Lecture Notes for Chapter 3: Growth of Functions

Having come up with your new and innovative algorithm, how do you measure its efficiency?
Certainly, we would prefer to design an algorithm which to be as efficient as possible, therefore we will require some method to prove that it will really operate as well as we had expected. Also, some algorithms are more efficient than others. But what do we mean by efficient? Do we mean CPU/memory/disk usage, and so on? And how do we measure the efficiency of an algorithm?

To mix performance (the amount of CPU/memory/disk usage) with complexity (how well the algorithm scales) is one of the most common misconceptions that occurs when analyzing the efficiency of an algorithm.

Determining the running time of an algorithm, i.e. CPU time, is not a particularly good indication of efficiency. There exist many aspects of the running time that can be influenced greatly by the performance of the underlying hardware on which the code will run, the compiler used to generate the machine code, in addition to the quality of the code. It is determined, therefore, how a designed algorithm acts as the size of the input increases. When n (input size) goes to infinity, for example, what effect would that have on running time?

Generally speaking, complexity (asymptotic analysis or asymptotic notation) is used to describe the efficiency of an algorithm. In asymptotic analysis, we don't measure the actual running time, on the contrary, we prefer to evaluate the efficiency of an algorithm in terms of input size. We calculate, how does the amount of time taken by an algorithm increases with the number of items of input. There exist two types of efficiency: time efficiency and space efficiency. Other measures could include transmission speed, temporary disk usage, long-term disk usage, power consumption, total cost of ownership, response time to external stimuli, etc. Time efficiency, also called time complexity, indicates the amount of time that an algorithm takes to run. Space efficiency, also called space complexity, describes how much working storage needed by the algorithm in addition to the space required for its input and output. Today, with the advancement of storage hardware and increased storage capacity, there is little to worry about the value of space required by an algorithm. However, the problem of time has not been reduced quite to the same extent. Furthermore, the analysis experience has shown that for most algorithms, we can obtain considerable improvement in speed than in space.
The time efficiency depends on several aspects, including:

- The quality of the code;
- The nature of the processor; and
- The input data.

In the analysis of time efficiency, the first two are ignored and focus particularly on the third. That is, we will assume that any algorithms being compared

Table 1: Time required to process n input items on different algorithms.

	n			
	10	**50**	**100**	**1,000**
$\log n$	0.0003 seconds	0.0006 seconds	0.0007 seconds	0.0010 seconds
n	0.0010 seconds	0.0050 seconds	0.0100 seconds	0.1000 seconds
$n \log n$	0.0033 seconds	0.0282 seconds	0.0664 seconds	0.9966 seconds
n^2	0.0100 seconds	0.2500 seconds	1.0000 seconds	100.00 seconds
n^3	0.1000 seconds	12.500 seconds	100.00 seconds	1.1574 days
n^4	1.0000 seconds	10.427 minutes	2.7778 hours	3.1710 years
n^6	1.6667 minutes	18.102 days	3.1710 years	3171.0 centuries
2^n	0.1024 seconds	35.702 centuries	4×10^{16} centuries	1×10^{166} centuries
$n!$	362.88 seconds	1×10^{51} centuries	3×10^{144} centuries	1×10^{2554} centuries

have similar code quality and are being executed on the same processor. Hence, the time efficiency will be assumed to depend only on the input data. Also, because of this, we will not measure the time efficiency in any particular time units. Order of growth associates input size to the running time of the algorithm, indeed, the order of growth only specifies how the running time scales as the input grows. An algorithm designer is only interested in how an algorithm is performed on large inputs, because even slow algorithms end up quickly on small inputs.
Let us assume that a sample computer can do 10,000 operations per second. We designed several algorithms that require log n, n, n^2, n^3, n^4, n^6, 2^n, and $n!$ operations to perform a given task on n input items, Table 1 shows the behavior of the designed algorithms on different input items. Table 2 and Table 3 show the explosive growth of 2^n and $n!$ in more details, respectively.

As shown in Table 1, with increasing n, the run-time in algorithms with complexity $logn$, n, $nlogn$, and n^2 has not increased so much. For example, if we increase the size of n from 10 to 100, the $logn$ runtime from 0.0003 seconds to 0.0007 seconds has increased. But for algorithms with complexity 2^n and $n!$, run-time has greatly increased. For 2^n, the runtime has increased from 0.1024 seconds to 4×10^{16} centuries!

In general, the purpose of analyzing algorithms is to study their growth rate. By analyzing algorithms, one can choose among the various available algorithms for a problem, an algorithm with the lowest growth rate.

Table 2: Time required to process n input items using a 2^n algorithm.

n						
15	**20**	**25**	**30**	**35**	**40**	**45**
3.28 seconds	1.75 minutes	55.9 minutes	1.24 days	39.8 days	3.48 years	1.12 centuries

Table 3: Time required to process n input items using a $n!$ algorithm.

n						
11	**12**	**13**	**14**	**15**	**16**	**17**
1.11 hours	13.3 hours	7.20 days	101 days	4.15 years	66.3 years	11.3 centuries

1 Orders of growth

Order of growth in algorithm means how the time for computation increases when you increase the input size. Order of growth provides only a raw description of the behavior of an algorithm. To analyze the efficiency of an algorithm in terms of orders of growth, five asymptotic notations including O (big-oh), Ω (big-omega), Θ (big-theta), o (little-oh), and ω (little-omega) are employed. These notations are used to symbolically express the asymptotic behavior of a given function. In the following discussion, we present, first, these notations informally, and then, after several examples, formal descriptions are presented. We also assume that f(n) and g(n) are positive functions from positive numbers to positive numbers. We are interested to characterize an algorithm's efficiency in terms of running time, hence, f(n) will be an algorithm's running time (how to calculate the f(n) will be described in Chapter 9), and g(n) will be the growth rate of f(n). To calculate f(n), we need to measure the number of elementary steps that the algorithm takes. In other words, the runtime for an algorithm can be measured by counting the number of steps required to solve the problem.

O-notation

$O(g(n))$ characterizes the set of all functions with a lower or same order of growth as $g(n)$ (when n tends towards a particular value or infinity). It only provides an asymptotic upper bound on the growth rate of the function. For example, the following relationships are all true:

$$1000n + 5000 \in O(nlogn), \;\; n^2 \in O(n^2), \;\; \frac{1}{10}n(n-1) \in O(n^2)$$

The first function (1000n + 5000) is linear, i.e. n, and hence have a lower order of growth than $nlogn$, while the last two functions are quadratic and hence has the same order of growth as n^2. On the other hand,

$$n^2 \notin O(n),\ \ \frac{n^3}{1000} \notin O(n^2),\ \ n^5 + n^2 + 100000 \notin O(n^3)$$

Note that the functions n^2 and $\frac{n^3}{1000}$ are quadratic and cubic and hence have a higher order of growth than n and n^2, respectively, and the last function has the fifth-degree polynomial hence has a higher order of growth than n^3.
Also, we have:

$$O(n^3) = \{n^3, n^2 log n, n^2, ...\}$$

The second asymptotic notation, $\Omega(g(n))$, characterizes the set of all functions with a higher or same order of growth as $g(n)$ (when n tends towards a particular value or infinity). For example,

$$n^4 \in \Omega(n^3),\ \ \frac{1}{1000}n(n-1) \in \Omega(n^2),\ \ n^5 + n^2 + 100000 \notin \Omega(2^n)$$

Finally, $\Theta(g(n))$ denotes the set of all functions that have the same order of growth as $g(n)$ (when n tends towards a particular value or infinity). For example:

$$\Theta(n^2) = \{2n^2, 2n^2 + \sqrt{n}, n^2 + n, n^2, n^2 + sin\, n, n^2 + log\, n, ...\}$$

After this informal explanations, in the following, we give the formal definitions.
DEFINITION A function $f(n)$ has order $O(g(n))$, denoted $f(n) \in O(g(n))$, if and only if for all sufficiently large values of n, the value of $f(n)$ is bounded above at most a positive constant multiple of $g(n)$, i.e., if there exists a positive constant c and a nonnegative integer n_0, such that for all $n \geq n_0$ satisfying

$$f(n) \leq cg(n)$$

The definition is depicted in Figure 1.
THEOREM 1. If $f(n) = a_m\ n^{\mathrm{m}} + a_{m-1}\ n^{\mathrm{m-1}} + a_{m-2}\ n^{\mathrm{m-2}} + \ldots + a_1\ m_0 + a_0$, then $f(n) = O(n^m)$?
Proof. We can write:

$$\begin{aligned} f(n) &\leq |f(n)| \\ &= \left|a_m n^m + a_{m-1} n^{m-1} + ... + a_1 n + a_\circ\right| \\ &\leq |a_m n^m| + \left|a_{m-1} n^{m-1}\right| + ... + |a_1 n| + |a_\circ| \\ &\leq n^m \sum_{i=\circ}^{m} |a_i| \end{aligned}$$

therefore, for $C = \sum_{i=\circ}^{m} |a_i|$, we have $f(n) \in O(n^m)$.
According to Theory 1, we will have:

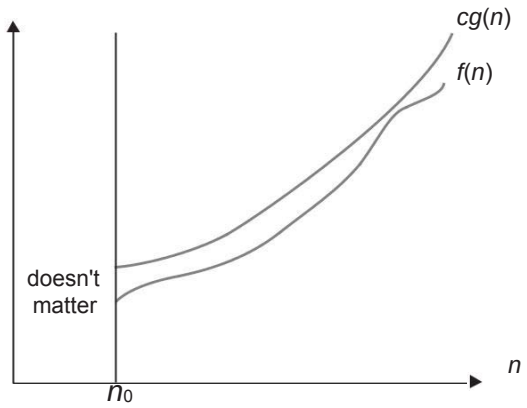

Figure 1: Big-oh notation: $f(n) \in O(g(n))$.

$$\sum_{i=1}^{n} i^0 = n \ \in O(n)$$

$$\sum_{i=1}^{n} i^1 = \frac{n(n+1)}{2} \ \in O(n^2)$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \ \in O(n^3)$$

$$\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2 \ \in O(n^4)$$

and in general:

$$\sum_{i=1}^{n} i^k = 1^k + \ 2^k + \cdots + \ n^k \ \in O(n^{k+1})$$

The following relationships are hold:

1. If the program have k instructions with the growth order $O(n^{m_1}), O(n^{m_2}), ..., O(n^{m_k})$, then the program's growth order is $O(n^m)$ in which m = Maximum $\{m_1, m_2, ..., m_k\}$.
2. $O(f(n))O(g(n)) = O(f(n)g(n))$
3. $f(n) = O(f(n))$

4. $O(cf(n)) = O(f(n))$, c > 0
5. $O(f(n)g(n)) = f(n)O(g(n))$
6. $O(f(n)) + O(g(n)) = O(f(n) + g(n))$.

EXAMPLE 1. Show that if $t(n) = (4n^3 + 5n^2 + 7n)(8\,logn)$ then $t(n) = O(n^3 logn)$.

Suppose $t_1(n) = (4n^3+5n^2+7n)$ and $t_2(n) = (8\,logn)$ then we have: $t_1(n) = O(n^3)$ and $t_2(n) = O(logn)$, hence:

$$t(n) = t_1(n)t_2(n) = O(n^3)O(logn) = O(n^3 logn)$$

EXAMPLE 2. We want to calculate the asymptotic behavior:

$$f\,(n) = \left\lfloor \frac{n}{k} \right\rfloor + \frac{1}{2}k^2 + \frac{5}{2}k - 3$$

where

$$k = \left\lfloor \sqrt[3]{n} \right\rfloor$$

For solving, we have:

$$k = \left\lfloor \sqrt[3]{n} \right\rfloor \Longrightarrow k = \sqrt[3]{n}\left(1 + \mathrm{O}\left(\frac{1}{\sqrt[3]{n}}\right)\right) = n^{\frac{1}{3}}(1 + O(n^{\frac{1}{3}}))$$

$$k^2 = n^{\frac{2}{3}}\left(1 + O\left(n^{-\frac{1}{3}}\right)\right)^2 = n^{\frac{2}{3}}(1 + \left(O\left(n^{-\frac{1}{3}}\right)\right)^2 + 2O(n^{-\frac{1}{3}}))$$

$$k^2 = n^{\frac{2}{3}} + n^{\frac{2}{3}}O\left(n^{\frac{2}{3}}\right) + 2n^{\frac{2}{3}}O(n^{-\frac{1}{3}})$$

$$k^2 = n^{\frac{2}{3}} + O\left(n^{\frac{1}{3}}\right) + O\,(1) = n^{\frac{2}{3}} + O(n^{\frac{1}{3}})$$

$$\left\lfloor \frac{n}{k} \right\rfloor = n^{\frac{2}{3}}\left(1 + O\left(n^{-\frac{1}{3}}\right)\right) + O\,(1) = n^{\frac{2}{3}} + O(n^{\frac{1}{3}})$$

$$f\,(n) = n^{\frac{2}{3}} + O\left(n^{\frac{1}{3}}\right) + \frac{1}{2}\left(n^{\frac{2}{3}} + O\left(n^{\frac{1}{3}}\right)\right) + \frac{5}{2}n^{\frac{1}{3}}(1 + O(n^{-\frac{1}{3}}))$$

$$f\,(n) = \frac{2}{3}n^{\frac{2}{3}} + \frac{3}{2}O\left(n^{\frac{1}{3}}\right) + \frac{5}{2}n^{\frac{1}{3}} + \frac{5}{2}O\,(1) - 3$$

$$f\,(n) = \frac{3}{2}n^{\frac{2}{3}} + O(n^{\frac{1}{3}})$$

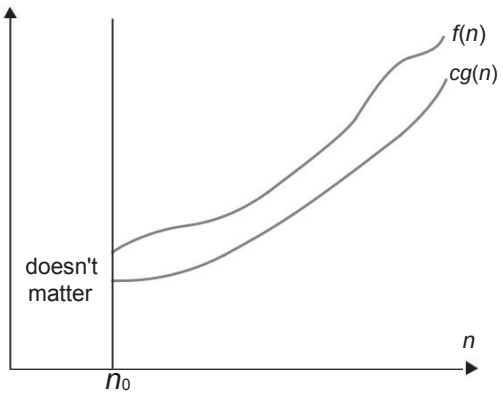

Figure 2: Big-omega notation: $f(n) \in \Omega(g(n))$.

Ω-notation

DEFINITION A function $f(n)$ is said to be in $\Omega(g(n))$, denoted $f(n) \in \Omega(g(n))$, if and only if for all sufficiently large values of n, the value of $f(n)$ is bounded below at most a positive constant multiple of $g(n)$, i.e., if there exists a positive constant c and a nonnegative integer n_0, such that for all $n \geq n_0$ satisfying

$$f(n) \geq cg(n)$$

below is an example of the formal proof that $5n^2 \in \Omega(n)$:

$$5n^2 \geq n, \text{ for all } n \geq 0,$$

for $c = 1$ and $n_0 = 0$.
For other examples, we have:

$$\Omega(n^2) = \{n^2 logn, n^2 loglogn, n^2, n^3, 100n^2 - 10000n, 100n^2 + 10000n...\}$$

The Ω definition is illustrated in Figure 2.

Θ-notation

DEFINITION A function $f(n)$ is said to be in $\Theta(g(n))$, denoted $t(n) \in \Theta(g(n))$, if and only if for all sufficiently large values of n, the value of $f(n)$ is bounded both above and below at most a positive constant multiple of $g(n)$, i.e., if there exists a positive constants c_1 and c_2 and a nonnegative integer n_0, such that for all $n \geq n_0$ satisfying

$$c_2 g(n) \leq f(n) \leq c_1 g(n)$$

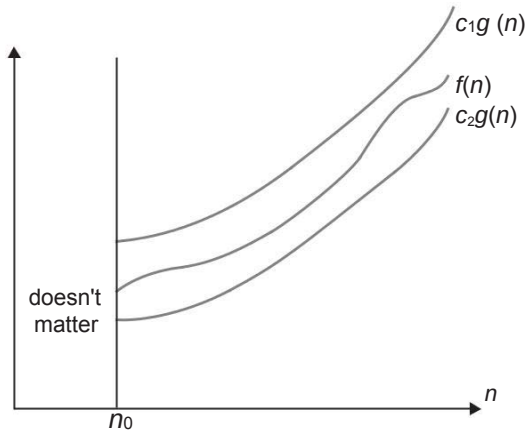

Figure 3: Big-theta notation: $f(n) \in \Theta(g(n))$.

The definition is illustrated in Figure 3.

THEOREM 2. For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

EXAMPLE 3. Prove $(\log\ n!) \in \Theta(n \log n)$?

First Solution: Using Stirling's formula ($n! \approx \sqrt{2\pi n} \left(\begin{array}{c} n \\ e \end{array} \right)^n$)

$$n! = \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \Rightarrow \log n! = \log\left(\frac{n}{e}\right)^n + \log\sqrt{2\pi n}$$

$$\Rightarrow \log n! = n \log\frac{n}{e} + \log\sqrt{2\pi n}$$

$$\Rightarrow \log n! = n \log n - \log e^n + \log\sqrt{2\pi n}$$

$$\Rightarrow \log n! = n \log n + \log\frac{\sqrt{2\pi n}}{e^n}$$

$$\Rightarrow \log n! \in \Theta(n \log n)$$

Second Solution: Here we have to prove that $\log\ n! \in O(n \log n)$ and $\log\ n \in \Omega(n \log n)$

$$\log n! = \log n + \log n - 1 + \cdots + \log 1 \leq \underbrace{\log n + \log n + \cdots + \log n}_{n} \leq \text{n } \log n$$

$$\Rightarrow \log n! \in \Theta(n \log n)$$

$$\log n! = \log n + \log n - 1 + \cdots + \log 1 \geq \frac{n}{2} \log \frac{n}{2}$$

$$\Rightarrow \log n! \geq \frac{n}{2} \log n - \frac{n}{2} \log 2 \geq \frac{1}{4} n \log n \Rightarrow \log n! \in \Omega(n \log n)$$

$$\Rightarrow \log n! \in \Theta(n \log n)$$

Third Solution: Note the following:

$$(n!)^2 = (1 \times 2 \times \ldots \times (n-1)\, n)^2$$

$$\Rightarrow (n!)^2 = (1 \times 2 \times \ldots \times (n-1)\, n)\,(n\,(n-1) \times \ldots 1) = \prod_{x=1}^{n} x\,(n-x+1)$$

$$\begin{cases} y = -x^2 + (n+1)\, x \\ x = \frac{n+1}{2} \Rightarrow y_{max} = \frac{(n+1)^2}{4} \\ x = 1 \Rightarrow y = n,\ x = n \Rightarrow y = n \Rightarrow y_{min} = n \end{cases}$$

$$\prod_{x=1}^{n} n \leq (n!)^2 \leq \prod_{x=1}^{n} \frac{(n+1)^2}{4} \Rightarrow n^n \leq (n!)^2 \leq \left(\frac{n+1}{2}\right)^{2n}$$

$$\Rightarrow n \log n \leq 2 \log n! \leq 2\, n \log \frac{n+1}{2} \leq 2\, n \log n \Rightarrow \frac{n}{2} \log n \leq \log n! \leq n \log n$$

$$\Rightarrow \log n! \in \Theta(n \log n)$$

Let function g(n, m) denote a function with two parameters n and m that can go to infinity independently at different rates. O(g(n, m)) is defined as follows:

$$O(g(n,m)) = \{f(n,m) : \text{ there exist positive constants } c, n_0, \text{ and } m_0$$

$$\text{such that } 0 \leq f(n,m) \leq cg(n,m) \text{ for all } n \geq n_0 \text{ and } m \geq m_0\}.$$

o-notation

DEFINITION The notation o(n), pronounced "Little-O of n" and is defined as follow:

$$o(g(n)) = \{f(n) : \text{ for all constants } c > 0, \text{there exists a constant}$$

$$n_0 > 0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0\}.$$

Little-o notation is related to Big-O notation, which both describe upper bounds, although Little-o is the stronger statement. Therefore, Big-O ($f(n) \in O(g(n))$) means that f(n) asymptotic growth is no faster than g(n), whereas

$f(n) \in o(g(n))$ means that f(n) asymptotic growth is strictly slower than g(n). See the following examples:

$$n^{2.9999} = o(n^3), \quad \frac{n^3}{logn} = o(n^3), \quad n^3 \notin o(n^3), \frac{n^3}{1000} \notin o(n^3).$$

EXAMPLE 4. Does $\log^2(\log n) = o(\log n)$?

All polylogarithmic functions grow more slowly than all positive polynomial functions. It means that for constants a, b > 0, we have

$$log^b n = o(n^a)$$

Substitute log n for n, 2 for b, and 1 for a, gives $\log^2(\log n) = o(\log n)$.

ω-notation

DEFINITION The notation ω(n), pronounced "Little-omega of n" and is defined as follow:

$$f(n) \in \omega(g(n)) = \{ \text{ for any real constant } c > 0, \text{there exists a constant}$$
$$n_0 \geq 1 \text{ such that } f(n) > cg(n) \text{ for every } n \geq n_0\}.$$

Little-ω notation is related to Ω notation, which both describe lower bounds, although Little-ω is the stronger statement. Therefore, $\Omega(f(n) \in \Omega(g(n)))$ means that g(n) asymptotic growth is no faster than f(n), whereas $f(n) \in \omega(g(n))$ means that g(n) asymptotic growth is strictly slower than f(n). See the following examples:

$$n^{3.0001} = \omega(n^3), \quad n^3 logn = \omega(n^3), \quad n^3 \notin \omega(n^3).$$

In this regard, we have the following relationships:

$$\omega(g(n)) \subset \Omega(g(n) - \Theta(g(n)), \quad \omega(g(n)) \subset \Omega(g(n) - O(g(n)).$$

Figure 4 should help you visualize the relationships between these notations. Informally, to summarizes these notations, we have:

- $f(n) = O(g(n)$, if eventually f grows slower than some multiple of g,
- $f(n) = o(g(n)$, if eventually f grows slower than any multiple of g,
- $f(n) = \Omega(g(n)$, if eventually f grows faster than some multiple of g,
- $f(n) = \omega(g(n)$, if eventually f grows faster than any multiple of g,
- $f(n) = \Theta(g(n)$, if eventually f grows grows at the same rate as g,

The asymptotic notations have far more similarities than differences. Table 4 summarizes the key features of these four definitions.

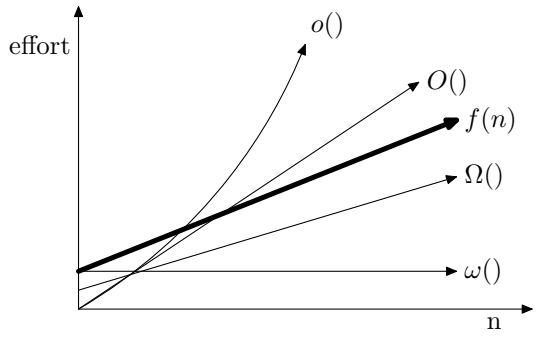

Figure 4: Relationships between asymptotic notations

Table 4: Key features of these four definitions

Definition	? $c > 0$	? $n_0 \geq 1$	$f(n)$? $c.\ g(n)$
$O(n)$	$\exists$	$\exists$	$\leq$
$o(n)$	$\forall$	$\exists$	$<$
$\Omega(n)$	$\exists$	$\exists$	$\geq$
$\omega(n)$	$\forall$	$\exists$	$>$

Comparison of functions

Numerous of the relational properties of real numbers, such as transitivity, reflexivity, symmetry and transpose symmetry, hold for asymptotic notations as well. Let f(n) and g(n) be functions that map positive integers to positive real numbers. We have

Transitivity:

$$
\begin{aligned}
f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) &\Rightarrow f(n) = \Theta(h(n)),\\
f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) &\Rightarrow f(n) = O(h(n)),\\
f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) &\Rightarrow f(n) = \Omega(h(n)),\\
f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) &\Rightarrow f(n) = o(h(n)),\\
f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) &\Rightarrow f(n) = \omega(h(n)).
\end{aligned}
$$

Reflexivity:

$$
\begin{aligned}
f(n) &= \Theta(f(n)),\\
f(n) &= O(f(n)),\\
f(n) &= \Omega(f(n)).
\end{aligned}
$$

Symmetry:

Table 5: Common growth functions

Class	Name	Comments
O(1)	constant	Pronounced "order 1" and denoting a function that runs in constant time; in other words, performance isn't affected by the size of the problem.
O(log n)	logarithmic	Pronounced "order log N" and denoting a function that runs in logarithmic time.
O(log log n)	double logarithmic	-
$O((\log n)^c),\ c > 1$	polylogarithmic	Pronounced "order $(\log n)^c$" and denoting a function that runs in logarithmic time; Note that O(log n) is exactly the same as $O(\log(n^c))$. The logarithms with different bases are equal and vary only by a constant factor so that the big-oh notation ignores that.
O(n)	linear	-
O(n log n)	linearithmic, or loglinear, or quasilinear	Pronounced "order N log N" and denoting a function that runs in time proportional to the size of the problem and the logarithmic time e.g. several divide-and-conquer algorithms.
$O(n^2)$	quadratic	Pronounced "order N squared" and denoting a function that runs in quadratic time; typically, describes the efficiency of algorithms with two nested loops e.g. primary sorting algorithms and some operations on matrices.
$O(n^3)$	cubic	Typically, characterizes efficiency of algorithms with three nested loops.
$O(n^c)$	polynomial	Typically, characterizes efficiency of algorithms with c embedded loops.
$O(2^n)$	exponential	Typical for algorithms that generate all subsets of an n-element set.
$O(c^n),\ c > 1$	exponential	Note that $O(n^c)$ and $O(c^n)$ are very different. The latter grows much, much faster, no matter how big the constant c is. A function that grows faster than any power of n is called superpolynomial. One that grows slower than an exponential function of the form c n is called subexponential.
O(n!)	factorial	Pronounced "order N factorial" and denoting a function that runs in factorial time; typical for algorithms that generate all permutations of an n-element set.

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n)).$$

Transpose symmetry:

$$f(n) = O(g(n)) \text{ if and only if } g(n) = \Omega(f(n)),$$
$$f(n) = o(g(n)) \text{ if and only if } g(n) = \omega(f(n)).$$

Table 5 shows a number of common orders of growth in increasing order, so that most the time efficiencies of a large number of algorithms fall into only these orders of growth.

2 Useful Theorems Involving the Asymptotic Notations

Using the formal description of the asymptotic notations, their general characteristics can be proved. The following theorem, in particular, is useful in

analyzing algorithms that has two consecutively parts.

THEOREM 3. If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then

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

Proof. Since $f_1(n) \in O(g_1(n))$, there exist some positive constant c_1 and some non-negative integer n_1 such that

$$f_1(n) \leq c_1 g_1(n) \text{ for all } n \geq n_1.$$

Similarly, since $f_2(n) \in O(g_2(n))$,

$$f_2(n) \leq c_2 g_2(n) \text{ for all } n \geq n_2.$$

Let us denote $c_3 = \max\{c_1, c_2\}$ and consider $n \geq \max\{n_1, n_2\}$ so that we can use both inequalities. Adding them yields the following:

$$f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n)$$

$$\leq c_3 g_1(n) + c_3 g_2(n) = c_3[g_1(n) + g_2(n)]$$

$$\leq c_3 2 \max\{g_1(n), g_2(n)\}.$$

Hence, $f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})$, with the constants c and n_0 required by the O definition being $2c_3 = 2 \max\{c_1, c_2\}$ and $\max\{n_1, n_2\}$, respectively.

Application of Theorem 3. Suppose that the program P consists of two subprograms P1 which takes time $f(n)$ and P2 which takes time $g(n)$, and P2 is executed after P1. In this case, the complexity of time P, according to this theory, is equal to $max\{f(n), g(n)\}$. Intuitively, the time complexity of an algorithm is equal to the time complexity of a part of the algorithm that has the highest execution time. The theorem also applies to more than two subprograms.

THEOREM 4. If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then

$$f_1(n).f_2(n) \in O(g_1(n), g_2(n)).$$

Proof. Since $f_1(n) \in O(g_1(n))$, there exist some positive constant c_1 and some non-negative integer n_1 such that

$$f_1(n) \leq c_1 g_1(n) \text{ for all } n \geq n_1.$$

Similarly, since $f_2(n) \in O(g_2(n))$,

$$f_2(n) \leq c_2 g_2(n) \text{ for all } n \geq n_2.$$

Let us denote $c = c_1.c_2$ and consider $n_0 = max\{n_1, n_2\}$ so for all $n \geq n_0$ we have:

$$f_1(n).f_2(n) \leq c_1 g_1(n).c_2 g_2(n)$$

$$\leq c(g_1(n).g_2(n))$$

Application of Theorem 4 This theory is used to analyze loops. If there is a loop whose statements have the complexity of time $g(n)$ and this loop is repeated $f(n)$ times, then the time complexity of the whole loop is equal to $f(n).g(n)$.

Lemma 1. Determining that a function f(n) is polynomially bounded is equal to showing that

$$log(f(n)) = O(logn)$$

Proof. This is true because if f(n) be polynomially bounded, then there exist constants c, k, n_0 such that for all $n \geq n_0$, $f(n) \leq cn^k$. Hence, log(f(n)) $\leq$ kclog n, which, since c and k are constants, means that log(f(n)) = O(log n).

EXAMPLE 5. Does $\lceil log\ n \rceil!$ polynomially bounded?

To proof, we use the following two facts:

1. log(n!) = Θ(n log n)(see example 3),
2. $\lceil log\ n \rceil$ = Θ(log n), because

- $\lceil$log n$\rceil \geq$ log n,
- $\lceil$log n$\rceil <$ logn + 1 $\leq$ 2log n for all n $\geq$ 2.

log ($\lceil$log n$\rceil$!) = $\Theta(\lceil$log n$\rceil$log$\lceil$log n$\rceil)$
= Θ(log n loglog n) = ω(log n).

Therefore, log$\lceil$log n$\rceil$! $\notin$ O(log n), and so $\lceil$log n$\rceil$! is not polynomially bounded.

EXAMPLE 6. Does $\lceil loglog\ n \rceil!$ polynomially bounded?

log$\lceil$loglog n$\rceil$! = $\Theta(\lceil$loglog n$\rceil$log$\lceil$loglog n$\rceil)$
= Θ(loglog n logloglog n)
= o((loglog n)^2)
= o(log^{2}log n)
= o((log n)^2) (see exampe 4).

Therefore, log$\lceil$loglog n$\rceil$! = O(log n), and so $\lceil$loglog n$\rceil$! is polynomially bounded.

3 Applying Limits for Analyzing Orders of Growth

Though the formal descriptions of addressed asymptotic notations are necessary for proving the general characteristics of asymptotic functions, they are seldom utilized for comparing the orders of growth of two particular functions. A very practical way to compare the asymptotic behavior of two functions is based on computing the limit of the ratio of those functions. Let f(n) and g(n) denote the two positive functions, three cases may occur:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases}0 & \text{indicates that } f(n) \text{ has a smaller order of growth than } g(n),\\ c & \text{indicates that } f(n) \text{ has the same order of growth as } g(n),\\ \infty & \text{indicates that } f(n) \text{ has a larger order of growth than } g(n).\end{cases}$$

Note that:

- the first case means $f(n) \in o(g(n))$,
- the first two cases mean that $f(n) \in O(g(n))$,
- the second case means that $f(n) \in \Theta(g(n))$,
- the last two cases mean that $f(n) \in \Omega(g(n))$,
- the last case means $f(n) \in \omega(g(n))$.

This technique is often more helpful than the one based on the formal descriptions because it can use the advantage of the powerful calculus methods developed for computing limits, such as L'Hopital's rule

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{f'(n)}{g'(n)}$$

In the following, we give six examples of using the L'Hopital's rule to compare orders of growth of two functions.

EXAMPLE 7. Compare the orders of growth of $f(n) = a^n$ and $g(n) = b^n$, where $b > a > 1$.

$$\lim_{n\to\infty}\frac{a^n}{b^n}=\lim_{n\to\infty}(\frac{a}{b})^n=0.$$

This means that $f(n) \in o(g(n))$.

EXAMPLE 8. Compare the orders of growth of $f(n) = \log_a n$ and $g(n) = \log_b n$.

$$\lim_{n\to\infty}\frac{log_a n}{log_b n}=\lim_{n\to\infty}\frac{(1/n)(log_a e)}{(1/n)(log_b e)}=\frac{log_a e}{log_b e}>0.$$

or

$$\lim_{n\to\infty}\frac{log_a n}{log_b n}=\lim_{n\to\infty}\frac{1/(n\ ln\ a)}{1/(n\ ln\ b)}=\frac{ln\ b}{ln\ a}>0.$$

This means that $f(n)\in\Theta(g(n))$.

EXAMPLE 9. Compare the orders of growth of $f(n)=\frac{1}{10}n(n-1)$ and $g(n)=n^2$.

$$\lim_{n\to\infty}\frac{f'(n)}{g'(n)}=\lim_{n\to\infty}\frac{\frac{1}{10}n(n-1)}{n^2}=\frac{1}{10}\lim_{n\to\infty}\frac{(n^2-n)}{n^2}=\frac{1}{10}\lim_{n\to\infty}(1-\frac{1}{n})=\frac{1}{10}.$$

This means that all three $f(n)\in O(g(n))$, $f(n)\in\Theta(g(n))$ and $f(n)\in\Omega(g(n))$ are correct.

EXAMPLE 10. Compare the orders of growth of $f(n)=log_2 n$ and $g(n)=\sqrt{n}$.

$$\lim_{n\to\infty}\frac{(log_2 n)'}{(\sqrt{n})'}=\lim_{n\to\infty}\frac{(log_2 e)\frac{1}{n}}{\frac{1}{2\sqrt{n}}}=2\,log_2 e\lim_{n\to\infty}\frac{\sqrt{n}}{n}=2\,log_2 e\lim_{n\to\infty}\frac{1}{\sqrt{n}}=0.$$

Since the limit is equal to zero, then $log_2 n$ grows slower than any multiple of $\sqrt{n}$. It means that f(n) has a smaller order of growth than g(n), thus f(n)=o(g(n)).

EXAMPLE 11. Compare the orders of growth of $n!$ and 2^n. Taking advantage of Stirling's formula,
Stirling's formula:

$$n!\approx\sqrt{2\pi n}\left(\begin{array}{c}n\\e\end{array}\right)^n\quad\text{for large values of }\ n.$$

we get

$$\lim_{n\to\infty}\frac{n!}{2^n}=\lim_{n\to\infty}\frac{\sqrt{2\pi n}(\frac{n}{e})^n}{2^n}=\lim_{n\to\infty}\sqrt{2\pi n}\frac{n^n}{2^n e^n}=\lim_{n\to\infty}\sqrt{2\pi n}(\frac{n}{2e})^n=\infty.$$

The following relation can be helpful in comparing the order of growth of some functions together.

$$\frac{f(n)}{g(n)} = 2^{log\ \frac{f(n)}{g(n)}} = 2^{log\ f(n) - log\ g(n)}$$

thus, we have:

$$\lim_{n\to\infty}(log\ f(n) - log\ g(n)) = -\infty \Longrightarrow f(n) \in o(g(n)).$$

$$\lim_{n\to\infty}(log\ f(n) - log\ g(n)) = c > 0 \Longrightarrow f(n) \in \Theta(g(n)).$$

$$\lim_{n\to\infty}(log\ f(n) - log\ g(n)) = +\infty \Longrightarrow f(n) \in \omega(g(n)).$$

EXAMPLE 12. Compare the orders of growth of $f(n) = 2^n$ and $g(n) = 2^{2n}$.

$$\lim_{n\to\infty}(log\ 2^n - log\ 2^{2n}) = \lim_{n\to\infty}(n - 2n) = \lim_{n\to\infty}(-n) = -\infty \Longrightarrow f(n) \in o(g(n)).$$

4 Iterated Function

Let f(n) be a function over the reals, then, the iterates of f are: f(n), f(f(n)), f(f(f(n))), ... For positive integers i, we use the notation $f^{(i)}(n)$ to denote an elegant way to represent repeated operations for function f(n). That is, function f(n) iteratively applied i times to an initial value of n. Iterative fuction is defined as follow:

$$f^{(i)}(n) = \begin{cases} n & i = 0 \\ f(f^{(i-1)}(n) & i > 0 \end{cases}$$

For example, if $f(n) = 10n$, then $f^{(i)}(n) = 10^i n$. The notation $log^*\ n$ (pronounced "log star of n") is used to express the iterated logarithm, which is defined as follows.

$$log^*\ n = min\{i = 0 : log^{(i)} n \leq 1\}.$$

It is important to note that there is difference between $log^{(i)} n$ and $log^i n$. In $log^{(i)} n$, the logarithm function applied i times in succession to an initial value of n, whereas, in $log^i n$, the logarithm function raised to the ith power. The iterated logarithm is a very slowly growing function, for example, see the following iterated logarithms:
$log^*2 = 1$,
$log^*4 = 2$,
$log^*16 = 3$,
$log^*65536 = 4$,
$log^*(2^{65536}) = 5$.

Note that, in the most cases, the input of an algorithm is much less than 2^{65536}, therefore, we rarely encounter an input size n such that $log^* n > 5$.

5 Exercises for Chapter 3: Growth of Functions

5.1 Size of problem

1. Suppose a computer can perform one billion (10^9) basic operations per second, or in other words, each basic operation takes one nanosecond (10^{-9}). Fill in the table below for algorithms with different time complexities and input sizes.

input size/time complexity	log n	n	n^2	2^n
10				
10^2				
10^2				
10^3				

2. In the table below, in each row, a function f(n) is written, which shows the execution time of an algorithm in microseconds. For each time t that appears in the columns and each function f(n), specify the largest size of the problem that the algorithm can solve in time t.

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
log n							
$\sqrt{n}$							
n							
n log n							
n^2							
n^3							
2^n							
n!							

3. An algorithm with time complexity nlogn runs on a computer for 1 second. How long will the same algorithm run on another computer at 100 times faster?

4. A program with input size 10 and time complexity n^2 runs on a computer in 1 millisecond. How long does the same problem with input size 100 run on the same computer?

5. The execution time of a program for input size 50 is equal to 10 seconds. If the time complexity of this program is n^2, the execution time of this program for input size 100 is approximately closer to which of the following times?

a) 10 sec

b) 20 sec

c) 40 sec

d) 100 sec

6. The execution time of a program for input size 1000 is 30 seconds. If its time complexity is n^2, what is the largest input size to solve it in two minutes?

a) 2000

b) 4000

c) 6000

d) 60000

7. The execution time of a program for input sizes 100 and 1000 is 6 seconds and 10 minutes, respectively. Which of the following is most likely the complexity of this program?

a) constant

b) n

c) n^2

d) n^3

5.2 True or False?

8. $n^2 \log n = O(n^3)$

9. $n! = O(n^n)$

10. $n^4 = O(n^3)$

11. $10n^3 + 1000 = O(n^3)$

12. $n \log n = O(n\sqrt{n})$

13. $\sqrt{n} = O(\log n)$

14. $\log n = O(\sqrt{n})$

15. $n^4 + n^3 = O(n^2(10 + n^2))$

16. $n^3(1+2\sqrt{n}) = O(n^3)$

17. $n(1+2\sqrt{n}) = O(n \log n)$

18. $10n^2 + \sqrt{n} = O(n^2)$

19. $10n^2 + 5\sqrt{n} = O(1000n + \sqrt{n} + n\sqrt{n})$

20. $2 \log n + \sqrt{n} = O(n)$

21. $\sqrt{n}\log\ \mathrm{n} = \mathrm{O(n)}$

22. $1/\mathrm{n} = \mathrm{O}(\log\ \mathrm{n})$

23. $\log\ \mathrm{n} = \mathrm{O}(1/\mathrm{n})$

24. $\log\ \mathrm{n} = \mathrm{O}(\mathrm{n}^{-1/2})$

25. $n + 100\sqrt{n} = \mathrm{O}(\sqrt{n}\log n)$

26. $2^{n+1} = O(2^n)$

27. $2^{2n} = O(2^n)$

5.3 Rank the functions

List the following functions according to their order of growth from the lowest to the highest:

28. $O(\log\ n), \quad O(\sqrt{n}), \quad O(n!), \quad O(a^n), \quad O(n)$

29. $n^{1000}\ \ , \quad n!\ \ , \quad (1.005)^n$

30. $O(1+\varepsilon)^n\ \ , \ O(n\ \log\ \ n)\ \ , \ O(\frac{n^2}{\log\ \ n})$

31. $(log\ n)^{log\ n}, \quad \frac{n^2}{log\ n}, \quad 2^{\sqrt{2logn}}, \quad n\sqrt{n}, \quad \sqrt{log(n!)}$

32. $(n-10)!, \ \ 5log(10n+1)^{10}, \ \ 2^{2n}, \ \ 0.0001n^4 + 2n^3 + 1, \ \ ln^2 n, \ \ \sqrt[3]{n}$

33. Compare the following functions according to their order of growth.

$$f(n) = (\log\ n)^{\log\ n}, \quad g(n) = 4^{\log\ n}, \quad h(n) = \log^2 n$$

34. List the following functions according to their order of growth from the lowest to the highest:

$\log(log^* n)$ $\quad 2^{log^* n}$ $\quad (\sqrt{2})^{\log n}$ $\quad n^2$ $\quad n!$ $\quad (\log n\ )!$

$\left(\frac{3}{2}\right)^n$ $\quad n^3$ $\quad log^2 n$ $\quad (\log n!\ )$ $\quad 2^{2^n}$ $\quad n^{1/\log n}$

$\ln\ln n$ $\quad log^* n$ $\quad n2^n$ $\quad n^{\log\log n}$ $\quad \ln n$ $\quad 1$

$2^{\log n}$	$(\log\ n)^{\log\ n}$	e^n	$4^{\log n}$	$(n+1)!$	$\sqrt{\log n}$
$log^*(\log\ n)$	$2^{\sqrt{2\log n}}$	n	2^n	$n\log n$	$2^{2^{n+1}}$

35. Consider the following eighteen functions:

$\ln^2$ n	10000	log ($\log^2$ n)
n logn	n− n^4 + $10n^5$	2n/ log n
n^2	n^4	log n
$n^{1/3}$ + logn	$(\log n)^3$	n!
$\sqrt{n}$	$10n^2$ + log n	2^n
$(2/5)^n$	$(3/5)^n$	n

Group these functions so that f(n) and g(n) are in the same group if and only if f(n)= O(g(n)) and g(n) = O(f(n)), and list the groups in increasing order.

36. Draw a line from each of the five functions in the center to the best big-Ω value on the left, and the best big-O value on the right.

Ω(1/n)		O(1/n)
Ω(1)		O(1)
Ω(log log n)		O(log log n)
Ω($\log^2$ n)		O($\log^2$ n)
Ω(log n)		O(log n)
Ω($\sqrt[3]{n}$)		$O(\sqrt[3]{n})$
Ω(n/ log n)	1/(log n)	O(n/ log n)
Ω(n)	$4n^3 - 2n^2 + 10$	O(n)
Ω($n^{1.00001)}$	(n^2 + n)/($\log^2$ n + log n)	O($n^{1.00001}$)
Ω(n^2/ log n)	$2^{log^2 n}$	O(n^2/ $\log^2$ n)
Ω(n^2/ $\log^2$ n)	4^n	O(n^2/ log n)
Ω(n^2)		O(n^2)
Ω($n^{3/2}$)		O($n^{3/2}$)
Ω(3^n)		O(3^n)
Ω(6^n)		O(6^n)
Ω(n^n)		O(n^n)
Ω(n^{n^2})		O(n^{n^2})

37. Considering the following functions, which choice is the correct?

g_6	g_5	g_4	g_3	g_2	g_1
$2\sqrt{2log n}$	n	2^n	$nlogn$	n^{2+logn}	$log^*(n^n)$

1. $g_1 < g_2 < g_3 < g_4 < g_5 < g_6$
2. $g_6 < g_5 < g_3 < g_1 < g_2 < g_4$
3. $g_1 < g_6 < g_5 < g_3 < g_2 < g_4$

4. $g_6 < g_1 < g_5 < g_3 < g_2 < g_4$

38. Which of the following is the correct?

a) $n^3 \log n = O(n^{3+\varepsilon})$ $0 < \varepsilon < 0.1$

b) $\sqrt{n} = O(\log n)$

c) $n^{1+\varepsilon} = O(\log n)$ $0 < \varepsilon < 0.1$

d) $n^2 = O(\frac{n^2}{\log n})$

39. Considering the following functions, which choice is the correct?

g_6	g_5	g_4	g_3	g_2	g_1
$2^{log^* n}$	$(\sqrt{2})^{logn}$	n^2	$n!$	$logn!$	$2^{2^{n+1}}$

1. $g_6 < g_5 < g_4 < g_3 < g_2 < g_1$
2. $g_5 < g_4 < g_6 < g_2 < g_3 < g_1$
3. $g_5 < g_6 < g_2 < g_4 < g_3 < g_1$
4. $g_6 < g_4 < g_5 < g_2 < g_3 < g_1$

40. Considering the following functions, which of the following relationships shows the order of growth of these functions?

$$f_1(n) = n^2 \ log^2 \ n$$
$$f_2(n) = n^2 \ log^3 \ \sqrt{n}$$
$$f_3(n) = 4^{(logn + \lg logn)}$$

1. $f_1 = f_3 < f_2$
2. $f_2 < f_1 < f_3$
3. $f_1 < f_2 < f_3$
4. $f_1 = f_2 = f_3$

41. Which of the following is the correct?

1. $n^{\frac{1}{10}} \in \Omega(\log n)$
2. $\log(n!) \in O(\log n)$
3. $8n^2 - 3n - 4 \in O(n \log n)$
4. $1^n + 2^n + ... + n^n \in O(n^n)$

5.4 Prove using the definition of notation

For each of the following pairs of functions, find c $\in$ IR$^+$ such that f(n) $\leq$ cg(n) for all n $>$ 1.

42. f(n) $=$ $2n^2 + 4n$, g(n) $=$ n^3

43. f(n) $=$ $5n^2 + n + 1$, g(n) $=$ $2n^2$

44. f(n) $=$ $4n + 5\log n$, g(n) $=$ $n\log n$

45. f(n) $=$ $4 \times 2^n + n^3$, g(n) $=$ $n2^n$

46. f(n) $=$ $7n + 3$, g(n) $=$ $(n^2 - 6n)/2$

47. f(n) $=$ $2\lfloor\sqrt{n}\rfloor - 5$, g(n) $=$ $n - \lceil\sqrt{n}\rceil$

Prove by the Big-oh definition

48. $n^2 + 10\ n\ \in O(n^2)$

49. $5n^2\ \in O(n^2)$

50. $n(n-1)/2\ \in O(n^2)$

51. $n\ \in O(n^2)$

52. $n! + 7n^5\ \in O\left(n^n\right)$

53. $\frac{12n^5}{logn} + 7n^4 \in O(n^5)$

54. n $=$ O(2^n)

55. 100000n + 1 $=$ O(2^n)

56. $n^2 = O(2^n)$

57. $2^n =$ O(n!)

58. $(n + 7)^2 =$ O(n^2)

59. $3n\lfloor\log n\rfloor =$ O(n^2)

60. $log(\lceil loglogn\rceil!) = o(n)$

Prove by the Ω definition

61. $5n^2 \in \Omega(n^2)$

62. $n(n-1)/2 \in \Omega(n^2)$

63. $n^2 + 10\ n \in \Omega(n^2)$

64. $n^3 \in \Omega(n^2)$

65. $2n^3 + 7n^2 \in \Omega\left(n^2\right)$

66. $2n^3 + 7n^2 \in \Omega\left(n^3\right)$

67. $\mathrm{n!} = \Omega(2^n)$

68. $log(\lceil logn \rceil !) = \omega(n)$

Prove by the Θ definition

69. $0.5n^2 + 3n \in \Theta(n^2)$

70. $60n^2 + 5n + 1 = \Theta(n^2)$

71. Does $2n + 3logn = \Theta(n)$? Prove your answer.

72. $\log\ n\,! = \Theta(n \log n)$

73. $\sum_{i=1}^{n} i\ \log i\ =\ \Theta(n^2 \log n)$

74. $7n^2 - 6n + 2 \in \Theta\left(n^2\right)$

75. $n^3 + n^2\ Log\, n \in \Theta\left(n^3\right)$

76. $7n^2 2^n + 5n^2\ Log\, n \in \Theta\left(n^2 2^n\right)$

77. $2n^{2^n} + 72^n \in \Theta\left(n^{2^n}\right)$

78. $\sum_{i=0}^{n} i^3 \in \Theta\left(n^4\right)$

79. $n^4 + 700n^2 \in \Theta\left(n^4\right)$

Without using the notations definition, show whether the following statements are true or not.

80. $n^2 + 10\ n\ \in O(n^2)$

81. $n^3 + 10\ n\ \in O(n^2)$

82. $5n^2\ \in O(n^2)$

83. $n(n-1)/2\ \in O(n^2)$

84. $n\ \in O(n^2)$

85. $n! + 7n^5\ \in O\left(n^n\right)$

86. $\frac{12n^5}{logn} + 7n^4 \in O(n^5)$

87. $\mathrm{n} = \mathrm{O}(2^n)$

88. $100000\mathrm{n} + 1 = \mathrm{O}(2^n)$

89. $\mathrm{n}^2 = O(2^n)$

90. $2^n = \mathrm{O}(\mathrm{n!})$

91. $5n^2\ \in \Omega(n^2)$

92. $n(n-1)/2\ \in \Omega(n^2)$

93. $n^2 + 10\ n\ \in \Omega(n^2)$

94. $n^3 \in \Omega(n^2)$

95. $2n^3 + 7n^2\ \in \Omega\left(n^2\right)$

96. $2n^3 + 7n^2\ \in \Omega\left(n^3\right)$

97. $\mathrm{n!} = \Omega(2^n)$

98. $0.5n^2 + 3n\ \in \Theta(n^2)$

99. $60n^2 + 5n + 1 = \Theta(n^2)$

100. $2n + 3logn = \Theta(n)$

101. $\log\ n\,! = \Theta(n \log n)$

102. $\sum_{i=1}^{n} i\ \log i\ =\ \Theta(n^2 \log n)$

103. $7n^2 - 6n + 2 \in \Theta\left(n^2\right)$

104. $n^3 + n^2\, Log\, n \in \Theta\left(n^3\right)$

105. $7n^2 2^n + 5n^2\, Log\, n \in \Theta\left(n^2 2^n\right)$

106. $2n^{2^n} + 72^n \in \Theta\left(n^{2^n}\right)$

107. $\sum_{i=0}^{n} i^3 \in \Theta\left(n^4\right)$

108. $n^4 + 700n^2 \in \Theta\left(n^4\right)$

Show through computing the limit of the ratio of two functions that (Questions 109-128):

109. $n^2 + 10n \in O(n^2)$

110. $5n^2 \in O(n^2)$

111. $n(n-1)/2 \in O(n^2)$

112. $n(n-1)/2 \in \Omega(n^2)$

113. $n^3 \in \Omega(n^2)$

114. $0.5n^2 + 3n \in \Theta(n^2)$

115. $7n^2 - 6n + 2 \in \Theta\left(n^2\right)$

116. $n^3 + n^2\, Log\, n \in \Theta\left(n^3\right)$

117. $7n^2 2^n + 5n^2\, Log\, n \in \Theta\left(n^2 2^n\right)$

118. $2n^{2^n} + 72^n \in \Theta\left(n^{2^n}\right)$

119. $n^4 + 700n^2 \in \Theta\left(n^4\right)$

120. $\frac{12n^5}{logn} + 7n^4 \in O(n^5)$

121. $2n^3 + 7n^2 \in \Omega\left(n^2\right)$

122. $2n^3 + 7n^2 \in \Omega\left(n^3\right)$

123. $(\mathrm{n} + 1)^2 = \mathrm{O}(\mathrm{n}^2)$

124. $3n \lfloor \log \mathrm{n} \rfloor = \mathrm{O}(\mathrm{n}^2)$

125. $\mathrm{n} = \mathrm{O}(2^n)$

126. $100000\mathrm{n} + 1 = \mathrm{O}(2^n)$

127. $\mathrm{n}^2 = O(2^n)$

128. $2^n = \mathrm{O}(\mathrm{n}!)$

Let the running time, T(n), for a number of algorithms is given below. Prove the correctness of the following phrases.

129. $T(n) = 6n + 4 \in \Omega(n)$

130. $T(n) = 3n + 2 \notin \Omega\left(n^2\right)$

131. $T(n) = 5^n + n^2 \in \Omega\left(2^n\right)$

132. For a > 1 and b > 1, prove the following relation is hold.

$$\log_a^n = \Theta(\log_b^n)$$

133. Prove or disprove?

$$n^{1.001} + n \log n = \Theta(n^{1.001})$$

134. Prove that running time $\mathrm{T(n)} = \mathrm{n}^3 + 20\mathrm{n} + 1$ is $\mathrm{O}(\mathrm{n}^3)$.

135. Prove that running time $\mathrm{T(n)} = \mathrm{n}^3 + 20\mathrm{n} + 1$ is not $\mathrm{O}(\mathrm{n}^2)$.

136. Prove that running time $\mathrm{T(n)} = \mathrm{n}^3 + 20\mathrm{n}$ is $\Omega(\mathrm{n}^2)$.

5.5 Find notations

Let $f(n) = N(g(n))$, where "N" indicates one of O, Ω, and Θ notations. Find for each of the following pairs of functions a "N" notation.

137. $\mathrm{f(n)} = 5\mathrm{n}^2 + 2\mathrm{n} + 10, \quad \mathrm{g(n)} = \mathrm{n}$

138. $\mathrm{f(n)} = \mathrm{n}^3 + 100\mathrm{n}^2 + 40, \quad \mathrm{g(n)} = 10000\mathrm{n} + 20$

139. $\mathrm{f(n)} = 6\frac{\mathrm{n}^3}{\log \mathrm{n}}, \quad \mathrm{g(n)} = \mathrm{n}^3$

140. $\mathrm{f(n)} = 2\sqrt{n} + 10, \quad \mathrm{g(n)} = \log(\mathrm{n} + 10)$

141. $f(n) = n^{2^n} + 102^n, \quad g(n) = n^{2^n}$

142. $f(n) = n^{1.001} + n\log n, \quad g(n) = n^{1.001}$

143. $f(n) = n\sqrt{n} + n, \quad g(n) = n^2 - 10n$

144. $f(n) = n\sqrt{n}, \quad g(n) = 2n \log(n^4 + 4)$

145. $f(n) = (n^{10} + 1000)/(1 + 2^{-n}), \quad g(n) = n^{10} + 3$

146. $f(n) = 0.00001 \times 2^n - n^4, \quad g(n) = 10000n^4 + n^2$

147. $f(n) = 4 \times 2^n + n^2, \quad g(n) = 2^n$

Compare the following pairs of functions. In each case, say whether $f = o(g)$, $f = \omega(g)$, or $f = \Theta(g)$, and prove your claim based on formal definition or limit computation (Questions 148-153).

148. $f(n) = 2n + \log n$, $g(n) = 1000n + (\log n)^{10}$

149. $f(n) = \log n$, $g(n) = \log\log(n^2)$

150. $f(n) = n^3/\log n$, $g(n) = n(\log n)^3$

151. $f(n) = (\log n)^{10^6}$, $g(n) = n^{10^{-6}}$

152. $f(n) = n \log n^2$, $g(n) = (\log n)^{\log n}$

153. $f(n) = n^3 3^n$, $g(n) = 4^n$

Find a theta notation for the following functions.

154. $\frac{(n+1)(n+3)}{n+2}$

155. $2\log n + 4n + 3n\log n$

156. $2 + 4 + 6 + ... + 2n$

157. $\frac{(n^2+\log n)(n+3)}{n+n^2}$

158. $2 + 4 + 8 + 16 + ... + 2^n$

159. $\sqrt{10n^2 + 7n + 3}$

160. $2^{n+1} + 3^{n-1}$

161. $(n^2+1)^{10}$

162. $2\mathrm{nlog(n+2)^2 + (n+2)^2 log\frac{n}{2}}$

5.6 Property of notations

163. Which of the following is correct?

1. $if\ f(n) = O(g(n))\ and\ f(n) = \Theta(g(n))\ then\ f(n) = \Omega(g(n))$
2. $if\ f(n) = \Omega(g(n))\ and\ f(n) = \Theta(g(n))\ then\ f(n) = O(g(n))$
3. $if\ f(n) = \Omega(g(n))\ and\ f(n) = O(g(n))\ then\ f(n) = \Theta(g(n))$
4. $if\ f(n) = O(g(n))\ and\ g(n) = \Omega(f(n))\ then\ f(n) = \Theta(g(n))$

164. Suppose f and g are two arbitrary functions as $f, g : N \to R^+$, as well as suppose that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty$, which option is correct?

1. $f(n) \in O(g(n)), g(n) \notin \Omega(f(n))$
2. $g(n) \in O(f(n)), f(n) \in O(g(n))$
3. $f(n) \in \Omega(g(n)), f(n) \notin \Theta(g(n))$
4. $f(n) \in \Theta(g(n)), g(n) \notin \Omega(f(n))$

Which of the propositions of Questions 165 to 172 is true and which is false? Provide an encounter if the proposition is false.

165. If $f(n) = \Theta(h(n))$ and $g(n) = \Theta(h(n))$ Then $f(n) + g(n) = \Theta(h(n))$

166. If $f(n) = \Theta(g(n))$ Then $cf(n) = \Theta(g(n))$ for each $c \neq 0$

167. If $f(n) = \Theta(g(n))$ Then $2^{f(n)} = \Theta(2^{g(n)})$

168. If $f(n) = \Theta(g(n))$ Then $log\ f(n) = \Theta(log\ g(n))$. (It is assumed that for all $n \geq 1, f(n) \geq 1, g(n) \geq 1$)

169. If $f(n) = O(g(n))$ Then $g(n) = O(f(n))$

170. If $f(n) = O(g(n))$ Then $g(n) = \Omega(f(n))$

171. If $f(n) + g(n) = \Theta(h(n))$ where in $h(n) = max\{f(n), g(n)\}$

172. If $f(n) + g(n) = \Theta(h(n))$ where in $h(n) = min\{f(n), g(n)\}$

173. Find functions f, g, h, and t that apply in the following relationships.

$$f(n) = \Theta(g(n)), \qquad h(n) = \Theta(t(n)),$$
$$f(n) - h(n) \neq \Theta(g(n) - t(n))$$

174. Find functions f and g that apply in the following relationships.

$$f(n) \neq O(g(n)), \qquad g(n) \neq O(f(n))$$

175. Find functions f and g that apply in the following relationships.

$$0 < f(n) < f(n+1), \qquad 0 < g(n) < g(n+1)$$

$$f(n) \neq O(g(n)), \qquad g(n) \neq O(f(n))$$

176. Suppose f(n) = Θ(g(n)). Prove that h(n) = O(f(n)) iff h(n) = O(g(n)).

Suppose $T_1(n) \in \Omega(f(n))$ and $T_2(n) \in \Omega(g(n))$, Prove or disprove?

177. $T_1(n) + T_2(n) \in \Theta(\max\{f(n), g(n)\})$

178. $T_1(n) * T_2(n) \in \Omega(\max\{f(n), g(n)\})$

179. $T_1(n) + T_2(n) \in \Theta(\max\{f(n), g(n)\})$

180. $T_1(n) * T_2(n) \in \Theta(f(n)) * (g(n))$

181. Suppose $f, g : N \to R \geq 0$, prove?

- $f(n) + g(n) = O(\max\{f(n), g(n)\})$
- $f(n) + g(n) = \Theta(\max\{f(n), g(n)\})$
- $f(n) + g(n) = \Omega(\max\{f(n), g(n)\})$

182. Let $T_1(n) = \mathrm{O}(F(n))$ and $T_2(n) = \mathrm{O}(F(n))$, which choice is the correct?

1. $T_1(n) + T_2(n) = \mathrm{O}(F(n))$
2. $T_1(n) / T_2(n) = \mathrm{O}(1)$
3. $T_1(n) * T_2(n) = \mathrm{O}(F(n))$
4. $T_1(n) = \mathrm{O}(T_2(n))$

183. Show if $f(n) = a_m\ n^{\mathrm{m}} + a_{m-1}\ n^{\mathrm{m}-1} + a_{m-2}\ n^{\mathrm{m}-2} + \ldots + a_1\ m_0 + a_0$, then $f(n) = \Omega(n^m)$?

184. Show if $f(n) = a_m\ n^{\mathrm{m}} + a_{m-1}\ n^{\mathrm{m}-1} + a_{m-2}\ n^{\mathrm{m}-2} + \ldots + a_1\ m_0 + a_0$, then $f(n) = \Theta(n^m)$?

Prove the following statements.

185. $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n)) \Rightarrow f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$.

186. $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n)) \Rightarrow f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))$.

187. $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n)) \Rightarrow f_1(n) + f_2(n) = O(\max\{g_1(n), g_2(n)\})$.

188. $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n)) \Rightarrow f_1(n) + f_2(n) = \Omega(\min\{g_1(n), g_2(n)\})$.

189. $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n)) \Rightarrow f_1(n) \times f_2(n) = O(g_1(n) \times g_2(n))$.

190. $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n)) \Rightarrow f_1(n) \times f_2(n) = \Omega(g_1(n) \times g_2(n))$.

191. Suppose that $f_1(n) = \Theta(g_1(n))$ and $f_2(n) = \Theta(g_2(n))$. Justify whether the following statements are true.

- $f_1(n) + f_2(n) = \Theta(g_1(n) + g_2(n))$
- $f_1(n) + f_2(n) = \Theta(\max\{g_1(n), g_2(n)\})$
- $f_1(n) + f_2(n) = \Theta(\min\{g_1(n), g_2(n)\})$

192. Prove or disprove: for all functions f(n) and g(n), either $f(n) = O(g(n))$ or $g(n) = O(f(n))$.

193. Prove or disprove: If $f(n) > 0$ and $g(n) > 0$ for all n, then $O(f(n) + g(n)) = f(n) + O(g(n))$.

194. Prove or disprove?

if $f(n) = \Theta(h(n))$ **and** $g(n) = \Theta(h(n))$ **then** $f(n) + g(n) = \Theta(h(n))$

5.7 More exercises

195. Show

$$2^{\sqrt{2logn}} = \Theta(n^{\sqrt{\frac{2}{logn}}})$$

196. Prove that $cn^k + d = O(2^n)$ for all c, d, k $\in IR^+$.

197. Multiply $2\log n + 10 + O(1/n)$ by $n^2 + O(\sqrt{n})$ and simplify your answer as much as possible.

198. For each of the following functions, indicate how much the function's value will change if its argument is increased fourfold.

1. $log_2 n$
2. $\sqrt{n}$
3. n^2
4. n^3
5. 2^n

199. How many of the following relationships are true?

$$3^{2^n} = \Omega(2^{3^n}), \quad n! = O(n^n), \quad n^{2+\frac{1}{\sqrt{n}}} = o(n^2)$$

1. 0
2. 1
3. 2
4. 3

200. P is a problem whose complexity is $\Omega(n^2)$ and also $O(n^3)$. Algorithm A solves P. Which of the following choices can be complexity of A?

1. $O(n^4)$
2. $o(n^2)$
3. $\Theta(n^3)$
4. $\Theta(n^2)$

201. Prove or disprove:

$$O\left(\left(\frac{n^2}{\log\log n}\right)^{1/2}\right) = O\left(\lfloor\sqrt{n}\rfloor\right)$$

202. Prove or disprove: $2^{(1+O(1/n))^2} = 2+\mathrm{O}(1/\mathrm{n})$.

In questions 203-207, find two functions f(n) and g(n) that satisfy the following relationships. If no such f and g exist, write "None."

203. f(n) = o(g(n)) and f(n) $\neq \Theta$(g(n)).

204. f(n) = Θ(g(n)) and f(n) = o(g(n)).

205. f(n) = Θ(g(n)) and f(n) $\neq$ O(g(n)).

206. f(n) = Ω(g(n)) and f(n) $\neq$ O(g(n)).

207. f(n) = Ω(g(n)) and f(n) $\neq$ o(g(n)).

208. Prove $1 + 2 + 3 + \cdots + n = \Theta(n^2)$?

209. Prove $1^k + 2^k + 3^k + \cdots + n^k = \Theta(n^{k+1})$?

210. Prove $1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} = \Theta(\log\, n)$?

211. Prove $o(f(n)) \subseteq O(f(n))\backslash\Omega(f(n))$?

212. Prove $o(f(n)) \subseteq O(f(n))\backslash\Theta(f(n))$?

213. For any real constants a and b, where $b > 0$, prove:

$$(n + b)^b = \Theta(n^b)$$

214. Explain why the statement, "The running time of algorithm A is at least $O(n^3)$," is meaningless.

215. Compare the following orders of growth.

$$f(n) = n, \quad g(n) = n^{1+sin\, n}.$$

216. Let $f(n)$ and $g(n)$ denote the order of growth two algorithms A and B, respectively; and also we have $g(n) = o(f(n)logn)$. Which of the following is more accurate?

1. For all n, A is faster than B.
2. For all n, B is faster than A.
3. For all n > c, A is faster than B.

4. For all n > c, B is faster than A.

217. Two functions $f(n)$ and $g(n)$ are given, both of which are nonnegative for n. Determine whether each of the following statements is "always correct" or "sometimes correct"?

1. $f(n) + g(n) = \Theta(max f(n), g(n))$
2. If $f(n) = \Omega(n^2)$ and $g(n) = \Omega(n)$ then $f(g(n)) = \Omega(n^3)$
3. $f(g(n)) = \Theta(g(f(n)))$

218. Suppose f(n) = O(g(n)). Which of the following for each function with positive values such as f(n) and g(n) is correct?

1. $g(n) = \Theta(f(n))$
2. $2^{g(n)} = O(2^{f(n)})$
3. $2^{g(n)} = \Omega(2^{f(n)})$
4. log(g(n)) = O(log(f(n))) (for $\log(g(n)) \leq 1$

219. The runtime of the two algorithms A and B are at worst no greater than 150 $n \log n$ and n^2, respectively. Which of the following is correct?

1. For large n, program B is better than A.
2. For large n, program A is better than B.
3. Perhaps program A is easier to implement.
4. For some n, program B is faster than program A.

Find the order of growth of the following sums (Questions 220-230):

220. $\sum_{i=3}^{n+1} 1$

221. $\sum_{i=3}^{n+1} i$

222. $\sum_{i=0}^{n-1} i(i+1)$

223. $\sum_{i=1}^{n} 3^{i+1}$

224. $\sum_{i=1}^{n} \sum_{j=1}^{n} ij$

225. $\sum_{i=1}^{n} \frac{1}{i(i+1)}$

226. $\sum_{i=0}^{n-1} \left(i^2+1\right)^2$

227. $\sum_{i=2}^{n-1} \log i^2$

228. $\sum_{i=1}^{n} (i+1)2^{i-1}$

229. $\sum_{i=1}^{n} (n-i)$

230. $\sum_{i=0}^{n-1} \sum_{j=0}^{i-1} (i+j)$

Explain which of the following statements are true (Questions 231-236).

231. The running time of algorithm A is at least $O(n^2)$.

232. The running time of algorithm A is at best $O(n^2)$.

233. The running time of algorithm A is at worst $O(n^2)$.

234. The running time of algorithm A is at average $\Theta(n^2)$.

235. The running time of algorithm A is at least $\Omega(n^2)$.

236. The running time of algorithm A is at best $\Omega(n^2)$.

237. Does the function $\lceil \sqrt{n} \rceil !$ polynomially bounded?

238. Compare the following functions in terms of asymptotically?

$$log(log^2\ n), \quad log^2(log\ n)$$

239. Compare the following functions in terms of asymptotically?

$$log(log^*\ n), \quad log^*(log\ n)$$

Let

$$t\,(n) = \sum_{i=0}^{d} a_i n^i$$

where $a > 0$, prove the following properties.

240. If $k \geq d$, then $t(n) = O(n^k)$.

241. If $k \leq d$, then $t(n) = \Omega(n^k)$.

242. If $k = d$, then $t(n) = \Theta(n^k)$.

243. If $k > d$, then $t(n) = o(n^k)$.

244. If $k < d$, then $t(n) = \omega(n^k)$.

245. Compare the growth order of the following functions.

$$(\ln(n))^n, \qquad n^{\ln(n)}$$

246. Compare the growth order of the following functions.

$$(\ln(n))!, \qquad n^{\ln\ln ln(n)}$$

247. Compare the growth order of the following functions.

$$e^{c\sqrt{n}}, \qquad e^{\sqrt{n}}$$

248. In the following relations, an error occurred. Find this mistake.

$$n = O(n),\ 2n = O(n),\ 3n = O(n),\ \ldots$$

$$\sum_{k=1}^{n} kn = \sum_{k=1}^{n} O(n) = O(n^2)$$

249. Simplify the following statement.

$$(\ln(n) + O(n^{-1}))(n + O(n^{\frac{1}{2}}))$$

250. Show that for large n we have:

$$1 + \frac{2}{n}O\left(n^{-2}\right) = (1 + \frac{2}{n})(1 + O(n^{-2}))$$

251. Generalize the asymptotic functions to two-variable functions.

252. Suppose that f(n) = $n^{log\ n}$ and g(n) = $(\log n)^n$. Which of the following phrases is true?

$$f(n) = O(g(n))$$
$$f(n) = \Omega(g(n))$$

253. Suppose that f(n) = $n^{\ log\ log\ log\ n}$ and g(n) = (log n)!. Which of the following phrases is true?

$$f(n) = O(g(n))$$
$$f(n) = \Omega(g(n))$$

254. Suppose that f(n) = (n!)! and g(n) = ((n − 1)!)!(n − 1)!$^{n!}$. Which of the following phrases is true?

$$f(n) = O(g(n))$$
$$f(n) = \Omega(g(n))$$

255. The approximate value of the factorial n can be represented as follows:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

which is called sterling number. Using this approximate value, check the correctness of the following relationships:

$$n! = o\left(n^n\right)$$

$$n! = \omega\left(2^n\right)$$

$$log\left(n!\right) = \Theta(nlogn)$$

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^{n+\frac{1}{12n}}$$

256. Let us assume in the table below $k \geq 1, \epsilon > 0$ and $c > 1$. For each row, check the following relationships. Then, write in the house corresponding to each correct relationship, the letter T, and with each false letter F. If there is no relationship, write "None".

$$f\left(n\right) = \Omega(g\left(n\right)), \quad f\left(n\right) = o(g\left(n\right)), \quad f\left(n\right) = \Theta(g\left(n\right)),$$
$$f\left(n\right) = O(g\left(n\right)), \quad f\left(n\right) = \omega(g\left(n\right)).$$

$f(n)$	$g(n)$	O	o	Ω	ω	Θ
$log^k\ n$	n^{ε}					
n^k	c^n					
$\sqrt{n}$	$n^{\sin(n)}$					
2^n	$n^{n/2}$					
$n^{log\ c}$	$c^{log\ n}$					
$\log(n!)$	$log\,(n^n)$					

In the table below, 30 functions $g_1(n), g_2(n), ..., g_{30}(n)$ are written from left to right respectively.

$log(log^* n)$	$(2/3)^n$	n^2	$n!$	$(n+1)!$
$log^*(logn)$	n^3	$\log(n!)$	$(logn)!$	$\ln(n)$
$\ln\ln(n)$	$log^* n$	$n^{loglogn}$	1	2^{2^n}
2^{logn}	$\left(\sqrt{2}\right)^{logn}$	4^{logn}	$\sqrt{logn}$	e^n
$2\sqrt{2logn}$	$logn^2$	2^n	$2^{2^{n+1}}$	n^{2^n}
$(logn)^{logn}$	$2^{log^* n}$	n	$nlogn$	$n^{log^{-1} n}$

Answer questions 257 to 259.

257. List these functions as follows:

$$g_1(n) = \Omega(g_2(n))$$
$$g_2(n) = \Omega(g_3(n))$$
$$g_3(n) = \Omega(g_4(n))$$
$$\vdots$$
$$g_{29}(n) = \Omega(g_{30}(n))$$

258. We define the relation R on this set of functions as follows:

$$g_i\,(n)\,Rg_j\,(n) \Leftrightarrow g_i\,(n) = \Theta(g_i(n))$$

Show that R is an equivalence relation and find the equivalence classes R.

259. Find the function $f(n)$ for all i = 1, 2, ..., 30 values, so that:

$$f\,(n) \neq \mathrm{O}\,(g_i\,(n))$$

$$f\,(n) \neq \Omega\,(g_i\,(n))$$

Assume f(n) and g(n) are two positive functions. Does the following phrases are true? Justfy your answer.

260. $f(n) = O(g(n)) \Rightarrow g(n) = O(f(n))$

261. $f(n) + g(n) = \Theta(\min(f(n),\ g(n)))$

262. $f(n) = o(g(n)) \Rightarrow 2^{f(n)} = o\left(2^{g(n)}\right)$

263. $f(n) = O(g(n)) \Rightarrow 2^{f(n)} = O(2^{g(n)})$

264. $f(n) = O(g(n)) \Rightarrow logf(n) = O(log\ g(n))$

265. $f(n) = O(g(n)) \Rightarrow f(n)/h(n) = O(g(n)/h(n))$

266. $f(n) = O\left((f(n))^2\right)$

267. $f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$

268. $f(n) = \Theta\left(f\left(\frac{n}{2}\right)\right)$

269. $f(n) + o(f(n)) = \Theta(f(n))$

270. $f(n) = O(g(n)) \Rightarrow f(n)^k = O(g(n)^k)$

271. Iterated logarithm is defined as follows:

$$log^{(i)}n = \begin{cases} n, & i = 0 \\ \log\left(lg^{(i-1)}\right) & i \geq 1,\ log^{(i-1)}\ n > 0 \\ \text{undefined} & i < 0 \end{cases}$$

We define:

$$log^*n = \min\left\{i \geq 0 \;\middle|\; log^{(i)}\ n \leq 1\right\}$$

For example, $log^*2^{256} = 5$, because $log^*\ 2^{256} < 1$.
Using this definition, get the following iterated logarithms.

$$log^*2, \quad log^*4, \quad log^*16, \quad log^*65536, \quad log^*2^{65536}$$

272. Assume that f(n) is a positive function, c is a constant and i is a positive integer. We define the functions $f^{(i)}$ and f_c^* as follows:

$$f^{(i)}(n) = \begin{cases} f\left(f^{(i-1)}(n)\right), & i \geq 1 \\ n, & i = 0 \end{cases}$$

$$f_c^*(n) = \min\left\{i \geq 0 | f_{(n)}^{(i)} \leq c\right\}$$

For example, if $f(n) = \frac{n}{2}$, we have $f_2^* = \frac{n}{2^k} \leq 2$. Because:

$$f(n) = \frac{n}{2}, f(f(n)) = f\left(\frac{n}{2}\right) = \frac{n}{4}$$

$$f(f(f(n))) = f\left(f\left(\frac{n}{2}\right)\right) = f\left(\frac{n}{4}\right) = \frac{n}{8}$$

So

$$f^{(k)}(n) = \frac{n}{2^k} \Rightarrow f_2^*(n) = \frac{n}{2^k} \leq 2$$

and for $\frac{n}{k^2} \leq 2$, we must have $n \leq 2^{(k+1)}$ and or $k \geq logn - 1$. For each of the following functions, find $f_c^*(n)$ and write in the table.

$f(n)$	c	$f_c^*(n)$
$\log n$	1	
$n-1$	0	
$n/2$	1	
$\sqrt{n}$	2	
$\sqrt{n}$	1	
$n^{1/3}$	2	
$\frac{n}{\log n}$	2	

Solutions for Chapter 3: Growth of Functions

The following notatations and relations can be helpful in solving the growth of functions problems.

$$a^{-1} = \frac{1}{a}, \qquad (a^m)^n = a^{mn}, \qquad a^m a^n = a^{m+n}.$$

Some notations:

$log\ n = log_2\ n$ (binary logarithm),

$ln\ n = log_e\ n$ (natural logarithm),

$log^k\ n = (log\ n)^k$ (exponentiation),

$log\ log\ n = log(log\ n)$ (composition).

Useful relations for lagarithm function, where $a > 0$, $b > 0$, $c > 0$, and n, are real and logarithm bases are not 1:

$a = b^{log_b\ a}$,

$log_c\ (ab) = log_c\ a + log_c\ b$,

$log_b\ a^n = nlog_b\ a$,

$log_b\ a = \frac{log_c\ a}{log_c\ b}$,

$log_b\ (1/a) = -log_b\ a$,

$log_b\ a = \frac{1}{log_a b}$,

$a^{log_b\ c} = c^{log_b a}$.

For each $b > 1$ and $x > 0$; $log_b n \leq O(n^x)$.

For each $r > 1$ and $d > 0$; $n^d \leq O(r^n)$.

For all $x > 0$, $log_2 x \leq x$.

For example:

1. $(log\ n)^{log\ n} = n^{loglog\ n}$ because $a^{log_b c} = c^{log_b a}$.

2. $4^{log\ n} = n^2$ because $a^{log_b c} = c^{log_b a}$.

3. $2^{log\ n} = n$ because $a^{log_b c} = c^{log_b a}$.

4. $2 = n^{\frac{1}{log\ n}}$, because by applying $log_b\ a = \frac{1}{log_a b}$ to the power $\frac{1}{log\ n}$ and then using $a^{log_b\ c} = c^{log_b a}$.

5. $2^{\sqrt{2log\ n}} = n^{\sqrt{\frac{2}{logn}}}$.

6. $(\sqrt{2})^{log\ n} = \sqrt{n}$, because $(\sqrt{2})^{log\ n} = 2^{\frac{1}{2}logn} = 2^{log\sqrt{n}} = \sqrt{n}$.

7. $log^*(logn) = (log^* n) - 1$.

Solutions