Logarithms and Asymptotic Complexity

1 Asymptotic Notation

There are three asymptotic symbols we use in this course: O, Ω, and Θ. You need to understand the intuitive meaning of each.

O (say, "big O") means "Eventually approximately no greater than a constant multiple of…" For example, if $F(n) = 2n^3 + n^2 + 5n + 17$, we could write $F(n) = O(n^3)$, meaning that, for large values of n, $F(n)$ is "approximately" no greater than a constant times n^3. (That constant is 2.) We could also write $F(n) = O(n^4)$, but we could not write $F(n) = O(n^2)$, because as n gets greater and greater, the ratio of $F(n)$ to n^2 eventually exceeds any constant. Similarly Ω means "Eventually approximately no less than a constant multiple of…" This means that $F(n) = O(G(n))$ if and only if $G(n) = \Omega(F(n))$. We also have Θ, which means simultaneously O and Ω. For example, all of these statements are true:

(i) $5n + 8 = O(n)$

(ii) $5n + 8 = O(n^2)$

(iii) $5n + 8 = O(n^3)$

(iv) $3n^2 + 5n + 8 = O(n^2)$

(v) $3n^2 + 5n + 8 = O(n^3)$

(vi) $5n + 8 = \Theta(n)$

(vii) $3n^2 + 5n + 8 = \Theta(n^2)$

(viii) $3n^2 + 5n + 8 = \Omega(n^2)$

(ix) $3n^2 + 5n + 8 = \Omega(n)$

(x) $2n^3 + 3n^2 + 5n + 8 = \Theta(n^3)$

On the other hand, the statements in the following list are **false**.

(xi) $5n + 8 = \Omega(n^2)$

(xii) $3n^2 + 5n + 8 = \Omega(n^3)$

(xiii) $3n^2 + 5n + 8 = \Theta(n)$

(xiv) $3n^2 + 5n + 8 = O(n)$

(xv) $2n^3 + 3n^2 + 5n + 8 = \Omega(n^2)$

The numbers 5, 8, 3, and 2 in the statements above are arbitrary. I could have replaced them by any positive constants.

Meaning of Notation. The equal sign used in asymptotic notation does not mean equals. Symbols such as $O(n)$ and $\Theta(n)$ are not the names of functions, but rather, *classes* of functions, specifically *asymptotic classes* of functions.

- In the statement $5n^2 + 2n + 8 = O(n^2)$, $5n^2 + 2n + 8$ is a function, but $O(n^2)$ is not: it is the class of **all** functions which are asymptotically equivalent to n^2. The equal sign does not mean equality, it means membership, and the statement really should be written $5n^2 \in O(n^2)$. But, for historical reasons, it's written as "=" and it's too late to change it.
- Similarly, we can write $O(n^2) = O(n^3)$, but we **cannot** write $O(n^3) = O(n^2)$. How can this be? Equality is supposed to be symetric! Everyone knows that if $a = b$, then $b = a$. In this case, the "=" sign does **not** mean equality, it means *containment.* The statment should really be written $O(n^2) \subset O(n^3)$, meaning that one asymptotic class is a subclass of the other. But, again, we're stuck with that notation.

Why Asymptotic Notation? The time complexity of a program is the amount of time it takes to run, and the space complexity is how much of a machine's memory it uses. But how do we measure time? In seconds? Milliseconds? Those measurements are not robust. If a faster machine is used, the same program takes less time, but that has nothing to do with the program itself. Asymptotics give us a way of measuring the time (or space) complexity of a program without knowing anything about the machine. For example, the classic sorting algorithm *bubblesort* takes $O(n^2)$ time to sort an array of n items, which means it's bounded by Kn^2 where K is a constant which depends on the machine. On the other hand, the standard implementation of the algorithm *mergesort* takes $\Theta(n \log n)$ time and $\Theta(n)$ space.

1.1 Polynomial Time and Space

The *polynomial class* of functions, written $\mathcal{P}$ or $\mathcal{P}(n)$, is the union of all functions asymptotic to polynomial functions, *i.e.,*$F \in \mathcal{P}$ if there is some constant k such that $F(n) = O(n^k)$ for some constant k. Functions that are less than any polynomial function, such as logarithms, are also $\mathcal{P}$, since (for example) $\log n < n$, and thus $\log n = O(n)$. If we say a specific algorithm takes polynomial time, we mean that, with an input of size n, measured in bits, the algorithm runs in $\mathcal{P}(n)$ time. We define polynomial space similarly.

We write $\mathcal{P}$–TIME to be the class of all problems which can be solved by polynomial time algorithms; similarly $\mathcal{P}$–SPACE consists of all problems which can be solved by polynomial space algorithms. It is easy to prove that $\mathcal{P}\text{–TIME} \subseteq \mathcal{P}\text{–SPACE}$, but up to now, no one has been able to prove that those two classes are either equal or unequal, although "everyone" believes they are unequal.

2 Logarithms

If you plan to work in any technical area, such as science, engineering, or programming, you need to understand logarithms.

You should have learned logarithms in high school. However, through no fault of your own, many of you were deprived of a proper education because of the Covid lockdown. There are ways to catch up.

A Logarithm is an Exponent. We write $\log_b a = c$ if $b^c = a$. b is called the *base* of the logarithm, and a is the argument. In applications, the base is frequently understood. In science, engineering, and most other practical applications, the default base is 10, that is, $\log x$ means $\log_{10} x$. In mathematics, the default base is $e = \sum_{n=0}^{\infty} \frac{1}{n!} = 1 + 1 + \frac{1}{2} + \frac{1}{6} + \cdots = 2.71828\ldots$. The natural logarithm of x has base e and is written $\ln x$. In computer science, including this course, the default base is 2. Thus, $\log \frac{1}{2} = -1$, $\log 1 = 0$, $\log 2 = 1$, $\log 4 = 2$, $\log 64 = 6$, *etc.*

Here are some fundamental identities involving exponents and logarithms, which you must thoroughly understand, not just memorize.

(xvi) $a^{bc} = (a^b)^c$

(xvii) The base of a logarithm must be positive and cannot be 1. (In practice, the base is usually greater than 1, although theoretically it could be a fraction, such as $\frac{1}{2}$.)

(xviii) The argument of a logarithm must be positive.

(xix) If $b > 1$, $\log_b x$ is positive if $x > 1$, negative if $x < 1$, and zero if $x = 1$.

(xx) $\log_b b^c = c$

(xxi) $b^{\log_b a} = a$

(xxii) $\log_b(xy) = \log_b x + \log_b y$

(xxiii) $\log_b \dfrac{x}{y} = \log_b x - \log_b y$

(xxiv) $\log_b x^y = y \log_b x$

(xxv) $\log_c x = \dfrac{\log_b x}{\log_b c}$

The last four rules holds for any base b.

Exercise 1. Evaluate these expressions. In each case, the value is an integer or a simple fraction. Recall that roots are fractional powers; for example, $\sqrt{x} = x^{\frac{1}{2}}$ and $\sqrt[3]{x^2} = x^{\frac{2}{3}}$.

(xxvi) $\log 8$

(xxvii) $\log_4 \sqrt{2}$

(xxviii) $\log_3 9$

(xxix) $\log_4 2$

(xxx) $\log\left(\frac{1}{2}\right)$

(xxxi) $\log_3 \sqrt[3]{9}$

(xxxii) $\log(\sqrt{8} - \sqrt{2})$

(xxxiii) $2^{\log_4 25}$ (Yes, it's an integer!)

(xxxiv) $\log\log\log\log 65536$

Asymptotic Classes using Logarithms. Logarithm functions grow more slowly than polynomial functions. For example, $\log n$ grows more slowly than n, or even $n^{.01}$.

Exercise 2. Mark each of these statements True (T) or False (F).

(xxxv) ________ $n^2 = \Theta(n^3)$

(xxxvi) ________ $\log n = O\big(n^{0.01}\big)$

(xxxvii) ________ $\log n^2 = \Theta(\log n^3)$

(xxxviii) ________ $\log n = O\big(n^{0.01}\big)$

(xxxix) ________ $\log^2 n = O(\log n)$

(xl) ________ $\log\log n = O(\log n)$

(xli) ________ If $f(n) = \Theta(n)$, then we know that $\log f(n) = \Theta(\log n)$.

(xlii) ________ If $\log g(n) = \Theta(\log n)$, then we know that $g(n) = \Theta(n)$.

2.1 Logarithmic Scales

Certain scientific quantities are expressed using a logarithmic scales.

(xliii) The Richter measure of the strength of a seismic event is proportional to the base 10 logarithm of its energy. An earthquake of magnitude 5 is ten times as powerful as an earthquake of magnitude 4.

(xliv) What other physical scales used in practive are logarithmic? I can think of three more.

3 Asymptotic Time Complexity of Code Fragments

Exercise 3 In each case, find the asymptotic time complexity in terms of n.

(xlv)
```
for(int i = 0; i < n; i++)
  cout << "Hello";
```

(xlvi)
```
for(int i = 0; i < n; i++)
  for(int j = 0; j < n; j++)
    cout << "Hello";
```

(xlvii)
```
for(int i = 0; i < n; i++)
  for(int j = i; j < n; j++)
    cout << "Hello";
```

(xlviii)
```
for(int i = 1; i < n; i=2*i)
  cout << "Hello";
```

(xlix)
```
for(int i = n; i > 1; i=i/2)
  cout << "Hello";
```

(l)
```
for(int i = 2; i < n; i=i*i)
  cout << "Hello";
```

(li)
```
for(int i = n; i > 1; i=sqrt(i))
  cout << "Hello";
```

(lii)
```
for(int i = 0; i < n; i=i++)
  for(int j = 1; j < i; j=2*j
    cout << "Hello";
```

(liii)
```
for(int i = 1; i < n; i=i++)
  for(int j = i; j < n; j=2*j)
    cout << "Hello";
```

4 The Greek Alphabet

If you plan to work in any technical area, such as science, engineering, or programming, you need to know Roman numerals, as well as the Greek alphabet, both upper and lower case. Which are frequently used in technical writing. We list the Greek alphabet below, giving the name, upper case symbol, and lower case symbol for each of the 24 members of the alphabet.

alpha	A	α	beta	B	β	gamma	Γ	γ	delta	Δ	δ, ∂
epsilon	E	ϵ, ε	zeta	Z	ζ	eta	H	η	theta	Θ	θ
iota	I	ι	kappa	K	κ	lambda	Λ	λ	mu	M	μ
nu	N	ν	xi	Ξ	ξ	omicron	O	o	pi	Π	π
rho	P	ρ, ϱ	sigma	Σ	σ, ς	tau	T	τ	upsilon	Υ	υ
phi	Φ	ϕ	chi	X	χ	psi	Ψ	ψ	omega	Ω	ω