

MA3105

# Numerical Analysis

Autumn 2021

Satvik Saha  
19MS154

*Indian Institute of Science Education and Research, Kolkata,  
Mohanpur, West Bengal, 741246, India.*

## Contents

<b>1</b>	<b>Time complexity</b>	<b>1</b>
1.1	Runtime cost . . . . .	1
1.2	Asymptotic growth . . . . .	2

## 1 Time complexity

### 1.1 Runtime cost

When designing or implementing an algorithm, we care about its efficiency – both in terms of execution time, and the use of resources. This gives us a rough way of comparing two algorithms. However, such metrics are architecture and language dependent; different machines, or the same program implemented in different programming languages, may consume different amounts of time or resources while executing the same algorithm. Thus, we seek a way of measuring the ‘cost’ in time for a given algorithm.

For example, we may look at each statement in a program, and associate a cost  $c_i$  with each of them. Consider the following statements.

```
one = 1;           // c_1
two = 2;           // c_2
three = 3;         // c_3
```

The total cost of running these statements can be calculated as  $T = c_1 + c_2 + c_3$ , simply by adding up the cost of each statement. Similarly, consider the following loop construct.

```
sum = 0;           // c_1
for (i = 0; i < n; i++) // c_2
    sum += a[i];    // c_3
```

The total cost can be shown to be  $T(n) = c_1 + c_2(n + 1) + c_3n$ ; this time, we must take into account the number of times a given statement is executed. Note that this is linear. Another example is as follows.

```

sum = 0;                                // c_1
for (i = 0; i < n; i++)                  // c_2
    for (j = 0; j < n; j++)              // c_2
        sum += a[i][j];                  // c_4

```

The total cost can be shown to be  $T(n) = c_1 + c_2(n+1) + c_3n(n+1) + c_4n^2$ . Note that this is quadratic. Finally, consider the following recursive call.

```

int factorial (int n) {                  // c_1
    if (n == 0)                          // c_2
        return 1;                       // c_3
    return n * factorial(n - 1);          // c_4
}

f = factorial(n);                        // c_5

```

The cost can be shown to be  $T(n) = c_5 + (c_1 + c_2)(n+1) + c_3 + c_4n$ . This turns out to be linear.

In all these cases, we care about our total cost as a function of the input size  $n$ . Moreover, we are interested mostly in the *growth* of our total cost; as our input size grows, the total cost can often be compared with some simple function of  $n$ . Thus, we can classify our cost functions in terms of their asymptotic growths.

## 1.2 Asymptotic growth

**Definition 1.1.** The set  $O(g(n))$  denotes the class of functions  $f$  which are asymptotically bounded above by  $g$ . In other words,  $f(n) \in O(g(n))$  if there exists  $M > 0$  and  $n_0 \in \mathbb{N}$  such that for all  $n \geq n_0$ ,

$$|f(n)| \leq Mg(n).$$

This amounts to writing

$$\limsup_{n \rightarrow \infty} \frac{|f(n)|}{g(n)} < \infty.$$

*Example.* Consider a function defined by  $f(n) = an + b$ , where  $a > 0$ . Then, we can write  $f(n) \in O(n)$ . To see why, note that for all  $n \geq 1$ , we have

$$|f(n)| = |an + b| \leq an + |b| \leq (a + |b|)n.$$

Thus, setting  $M = a + |b| > 0$  completes the proof.

*Example.* Consider a polynomial function defined by

$$f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0,$$

with some non-zero coefficient. Then, we can write  $f(n) \in O(n^k)$ . Like before, note that for all  $n \geq 1$ , we have

$$|f(n)| \leq \sum_{i=0}^k |a_i| n^i \leq \sum_{i=0}^k |a_i| n^k = (|a_k| + |a_{k-1}| + \cdots + |a_0|) n^k.$$

Thus, setting  $M = |a_k| + \cdots + |a_0| > 0$  completes the proof.

**Theorem 1.1.** If  $f_1(n) \in O(g_1(n))$  and  $f_2(n) \in O(g_2(n))$ , then

$$f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\}).$$

**Definition 1.2.** The set  $\Omega(g(n))$  denotes the class of functions  $f$  are asymptotically bounded below by  $g$ . In other words,  $f(n) \in \Omega(g(n))$  if there exists  $M > 0$  and  $n_0 \in \mathbb{N}$  such that for all  $n \geq n_0$ ,

$$|f(n)| \geq M g(n).$$

This amounts to writing

$$\liminf_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0.$$

**Definition 1.3.** The set  $\Theta(g(n))$  denotes the class of functions  $f$  which are asymptotically bounded both above and below by  $g$ . In other words,  $f(n) \in \Theta(g(n))$  if there exist  $M_1, M_2 > 0$  and  $n_0 \in \mathbb{N}$  such that for all  $n \geq n_0$ ,

$$M_1 g(n) \leq |f(n)| \leq M_2 g(n).$$

This amounts to writing  $f(n) \in O(g(n))$  and  $f(n) \in \Omega(g(n))$ .

Another class of notation uses the idea of dominated growth.

**Definition 1.4.** The set  $o(g(n))$  denotes the class of functions  $f$  which are asymptotically dominated by  $g$ . In other words,  $f(n) \in o(g(n))$  if for all  $M > 0$ , there exists  $n_0 \in \mathbb{N}$  such that for all  $n \geq n_0$ ,

$$|f(n)| < M g(n).$$

This amounts to writing

$$\lim_{n \rightarrow \infty} \frac{|f(n)|}{g(n)} = 0.$$

**Definition 1.5.** The set  $\omega(g(n))$  denotes the class of functions  $f$  which asymptotically dominate  $g$ . In other words,  $f(n) \in \omega(g(n))$  if for all  $M > 0$ , there exists  $n_0 \in \mathbb{N}$  such that for all  $n \geq n_0$ ,

$$|f(n)| > Mg(n).$$

This amounts to writing

$$\lim_{n \rightarrow \infty} \frac{|f(n)|}{g(n)} = \infty.$$

**Definition 1.6.** We say that  $f(n) \sim g(n)$  if  $f$  is asymptotically equal to  $g$ . In other words,  $f(n) \sim g(n)$  if for all  $\epsilon > 0$ , there exists  $n_0 \in \mathbb{N}$  such that for all  $n \geq n_0$ ,

$$\left| \frac{f(n)}{g(n)} - 1 \right| < \epsilon.$$

This amounts to writing

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1.$$

We often abuse notation and treat the following as equivalent.

$$T(n) \in O(g(n)), \quad T(n) = O(g(n)).$$