 2x^2 + 1$$
$$f_2(x) = 4x^3 + x^2 + 2$$
$$g_1(x) = x^2$$
$$g_2(x) = x^3$$
$$f_1(x) + f_2(x) = 4x^3 + 3x^2 + 3$$

3.6.1.2 Proof - don't need to know

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
$\left|f_1(x) + f_2(x)\right| \leq \left|f_1(x)\right| + \left|f_2(x)\right| \leq C_1 \cdot \left|g_1(x)\right| + C_2 \cdot \left|g_2(x)\right| \leq C \cdot \max\left\{\left|g_1(x)\right|, \left|g_2(x)\right|\right\}$
The last inequality is true because
$C_1 y_1 + C_2 y_2 \leq C_1 \max\{y_1, y_2\} + C_2 \max\{y_1, y_2\} = (C_1 + C_2) \max\{y_1, y_2\} = C \max\{y_1, y_2\}$

3.6.2 The product rule

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

When multiplying two functions f_1 and f_2, then the O of the product is the product of the O of the two functions.

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
$\left|f_1(x) \cdot f_2(x)\right| = \left|f_1(x)\right| \cdot \left|f_2(x)\right| \leq C_1 \cdot \left|g_1(x)\right| \cdot C_2 \cdot \left|g_2(x)\right| = C \cdot \left|g_1(x) \cdot g_2(x)\right|$

3.6.3 Example

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

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

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

4 Big Omega

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

4.1 Defintion

Let $f(x)$ and $g(x)$ be functions from the set of real number 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

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

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

5 Theta

5.1 Definition

This provides a tight bound on the time complexity of something

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_1 and k such that $|f(x)| \leqslant C_1 \cdot |g(x)|$ and $|g(x)| \leqslant C_2 \cdot |f(x)|$ whenever $x \geqslant k$

5.2 Proving something is Θ

$$3x^2 + 2x + 1$$

$$1 \leqslant x \leqslant x^2 \text{ where } x \geqslant 1$$

$$3x^2 + 2x + 1 \leqslant 3x^2 + 2x^2 + x^2 = 6x^2$$

$$3x^2 \leqslant 3x^2 + 2x + 1 \leqslant 6x^2$$

$$k = 1, C_1 = 3, C_2 = 6, g(x) = x^2$$

6 Little-o notation

We would like to have a tool for disregarding or neglecting ”smaller order” terms. Little o notation gives us such a tool

It is based on the concept of limits

This is something that grows strictly faster, whereas big O would allow something at the same rate

6.1 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 $o(g(x))$ when:

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

Without the limit, this can be shown as:

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

This is actually saying

$C \cdot f(n) < g(n)$ for all values of C greater than 0

Solving is best done using the limit formula

The non limit formula is best for proving that something isn't little o

6.2 More on 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 $o(g(x))$

6.3 Special Case - Sublinear Functions

A function is called **sublinear** if it grows slower than a linear function. With little o notation we can make this very precise.

6.3.1 Definition

A function $f(x)$ is called **sublinear** if $f(x)$ is $o(x)$, so if

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

6.3.2 Examples

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

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

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

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

7 Little omega

ω is to o what Ω is to O

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

7.1 Definition

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

8 General Rules

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

8.1 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 $o(g(x))$.

8.2 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 $O(g(x))$.

8.3 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 $\Theta(g(x))$.

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

Equivalent to $\leqslant$

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

Equivalent to $\geqslant$

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

Equivalent to $=$

$$\Theta(g) = \{f : \mathbb{N} \to \mathbb{N} | \exists C_1, C_2, k > 0 : \quad \mathbf{C_1} \cdot \mathbf{g(n)} \leq \mathbf{f(n)} \leq \mathbf{C_2} \cdot \mathbf{g(n)} \forall n \geq k\}$$

Equivalent to $<$

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

Equivalent to $>$

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