Asymptotics

Andreas Klappenecker

§1 Introduction

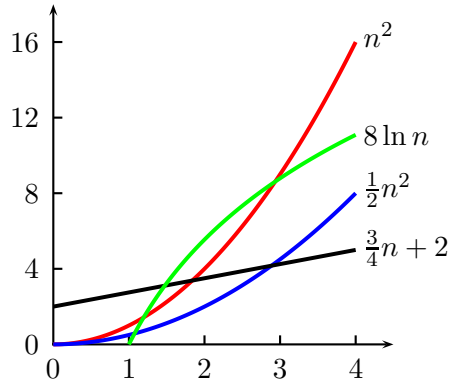
If we have the choice between numerous algorithms for a given problem, then we usually take ease of implementation, elegance, and estimates for the time and space consumption into account. Typically, we would like to get an estimate on how well our system can perform if the input reaches realistic sizes.

Counting the number of operations in an execution often provides too much detail, so we would like to get an easy estimate that tells us about the behaviour of the algorithm for larger input sizes. The Big Oh notation by Paul Bachmann and the little-oh notation by Edmund Landau are often used in expressing upper bounds on the running time while suppressing a certain level of detail.

Let us have a look at the definition of the Big Oh notation. Suppose that f and g are functions from the positive integers $\mathbf{Z}_+$ into the real or complex numbers. We say that f is Big Oh of g, and write $f \in O(g)$, if there exists a positive integer n_0 and a real constant C such that $|f(n)| \leq C|g(n)|$ for all $n \geq n_0$. Thus, $O(g)$ denotes the set of functions

$$O(g) = \left\{ f\colon \mathbf{Z}_+ \to \mathbf{C} \,\middle|\, \begin{array}{l} \text{there exists a positive integer } n_0 \text{ and a constant } C \\ \text{such that } |f(n)| \leq C|g(n)| \text{ for all integers } n \geq n_0 \end{array} \right\}.$$

For example, the set $O(n^2)$ contains the functions that are shown in the next figure:

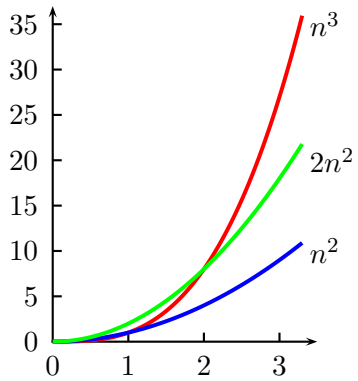


Let us check one of the examples that are given in the previous graph.

Example 1. The graph suggests that $3/(4n)+2 \in O(n^2)$ holds and we want to prove that this is indeed the case. It suffices to show that $|\frac{3}{4}n+2| \leq 3|n^2|$ holds for all integers $n \geq 1$. If we divide both sides of this inequality by n^2, then the resulting function on the left-hand side is $g(n) = \frac{3}{4}n^{-1}+2n^{-2}$. The function $g(n)$ is monotonically decreasing and we have $g(1) \leq 3$. Therefore, we can conclude that $g(n) \leq 3$ holds for all integers $n \geq 1$, and this fact implies our claim.

Remark 1. A more restrictive definition of the Big Oh notation is often used in the Computer Science literature. For instance, Knuth defines $O(f)$ in the Art of Computer Programming to be the set of all functions $g(n)$ such that there exists a constant C and an integer n_0 such that $|g(n)| \leq Cf(n)$ for all $n \geq n_0$. In this restrictive definition, the values $f(n)$ must be nonnegative real numbers for almost all n, otherwise $O(f)$ will be the empty set. In the analysis of number-theoretic algorithms it is sometimes helpful to allow functions with complex values.

If we compare the asymptotic growth of the two functions, then $f(n) \in O(g(n))$ means intuitively that $f(n)$ does not grow faster than $g(n)$. For instance, $n^3 \notin O(n^2)$, because the function n^3 grows more rapidly than any of the functions Cn^2 for all $C > 0$.



Exercise 1. Use the definition of the Big Oh notation to prove that $n^3 \notin O(n^2)$ holds.

Exercise 2. Show that $f(n) \in O(f(n))$.

Notation. The Big Oh notation is used in the literature in a somewhat idiosyncratic way. By tradition, the expression $f(n) \in O(g(n))$ is usually written in the form $f(n) = O(g(n))$. For example, $n^2 \ln n = O(n^3)$ means $n^2 \in O(n^3)$. We also find expressions such as $n + O(n^2) = O(n^3)$, and this should be interpreted as $n + O(n^2) \subseteq O(n^3)$.

There is basically nothing wrong with the equality signs, as long as you know how to interpret them. You should be aware of the fact that the sign "=" is **not** symmetric in expressions involving Big Oh notations. For instance, it does not make sense to write $n + O(n^2) = n^2$, since the set of functions $n + O(n^2)$ is not a subset of $\{n^2\}$.

Warning. Always interpret an equality sign "=" involving Big Oh expressions as "$\in$" or "$\subseteq$".

Exercise 3. You find in a paper the formula $O(f(n)) - O(f(n)) = 0$, and subsequently the authors derive numerous nonsensical results from this formula. What was the error of the authors, and what should the correct right-hand side of this formula have been?

Exercise 4. Suppose that $f(n) = O(g(n))$ and $g(n) = O(h(n))$. Show that the Big Oh notation is transitive in the sense that $f(n) = O(h(n))$ holds.

§2 Simple Criteria

How can we prove that $f(n)$ is an element of $O(g(n))$? We derive in this section some basic criteria that are helpful in this regard. Our first lemma is a simple way to establish $f(n) = O(g(n))$, but it is not always applicable.

Lemma 1. *Let f and g be functions from the positive integers to the complex numbers such that $g(n) \neq 0$ for all $n \geq n_0$ for some positive integer n_0. If the limit $\lim_{n\to\infty} |f(n)/g(n)|$ exists and is finite then $f(n) = O(g(n))$.*

Proof. If $\lim_{n\to\infty} |f(n)/g(n)| = C$, then for each $\epsilon > 0$ there exists a positive integer $n_0(\epsilon)$ such that $C - \epsilon \leq |f(n)/g(n)| \leq C + \epsilon$ for all $n \geq n_0$; this shows that $|f(n)| \leq (C + \epsilon)|g(n)|$ for all integers $n \geq n_0(\epsilon)$. It follows that $f(n) = O(g(n))$. □

The assumption that $g(n) \neq 0$ for almost all n is usually satisfied for the functions that occur in the analysis of algorithms. However, the limit of the quotient $|f(n)/g(n)|$ might not necessarily exist.

Example 2. If $f(n) = (1 + (-1)^n)n^2$ and $g(n) = n^2$, then the limit $\lim_{n\to\infty} |f(n)/g(n)|$ does not exist because the value of $|f(n)/g(n)|$ keeps fluctuating between 0 and 2. However, we have $f(n) = O(g(n))$.

Let us recall a notion from calculus. Suppose that $x = (x_n)_{n=1}^{\infty}$ is a sequence numbers with values in the extended real line $[-\infty, \infty]$; that is, the value of x_n is a real number or $\pm\infty$ for each $n \in \mathbf{Z}_+$. We let b_k denote the least upper bound of the tail $(x_k, x_{k+1}, \dots)$ of the sequence x, that is, we define

$$b_k = \sup\{x_k, x_{k+1}, \dots\} \quad \text{for all } k \in \mathbf{Z}_+.$$

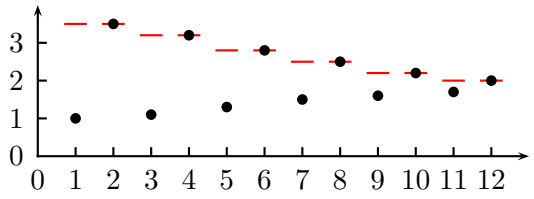
The numbers b_k form a nonincreasing sequence. The limit superior of the sequence x is defined as

$$\limsup_{n\to\infty} x_n = \lim_{k\to\infty} b_k.$$

The limit superior exists for all sequences over the extended real numbers.

Example 3. If $x_n = 1 + (-1)^n$, then the limit superior of the sequence (x_n) is given by $\limsup_{n\to\infty} x_n = 2$, since the least upper bound b_k of the tail sequence $(x_n)_{n=k}^{\infty}$ is $b_k = 2$ for all $k \geq 1$.

Example 4. We illustrate the concept of the limit superior with another simple example. Suppose that $x = (x_n)_{n=1}^{\infty}$ is a sequence of real numbers that can be partitioned into an increasing subsequence $(x_{2n-1})_{n=1}^{\infty}$ and a decreasing subsequence $(x_{2n})_{n=1}^{\infty}$, as shown in the next figure.



The sequence x is depicted by black dots and the sequence of least upper bounds (b_k) is depicted by lines. Notice that the values b_k are monotonically decreasing.

We use the limit superior to refine the criterion given in Lemma 1.

Lemma 2. *Let f and g be functions from the positive integers to the complex numbers such that $g(n) \neq 0$ for all $n \geq n_0$ for some positive integer n_0. We have $\limsup_{n\to\infty} |f(n)/g(n)| < \infty$ if and only if $f(n) = O(g(n))$.*

Proof. If $\limsup_{n\to\infty} |f(n)/g(n)| = C$, then for each $\epsilon > 0$ we have

$$|f(n)|/|g(n)| > C + \epsilon$$

for at most finitely many positive integers; so $|f(n)| \le (C+\epsilon)|g(n)|$ holds for all integers $n \ge n_0(\epsilon)$ for some positive integer $n_0(\epsilon)$, and this proves that $f(n) = O(g(n))$.

Conversely, if $f(n) = O(g(n))$, then there exists a positive integer n_0 and a constant C such that $g(n) \neq 0$ and $|f(n)|/|g(n)| \le C$ for all $n \ge n_0$. This implies that $\limsup_{n\to\infty} |f(n)/g(n)| \le C$. □

Exercise 5. Suppose that $p(n) = a_0 + a_1 n + \cdots + a_m n^m$ is a polynomial of degree m with complex coefficients. Show that $p(n) = O(n^k)$ for all $k \ge m$, and $p(n) \neq O(n^\ell)$ for $0 \le \ell < m$.

Exercise 6. Show that for fixed k, we have

$$\binom{n}{k} = \frac{n^k}{k!} + O(n^{k-1}) \quad \text{and} \quad \binom{n+k}{k} = \frac{n^k}{k!} + O(n^{k-1}),$$

where $\binom{n}{k} = n(n-1)(n-2)\cdots(n-k+1)/k!$ is the binomial coefficient.

Exercise 7. Show that $n/(n+1) = 1 + O(1/n)$.

Exercise 8. Sometimes $f(n) = O(g(n))$ is written as $f(n) \preceq g(n)$. Exercise 2 asserts that this relation is reflexive, $f(n) \preceq f(n)$, and Exercise 4 shows that this relation is transitive. Prove that there exist functions $f(n)$ and $g(n)$ such that neither $f(n) \preceq g(n)$ nor $g(n) \preceq f(n)$ holds; so one cannot always compare the asymptotic growth of two functions.

Exercise 9. Prove or disprove: $e^n = O(2^n)$.

Exercise 10. Prove or disprove: $n^{\ln n} = O(e^{(\ln n)^2})$.

§3 Big Oh Calculus

One advantage of Big Oh expressions is that they are generally easier to manipulate than accurate expressions. In this section, we collect some rules that are simple consequences of the Big Oh definition.

Constants. If c is a nonzero constant, then

$$cO(f(n)) = O(f(n)), \tag{1}$$
$$O(cf(n)) = O(f(n)). \tag{2}$$

Idempotency. The Big Oh operator is idempotent, meaning that

$$O(O(f(n)) = O(f(n)). \tag{3}$$

Multiplications. The multiplication of Big Oh expressions follows the rules

$$O(f(n))O(g(n)) = O(f(n)g(n)), \tag{4}$$
$$O(f(n)g(n)) = f(n)O(g(n)). \tag{5}$$

Absorbtion. We can simplify Big Oh expressions using the rule

$$O(f(n)) + O(g(n)) = O(g(n)) \text{ provided that } f(n) = O(g(n)). \tag{6}$$

Powers. For all positive integers k, we have

$$(f(n) + g(n))^k = O\big((f(n))^k\big) + O\big((g(n))^k\big). \tag{7}$$

Linear Combinations. If $f(n) = O(h(n))$ and $g(n) = O(h(n))$, then

$$af(n) + bg(n) = O(h(n)) \quad \text{for all } a, b \in \mathbf{C}. \tag{8}$$

Swap. The next rule allows you to swap Big Oh terms.

$$\text{If } f(n) = g(n) + O(h(n)) \text{ then } g(n) = f(n) + O(h(n)). \tag{9}$$

You should prove *all* these rules to gain a good working knowledge of Big Oh calculations.

Exercise 11. Prove the multiplication by constant rules (1) and (2).

Exercise 12. Prove the idempotency rule given by equation (3).

Exercise 13. Prove the multiplication rules (4) and (5).

Exercise 14. Prove the absorbtion rule (6) and the power rule (7).

Exercise 15. Prove the linear combination rule given in (8).

Exercise 16. Prove the rule (9).

Exercise 17. Find examples for each of the rules (1)–(9) and at least five examples of an erroneous usage of the Big Oh notation in recent papers or books of Computer Science.

§4 Analysis of the Running Time

One source for Big Oh expressions in the analysis of algorithms are estimates for the worst case running time of an algorithm. Typically, we would like to get a rough idea how fast the algorithm will be for an input of a certain size. We can use the following set of rules to obtain such an estimate.

1. Suppose that we have a language with primitive operations that include assignments $\leftarrow$, arithmetic operations such as addition $+$, subtraction $-$, multiplication $*$, comparison operators $=$, $\neq$, $\leq$, and $\geq$, a dereferencing operator $*$, an array index operator [ ], and more. We assume that the primitive operations have constant time $O(1)$.

2. The running time of a compound statement

   $S_1; S_2;$

   is $O(t_1) + O(t_2)$ if the running times of the statements S_1 and S_2 are respectively $O(t_1)$ and $O(t_2)$.

3. The running time of the conditional expression

   if S_1 **then** S_2; **else** S_3; **fi**;

   is bounded by $O(t_1) + O(\max(t_2, t_3))$, where t_k is the running time of the statement S_k with $k = 1, 2, 3$.

4. The running time of a for-loop

   for k = a **to** b **do**
   S_1;
   od;

   is $O((b - a + 1)\max(t_1, 1))$, where t_1 is a bound on the running time of the statement S_1.

5. The running time of a call to a function f that is defined by

   f(*param*)
   S_2;

   is given by $O(t_1) + O(t_2)$, where $O(t_1)$ is the time it takes to perform the assigments of the parameter and $O(t_2)$ is a bound on the running time of the function body S_2.

This list is incomplete, but it should be clear how to create rules for further language constructs. It should be emphasized that one may obtain too

pessimistic estimates for the running time; a more refined analysis that counts the operations more precisely is sometimes appropriate.

Example 5. Suppose that you want to write a program to evaluate a polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

so for a given input x, you would like to know the value $y = p(x)$. You can save multiplications by using the Horner scheme

$$p(x) = (\cdots((a_n x + a_{n-1})x + a_{n_2})x + \cdots a_1)x + a_0.$$

The following algorithm is based on the Horner scheme:

```
1: horner(int a[ ], int n, int x)
2: int res = a[n];
3: for k = n − 1 down to 0 do
4:     res = res * x + a[i];
5: od
6: return res;
```

We need $O(1)$ time steps to assign the parameters in line 1 and $O(1)$ timesteps do the assignment in line 2. The statement in line 4 can be executed in $O(1)$ steps, and the for-loop is repeated n times. So lines 3–5 have $O(n)$ running time, and line 6 is executed once in $O(1)$ time. Therefore, we can conclude that the overall running time of the algorithm is $O(1) + O(1) + O(n) + O(1) = O(n)$.

Exercise 18. Give a more detailed analysis of the horner algorithm which takes into account that the primitive operations can have different execution times, such as τ_* for the multiplication operation, τ_+ the execution time for an addition operation, $\tau_{[]}$ the time for the dereferencing operation and so on. Account for all operations in the execution of this algorithm without using Big Oh notation.

Exercise 19. Gives at least two different reasons why in modern computer architectures the approach taken in the previous exercise cannot provide a realistic measure of the total execution time.

§5 Examples

In this section, we give a number of examples that illustrate the material that we have learned so far. Our first example is borrowed from Sedgewick and Flajolet [4].

Example 6. Suppose that an algorithm uses a for loop with $n+1$ repetitions and calls in the k iteration a subroutine that takes $k!$ steps, then the overall complexity of the algorithm is proportional to the sum $\sum_{k=0}^{n} k!$. The terms are so rapidly increasing that the last term determines gives a good estimate for the whole sum. Indeed,

$$\sum_{k=0}^{n} k! = n! \left(1 + \frac{1}{n} + \sum_{k=0}^{n-2} \frac{k!}{n!} \right).$$

The $n-1$ terms in the last sum are less than $(n(n-1))^{-1} = O(n^{-2})$, so the sum is $O(n^{-1})$. Therefore, we have $\sum_{k=0}^{n} k! = n!(1 + O(n^{-1}))$.

Our second example, taken from [1], shows how an estimate of an exact number can lead to an estimate that is easier to interpret. This example illustrates how rule (5) is used in practice. We note here without proof that $(1 + O(\epsilon))^{-1} = 1 + O(\epsilon)$; this property will become obvious in the next section.

Example 7. Suppose that you have determined that the number of winning positions in a game are given by

$$W = \lfloor n/k \rfloor + \frac{1}{2}k^2 + \frac{5}{2}k + 3 \quad \text{with} \quad k = \lfloor \sqrt[3]{n} \rfloor.$$

We observe that $k = n^{1/3} + O(1) = n^{1/3}(1 + O(n^{-1/3}))$, where the last equality follows from (5). Consequently, we have

$$k^2 = n^{2/3}(1 + O(n^{-1/3}))^2 = n^{2/3}(1 + O(n^{-1/3})) = n^{2/3} + O(n^{1/3}).$$

We also have

$$\begin{aligned} \lfloor n/k \rfloor &= n^{1-1/3}(1 + O(n^{-1/3}))^{-1} + O(1) \\ &= n^{2/3}(1 + O(n^{-1/3})) + O(1) = n^{2/3} + O(n^{1/3}). \end{aligned}$$

Therefore, the total number of winning position is

$$\begin{aligned} W &= n^{2/3} + O(n^{1/3}) + \frac{1}{2}(n^{2/3} + O(n^{1/3})) + \frac{5}{2}(n^{1/3} + O(1)) \\ &= \frac{3}{2}n^{2/3} + O(n^{1/3}). \end{aligned}$$

A frequent task in the analysis of algorithms is to estimate a sum for which no closed form solution is known. The following example from [2] shows how partitioning of the sum can lead to good estimates.

Example 8. Suppose that we want to estimate the sum

$$f(n) = \sum_{d=3}^{n/2} \frac{1}{d(n/d)^d}$$

for large n. We observe that the terms within the sum behave quite differently for different sizes of d. In such a case, it is often advisable to divide the sum into smaller parts and try to estimate the parts. For example, we can express $f(n)$ as

$$f(n) \le \Sigma_1 + \Sigma_2 + \Sigma_3 = \sum_{d=3}^{8} \frac{1}{d(n/d)^d} + \sum_{d=8}^{\sqrt{n}} \frac{1}{d(n/d)^d} + \sum_{d=\sqrt{n}}^{n/2} \frac{1}{d(n/d)^d}.$$

There is not general rule how to break such a sum into parts, and this can be considered as the art of asymptotics. We notice that

$$\Sigma_1 \le \sum_{d=3}^{8} \frac{1}{3(n/8)^3} = O(n^{-3}),$$

where we replaced the values of d with its bound so that each individual term in the sum becomes larger. We repeat the same trick for Σ_2 and get

$$\Sigma_2 \le \sum_{d=8}^{\sqrt{n}} \frac{1}{8(n/\sqrt{n})^8} = O(n^{-4}\sqrt{n}),$$

where the bound is obtained by observing that we sum $O(\sqrt{n})$ terms of size $O(n^{-4})$. Repeating once again the same trick, we notice that the third sum Σ_3 consists of very small terms, since

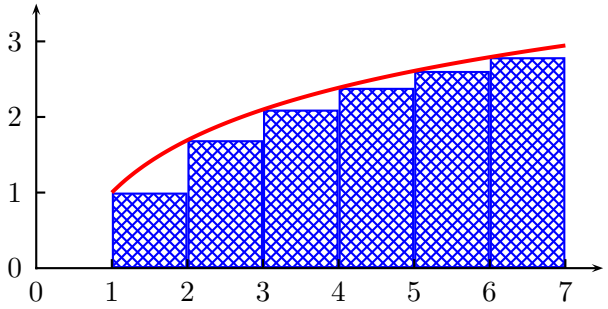
$$\Sigma_3 \le \sum_{d=\sqrt{n}}^{n/2} \frac{1}{\sqrt{n}2^{\sqrt{n}}} = O\left(\frac{\sqrt{n}}{2^{\sqrt{n}}}\right).$$

Therefore, $f(n) = O(n^{-3}) + O(n^{-4}\sqrt{n}) + O(\sqrt{n}2^{-\sqrt{n}}) = O(n^{-3})$.

§6 Finite Sums

It is often possible to obtain a useful estimate of the asymptotic value of a finite sum by comparing it with an integral. This is a useful application of knowledge that you have gained in your calculus courses. The next figure

shows that the sum of the values that is represented by the height of the shaded bars can be approximated by the integral of the function values; the next theorem allows us to determine the error that that we make in such a comparison.



We need to introduce some notation. If x is a real number, then we denote by $\lfloor x \rfloor$ the largest integer less than or equal to x, and by $\{x\}$ the fractional part $x - \lfloor x \rfloor$.

Theorem 3 (Euler)**.** *If a function $f\colon \mathbf{R} \to \mathbf{R}$ has a continuous derivative on the interval $[1, n]$, then*

$$\sum_{k=1}^{n-1} f(n) = \int_1^n f(x)dx - \frac{1}{2}(f(n) - f(1)) + \int_1^n (\{x\} - 1/2)f'(x)dx.$$

Proof. Integration by parts shows that for an integer k one obtains

$$\begin{aligned} \int_k^{k+1} (\{x\} - \frac{1}{2})f'(x)dx &= (x - k - \frac{1}{2})f(x)\Big|_k^{k+1} - \int_k^{k+1} f(x)dx \\ &= \frac{1}{2}(f(k+1) - f(k)) - \int_k^{k+1} f(x)dx. \end{aligned}$$

If we add both sides of the equation for $1 \leq k < n$, then we obtain

$$\int_1^n (\{x\} - \frac{1}{2})f'(x)dx = \sum_{k=1}^{n-1} f(k) + \frac{1}{2}(f(n) - f(1)) - \int_1^n f(x)dx,$$

and this proves our claim. □

Remark 2. We could have written Euler's summation formula in the form

$$\sum_{k=2}^{n} f(k) = \int_1^n f(x)dx + \int_1^n \{x\}f'(x)dx,$$

but the form given in Theorem 3 has a natural generalization using so-called Bernoulli polynomials, as we will see below.

Example 9. We want to show that the Harmonic number $H_{n-1} = \sum_{k=1}^{n-1} 1/k$ can be estimated by

$$H_{n-1} = \log n + \gamma + O(n^{-1}) \quad \text{for some constant } \gamma.$$

If we set $f(x) = 1/x$ then its derivative is given by $f'(x) = -1/x^2$ and Euler's summation formula yields

$$\begin{aligned} \sum_{k=1}^{n-1} \frac{1}{k} &= \ln n - (\frac{1}{n} - 1) - \int_1^n \frac{\{x\}}{x^2} dx \\ &= \ln n + 1 - \frac{1}{n} - \int_1^\infty \frac{\{x\}}{x^2} dx + \int_n^\infty \frac{\{x\}}{x^2} dx \end{aligned}$$

The improper integral $\int_1^\infty \{x\} x^{-2} \mathrm{d}x$ exists, since it is dominated by $\int_1^\infty x^{-2} \mathrm{d}x$, and

$$0 \leq \int_n^\infty \frac{\{x\}}{x^2} \mathrm{d}x \leq \int_n^\infty \frac{1}{x^2} \mathrm{d}x = \frac{1}{n}.$$

Thus, we can conclude that

$$\sum_{k=1}^{n-1} \frac{1}{k} = \ln n + 1 - \int_1^\infty \frac{\{x\}}{x^2} dx + O(n^{-1}), \tag{10}$$

and this proves our claim if we set $\gamma = 1 - \int_1^\infty \{x\}/x^2 dx$. In fact, it follows from (10) that γ is the famous Euler-Mascheroni constant

$$\gamma = \lim_{n \to \infty} \sum_{k=1}^{n-1} 1/n - \ln n \approx 0.577215 \cdots .$$

Exercise 20. Show that

$$\sum_{k=1}^{n-1} k^a = \frac{n^{1+a}}{1+a} + O(n^a) \text{ if } a \geq 0.$$

Exercise 21. The Riemann zeta function is defined by the equation

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} \text{ if } s > 1 \text{ and by } \zeta(s) = \lim_{x \to \infty} \left(\sum_{n=1}^{x} \frac{1}{n^s} - \frac{x^{1-s}}{1-s} \right) \text{ if } 0 < s < 1.$$

Show that

$$\sum_{k=1}^{n} \frac{1}{k^s} = \frac{n^{1-s}}{1-s} + \zeta(s) + O(n^{-s}) \quad \text{if } s > 0 \text{ and } s \neq 0.$$

Euler's General Summation Formula. Euler developed the idea behind Theorem 3 even further. We contend ourselves with a brief exposition of the general summation formula without giving proofs. Let us first state the result and then explain all the notations.

Theorem 4. *Suppose that $f(x)$ has m continuous derivatives in the interval $[1, n]$. Then*

$$\sum_{k=1}^{n-1} f(k) = \int_1^n f(x)dx + \sum_{k=1}^{m} \frac{B_k}{k!}(f^{(k-1)}(n) - f^{(k-1)}(1)) + R_{mn},$$

where

$$R_{mn} = \frac{(-1)^{m+1}}{m!} \int_1^n B_m(\{x\}) f^{(m)}(x)dx.$$

The coefficients B_k in the previous theorem are the Bernoulli numbers that are defined by the infinite series

$$\frac{z}{e^z - 1} = \sum_{k=0}^{\infty} \frac{B_k}{k!} z^k.$$

One can show that

$$(B_0, B_1, B_2, B_3, B_4) = (1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}),$$

and $B_3 = B_5 = B_7 = B_9 = \cdots = 0$. The residue term R_{nm} contains a so-called Bernoulli polynomial $B_m(x)$ that is defined by

$$B_m(x) = \sum_{k=0}^{n} \binom{n}{m} B_k x^{m-k}.$$

Example 10. We illustrate how Theorem 4 can be used to derive Stirling's approximation formula for $n!$, following Knuth [3]. If we set $f(x) = \ln x$, then $f^{(k-1)}(x) = (-1)^k(k-2)!/x^{k-1}$ and Theorem 4 yields

$$\ln(n-1)! = n\ln n - n + 1 - \frac{1}{2}\ln n + \sum_{k=2}^{m} \frac{B_k(-1)^k}{k(k-1)} \left(\frac{1}{n^{k-1}} - 1\right) + R_{mn}.$$

We note that the limit σ given by

$$\lim_{n\to\infty} \left(\ln n! - n\ln n + n - \frac{1}{2}\ln n\right) = 1 + \sum_{k=2}^{m} \frac{B_k(-1)^{k+1}}{k(k-1)} + \lim_{n\to\infty} R_{mn}$$

exists, since $\lim_{n\to\infty} R_{mn} = -\frac{1}{m}\int_1^\infty B_m(\{x\})/x^m dx$, and this improper integral exists since it is bounded by $c\int_1^\infty x^{-m}dx$ for some constant c. Therefore, we obtain

$$\ln n! = (n+\frac{1}{2})\ln n - n + \sigma + \sum_{k=2}^{m} \frac{B_k(-1)^k}{k(k-1)n^{k-1}} + O\left(\frac{1}{n^m}\right).$$

In particular, if we choose $m = 5$, then we obtain

$$\ln n! = (n+\frac{1}{2})\ln n - n + \sigma + \frac{1}{12n} - \frac{1}{360n^3} + O\left(\frac{1}{n^5}\right).$$

Taking the exponential of both sides yields

$$n! = e^\sigma \sqrt{n}\left(\frac{n}{e}\right)^n \exp\left(\frac{1}{12n} - \frac{1}{360n^3} + O\left(\frac{1}{n^5}\right)\right).$$

It can be shown that $e^\sigma = \sqrt{2\pi}$. Using the expansion $\exp(x) = 1 + z + z^2/2! + z^3/3! + z^4/4! + O(z^5)$, we obtain as a final result

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \frac{1}{12n} + \frac{1}{288n^2} - \frac{139}{51840n^3} - \frac{571}{2488320n^4} + O\left(\frac{1}{n^5}\right)\right).$$

As a mnemonic, the product of the numbers $1, 2, \ldots, n$ is roughly equal to the product of the n equal numbers (n/e); the above expressions is of course much more precise than that.

Exercise 22. Check the last step in the derivation of Stirling's formula and show that the two expressions for $n!$ are equal.

Exercise 23. Approximate 8! using Sterling's formula.

§7 Further Reading

The book [1] contains an in-depth discussion of the tricks of the trade in asymptotics. Knuth [3] gives a brief overview of asymptotic techniques. Applications of such methods can be found throughout *The Art of Computer Programming* book series. A discussion of asymptotic series is contained in [4].

References

[1] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, MA, 1994.

[2] D.H. Greene and D.E. Knuth. *Mathematics for the Analysis of Algorithms*. Birkhäuser, Boston, 1981.

[3] D.E. Knuth. *The Art of Computer Programming – Fundamental Algorithms*, volume I. Addison-Wesley, Reading, MA, 3rd edition, 1997.

[4] R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley, Reading, MA, 1996.