

## Section 1.8 The Growth of Functions

---

---

We quantify the concept that  $g$  *grows at least as fast as*  $f$ .

What really matters in comparing the complexity of algorithms?

- We only care about the behavior for *large* problems.
  - Even bad algorithms can be used to solve small problems.
  - Ignore implementation details such as loop counter incrementation, etc. We can straight-line any loop.
- 

### The Big-O Notation

**Definition:** Let  $f$  and  $g$  be functions from  $\mathbb{N}$  to  $\mathbb{R}$ . Then  $g$  *asymptotically dominates*  $f$ , denoted  $f$  is  $O(g)$  or ' $f$  is big-O of  $g$ ,' or ' $f$  is order  $g$ ,' iff

$$\exists k \in \mathbb{N} \exists C > 0 \forall n > k \quad |f(n)| \leq C |g(n)|$$

Note:

- Choose  $k$
- Choose  $C$ ; it may depend on your choice of  $k$
- Once you choose  $k$  and  $C$ , you must prove the truth of the implication (often by induction)

---

An alternative for those with a calculus background:

**Definition:** if  $\lim_n \frac{f(n)}{g(n)} = 0$  then  $f$  is  $o(g)$  (called *little-o* of  $g$ )

---

**Theorem:** If  $f$  is  $o(g)$  then  $f$  is  $O(g)$ .

Proof: by definition of limit as  $n$  goes to infinity,  $f(n)/g(n)$  gets arbitrarily small.

That is for any  $\epsilon > 0$ , there must be an integer  $N$  such that when  $n > N$ ,  $|f(n)/g(n)| < \epsilon$ .

Hence, choose  $C = 1/\epsilon$  and  $k = N$ .

Q. E. D.

---

It is usually easier to prove  $f$  is  $o(g)$

- using the theory of limits
- using L'Hospital's rule
- using the properties of logarithms

etc.

Example:

$$3n + 5 \text{ is } O(n^2)$$

Proof: It's easy to show  $\lim_n \frac{3n + 5}{n^2} = 0$  using the theory of limits.

Hence  $3n + 5$  is  $o(n^2)$  and so it is  $O(n^2)$ .

Q. E. D.

We will use induction later to prove the result from scratch.

---

Also note that  $O(g)$  is a set called a

*complexity class.*

It contains all the functions which  $g$  dominates.

$f$  is  $O(g)$  means  $f \in O(g)$ .

---

## Properties of Big-O

- $f$  is  $O(g)$  iff  $O(f) \subseteq O(g)$
- If  $f$  is  $O(g)$  and  $g$  is  $O(f)$  then  $O(f) = O(g)$

- The set  $O(g)$  is closed under addition:

If  $f$  is  $O(g)$  and  $h$  is  $O(g)$  then  $f + h$  is  $O(g)$

- The set  $O(g)$  is closed under multiplication by a scalar  $a$  (real number):

If  $f$  is  $O(g)$  then  $af$  is  $O(g)$

that is,

$O(g)$  is a *vector space*.

(The proof is in the book).

Also, as you would expect,

- if  $f$  is  $O(g)$  and  $g$  is  $O(h)$ , then  $f$  is  $O(h)$ .

In particular

$$\frac{O(f) \quad O(g) \quad O(h)}{\quad \quad \quad}$$

**Theorem:** If  $f_1$  is  $O(g_1)$  and  $f_2$  is  $O(g_2)$  then

i)  $f_1 f_2$  is  $O(g_1 g_2)$

ii)  $f_1 + f_2$  is  $O(\max\{g_1, g_2\})$

Proof of ii): There is a  $k_1$  and  $C_1$  such that

$$1. f_1(n) < C_1 g_1(n)$$

when  $n > k_1$ .

There is a  $k_2$  and  $C_2$  such that

$$2. f_2(n) < C_2 g_2(n)$$

when  $n > k_2$ .

We must find a  $k_3$  and  $C_3$  such that

$$3. f_1(n)f_2(n) < C_3 g_1(n)g_2(n)$$

when  $n > k_3$ .

We use the inequality

$$\text{if } 0 < a < b \text{ and } 0 < c < d \text{ then } ac < bd$$

to conclude that

$$f_1(n)f_2(n) < C_1 C_2 g_1(n)g_2(n)$$

as long as  $k > \max\{k_1, k_2\}$  so that both inequalities 1 and 2. hold at the same time.

Therefore, choose

$$C_3 = C_1 C_2$$

and

$$k_3 = \max\{k_1, k_2\}.$$

Q. E. D.

---

### Important Complexity Classes

$$\begin{array}{cccccc} O(1) & O(\log n) & O(n) & O(n \log n) & O(n^2) & \\ & O(n^j) & O(c^n) & O(n!) & & \end{array}$$

where  $j > 2$  and  $c > 1$ .

---

Example:

Find the complexity class of the function

$$(nn! + 3^{n+2} + 3n^{100})(n^n + n2^n)$$

Solution:

This means to simplify the expression.

Throw out stuff which you know doesn't grow as fast.

We are using the property that if  $f$  is  $O(g)$  then  $f + g$  is  $O(g)$ .

- Eliminate the  $3n^{100}$  term since  $n!$  grows much faster.

- Eliminate the  $3^{n+2}$  term since it also doesn't grow as fast as the  $n!$  term.

Now simplify the second term:

Which grows faster, the  $n^n$  or the  $n2^n$ ?

- Take the log (base 2) of both.

Since the log is an increasing function whatever conclusion we draw about the logs will also apply to the original functions (why?).

- Compare  $n \log n$  or  $\log n + n$ .

- $n \log n$  grows faster so we keep the  $n^n$  term

The complexity class is

$$O(n n! n^n)$$

---

If a flop takes a nanosecond, how big can a problem be solved (the value of  $n$ ) in

a minute?

a day?

a year?

for the complexity class  $O(n n! n^n)$ .

Note: We often want to compare algorithms in the same complexity class

---

Example:

Suppose

Algorithm 1 has complexity  $n^2 - n + 1$

Algorithm 2 has complexity  $n^2/2 + 3n + 2$

Then both are  $O(n^2)$  but Algorithm 2 has a smaller leading coefficient and will be faster for large problems.

Hence we write

Algorithm 1 has complexity  $n^2 + O(n)$

Algorithm 2 has complexity  $n^2/2 + O(n)$

---