Lecture 1
The Growth of Functions and Big-O Notation

Last Updated: August 26th, 2022

1 Introduction

This lecture considers functions of the type $f : \mathcal{N} \to \mathcal{R}$, i.e. functions whose domain is the set $\mathcal{N} = \{0, 1, 2, \ldots\}$ of natural numbers, and whose codomain is the set $\mathcal{R}$ of real numbers. Using more familiar function notation, we say that $f(n)$ assigns each input natural number n to an output real number $f(n)$. Of primary importance is the long-term growth of such a function, meaning how the function values $f(n)$ increase or decrease for increasingly larger values of n. Most of the functions we consider in this lecture are nonnegative, meaning that $f(n) \geq 0$ for all $n \in \mathcal{N}$.

To describe the growth of a function we use **big-O notation** which includes the symbols $\mathtt{O}$, Ω, Θ, $\mathtt{o}$, and ω. Big-O notation allows us to describe the long-term growth of a function $f(n)$, without concern for either constant multiplicative factors or lower-order additive terms that may appear in the rule describing the function. For example, big-O notation allows us to say that the function $f(n) = 3.5n^2 + 4n + 36$ has growth equal to $\Theta(n^2)$, since n^2 is the dominant term in the algebraic rule describing f, and we ingore the constant 3.5.

Why do we ingore constants and lower-order terms? Consider the following C code.

```
int f(unsigned int n)
{
    int sum = 0; int i,j;

    for(i=0; i < n; i++)
        for(j=0; j < n; j++)
            sum += (i+j)/6;

    return sum;
}
```

Assuming n is a nonnegative integer, let $T(n)$ denote the CPU time required by your favorite laptop to execute the code of $f(n)$ a function of input n. Of course, $T(n)$ will depend on factors that may be beyond our control and understanding, including the C compiler used to compile the code, the hardware of your laptop, the operating system on which it runs, and consideration of other applications that are simultaneously being run by your laptop while it executes program. However, we can say with certainty that the outer `for` loop iterates n times and, for each of those iterations, the inner loop will also iterate n times for a total of n^2 iterations. Moreover, each iteration will require approximately a constant number c of clock cycles to execute, where the number of clock cycles varies with each system. So rather than say "$T(n)$ is approximately cn^2 nanoseconds, for some constant c that will vary from laptop to laptop", we instead use big-O notation and write $T(n) = \Theta(n^2)$ nanoseconds. This conveys that the elapsed CPU time will grow quadratically with respect to n. It is helpful to draw some examples of what the graph of $T(n)$ might look like. Notice that each graph has the shape of a parabola whose equation is roughly cn^2, for some constant $c > 0$.

Therefore, big-O notation allows us to ignore constants that in most cases take on values that are beyond our control, yet they do not affect the "family" to which the function belongs, just as each possible $T(n)$ function belongs to the family of quadratic functions. Moreover, lower-order terms can be made to disappear by bounding them by a constant multiple of the dominant term. For example, $f(n) = 3.5n^2 + 4n + 36 \geq 3.5n^2$ while at the same time

$$f(n) = 3.5n^2 + 4n + 36 \leq 3.5n^2 + 4n^2 + 36n^2 = 43.5n^2,$$

and so $f(n)$ is sandwiched in between two quadratic functions which makes its growth essentially quadratic.

Definition 1.1. The following table shows the most common kinds of growth that are used within big-O notation.

Function	**Type of Growth**
1	constant growth
$\log n$	logarithmic growth
$\log^k n$, for some integer $k \geq 1$	polylogarithmic growth
n^k for some positve $k < 1$	sublinear growth
n	linear growth
$n \log n$	log-linear growth
$n \log^k n$, for some integer $k \geq 1$	polylog-linear growth
$n^j \log^k n$, for some integers $j, k \geq 1$	polylog-polynomial growth
n^2	quadratic growth
n^3	cubic growth
n^k for some integer $k \geq 1$	polynomial growth
$2^{\log^c n}$, for some $c > 1$	quasi-polynomial growth
a^n for some $a > 1$	exponential growth

Definition 1.2. We write $f(n) = \Theta(g(n))$ iff there exists constants $c_1, c_2 > 0$ for which

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

for n sufficiently large.

Note that by writing "n sufficiently large" we mean that the above inequalities do not have to hold for all $n \geq 0$, but rather for all $n \geq k$, for some constant $k \geq 0$. Sometimes we provide an exact value for k, and other times use the "sufficiently large" phrase when the value of k is not important. It is helpful to draw the graphs of two functions f and g for which $f = \Theta(g)$.

Example 1.3. From the above discussion we have $f(n) = 3.5n^2 + 4n + 36 = \Theta(n^2)$ since

$$3.5n^2 \leq f(n) = 3.5n^2 + 4n + 36 \leq 3.5n^2 + 4n^2 + 36n^2 = 43.5n^2,$$

is true for all $n \geq 1$. And so $f(n) = \Theta(n^2)$ by Definition 1.2, where $c_1 = 3.5$ and $c_2 = 43.5$. □

We leave it as an exercise to prove that the big-Θ relation is symmetric, meaning that $f(n) = \Theta(g(n))$ iff $g(n) = \Theta(f(n))$. Because of this we say that $f(n)$ and $g(n)$ have the same **order of growth**, meaning that if we were sorting functions based on how fast (or slow) they grow, then $f(n)$ and $g(n)$ could be placed interchangeably in the sorting.

Also, the following proposition provides a convenient way to show that two functions have the same order of growth.

Proposition 1.4. Let $f(n)$ and $g(n)$ be two nonnegative functions, and suppose there is a constant $c > 0$ for which

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

Then $f(n) = \Theta(g(n))$. Also, if

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

exists but does not equal a positive constant, then $f(n) \neq \Theta(g(n))$.

Proof. By the mathematical definition of limit,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$$

means that, for every $\epsilon > 0$, n can be made sufficiently large so that

$$|\frac{f(n)}{g(n)} - c| < \epsilon$$

always holds true. In words, $\frac{f(n)}{g(n)}$ can be made arbitrarily close to c via increasing the value of n. Removing the absolute-value yields the inequalities

$$c - \epsilon < \frac{f(n)}{g(n)} < c + \epsilon,$$

which implies

$$c_1 \cdot g(n) = (c - \epsilon)g(n) \leq f(n) \leq (c + \epsilon)g(n) = c_2 \cdot g(n).$$

Since $c > 0$ and $\epsilon > 0$ are constants, the latter inequalities imply $f(n) = \Theta(g(n))$ so long as $c_1 = (c - \epsilon) > 0$. Therefore, by choosing $\epsilon = c/2$, we have $c_1 = c - c/2 = c/2 > 0$, which completes the proof.

□

Example 1.5. Use Proposition 1.4 to prove that if $f(n) = 3n^2 + 6n + 7$, then $f(n) = \Theta(n^2)$. In other words, $f(n)$ has quadratic growth.

Example 1.6. Suppose $a > 1$ and $b \neq 0$ are constants, with $|b| < a$. Use Proposition 1.4 to prove that $a^n + b^n = \Theta(a^n)$.

Example 1.7. Verify that $f(n)$ from Example 1.5 does not satisfy $f(n) = \Theta(n^3)$.

2 Little-o and Little-ω

Definition 2.1. Given two functions $f(n)$ and $g(n)$. $f(n) = \mathrm{o}(g(n))$ iff

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

Also, $f(n) = \omega(g(n))$ iff

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

Example 2.2. From Example 1.7, we see that $3n^2 + 6n + 7 = o(n^3)$, and hence $n^3 = \omega(n^2)$. □

Theorem 2.3. The following are all true statements.

1. $1 = o(\log n)$
2. $\log n = o(n^\epsilon)$ for any $\epsilon > 0$
3. $\log^k n = o(n^\epsilon)$ for any $k > 0$ and $\epsilon > 0$
4. $n^a = o(n^b)$ if $a < b$, and $n^a = \Theta(n^b)$ if $a = b$.
5. $n^k = o(2^{\log^c n})$, for all $k > 0$ and $c > 1$.
6. $2^{\log^c n} = o(a^n)$ for all $a, c > 1$.
7. For nonnegative functions $f(n)$ and $g(n)$,
$$(f + g)(n) = \Theta(\max(f, g)(n)).$$
8. If $f(n) = \Theta(h(n))$ and $g(n) = \Theta(k(n))$, then $(fg)(n) = \Theta((hk)(n))$.

Example 2.4. Use the results of Theorems 2.3 to determine a succinct expression that describes the growth of $(fg)(n)$, where $f(n) = n\log(n^4 + 1) + n(\log n)^2$ and $g(n) = n^2 + 2n + 3$.

L'Hospital's Rule. Suppose $f(n)$ and $g(n)$ are both differentiable functions with either

1. $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$, or
2. $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = 0$.

Then

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

Example 2.5. Prove Theorem 1.2 using L'Hospital's Rule.

Big-O and Big-Ω

Suppose we have a function $f(n)$ defined as follows.

$$f(n) = \begin{cases} 2n & \text{if n is even} \\ 3n^2 & \text{if n is odd} \end{cases}$$

What is the growth of $f(n)$? Unfortunately, we can neither say that $f(n)$ has linear growth, nor can we say it has quadratic growth. This is because neither of the limits

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

and

$$\lim_{n\to\infty} \frac{f(n)}{n^2}$$

exist, since infinitely often $f(n)$ jumps from being quadratic to linear, back to quadratic, back to linear, etc.. The best we can do is provide a lower and upper bound for $f(n)$.

Big-O $f(n) = \mathrm{O}(g(n))$ iff $f(n)$ does not grow any faster than $g(n)$. In other words, $f(n) \leq Cg(n)$ for some constant $C > 0$ and for all $n \geq k$, where $k \geq 0$ is a constant.

Big-Ω $f(n) = \Omega(g(n))$ iff $f(n)$ grows at least as fast as $g(n)$. In other words, $f(n) \geq Cg(n)$ for some constant $C > 0$ and for and for all $n \geq k$, where $k \geq 0$ is a constant.

Note: an equivalent way of saying "for all $n \geq k$, where $k \geq 0$ is a constant" is to say "for all n sufficiently large". The latter means that the statement is true once n reaches a threshold (i.e. some value $k \geq 0$) and remains true as n increases past that threshold.

Using these definitions, we see that $f(n) = \mathrm{O}(n^2)$ and $f(n) = \Omega(n)$.

Example 2.6. Suppose function $f(n)$ defined as follows.

$$f(n) = \begin{cases} 2n^{2.1} \log^3 n & \text{if } n \bmod 3 = 0 \\ 10n^2 \log^{50} n & \text{if } n \bmod 3 = 1 \\ 6n^2 \log^{30} n^{80} & \text{if } n \bmod 3 = 2 \end{cases}$$

Provide a big-O upper bound and big-Ω lower bound for $f(n)$.

Example 2.7. Let $T(n)$ denote the CPU time needed to execute the following code as a function of program variable n.

```
//Linear Search for x in array a
Boolean linear_search(int a[ ], int n, int x)
{
    int i;

    for(i=0; i < n; i++)
        if(a[i] == x)
            return true; //found x

    return false; //x is not a member of a[]
}
```

Provide a big-O upper bound and big-Ω lower bound for $T(n)$.

Theorem 2 (Log Ratio Test). Suppose f and g are continuous functions over the interval $[1, \infty)$, and

$$\lim_{n\to\infty} \log(\frac{f(n)}{g(n)}) = \lim_{n\to\infty} \log(f(n)) - \log(g(n)) = L.$$

Then

1. If $L = \infty$ then
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty.$$
2. If $L = -\infty$ then
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$
3. If $L \in (-\infty, \infty)$ is a constant then
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 2^L.$$

Example 2.8. Use the log-ratio test to show that $f(n) = n^{\log n}$ has quasi-polynomial growth.

Solution.

Consider the problem of determining the big-O growth of the series

$$\sum_{i=1}^{\infty} f(i),$$

where f is some nonnegative integer function. In other words, we want to determine the growth of the function $S(n) = f(1) + f(2) + \cdots + f(n)$ which is called the n th **partial sum** of the series.

Example 2.9. Determine the growth of $S(n)$ in case i) $f(i) = i$, ii) $f(i) = i^2$, and iii) $f(i) = i^3$.

Solution.

Unfortunately, the n th partial sum function $S(n)$ for many important series cannot be expressed using a formula. However, the next result shows how we may still determine the big-O growth of $S(n)$, which quite often is our main interest.

Integral Theorem. Let $f(x) > 0$ be an increasing or decreasing Riemann-integrable function over the interval $[1, \infty)$. Then

$$\sum_{i=1}^{n} f(i) = \Theta(\int_1^n f(x)dx),$$

if f is decreasing. Moreover, the same is true if f is increasing, provided $f(n) = \mathrm{O}(\int_1^n f(x)dx)$.

Proof of Integral Theorem. We prove the case when f is decreasing. The case when f is increasing is left as an exercise. The quantity $\int_1^n f(x)dx$ represents the area under the curve of $f(x)$ from 1 to n. Moreover, for $i = 1, \ldots, n-1$, the rectangle R_i whose base is positioned from $x = i$ to $x = i + 1$, and whose height is $f(i + 1)$ lies under the graph. Therefore,

$$\sum_{i=1}^{n-1} \mathrm{Area}(R_i) = \sum_{i=2}^{n} f(i) \leq \int_1^n f(x)dx.$$

Adding $f(1)$ to both sides of the last inequality gives

$$\sum_{i=1}^{n} f(i) \leq \int_1^n f(x)dx + f(1).$$

Now, choosing $C > 0$ so that $f(1) = C \int_1^n f(x)dx$ gives

$$\sum_{i=1}^{n} f(i) \leq (1 + C) \int_1^n f(x)dx,$$

which proves $\sum_{i=1}^{n} f(i) = \mathrm{O}(\int_1^n f(x)dx)$.

Now, for $i = 1, \ldots, n-1$, consider the rectangle R_i' whose base is positioned from $x = i$ to $x = i+1$, and whose height is $f(i)$. This rectangle covers all the area under the graph of f from $x = i$ to $x = i + 1$. Therefore,

$$\sum_{i=1}^{n-1} \mathrm{Area}(R_i') = \sum_{i=1}^{n-1} f(i) \geq \int_1^n f(x)dx.$$

Now adding $f(n)$ to the left side of the last inequality gives

$$\sum_{i=1}^{n} f(i) \geq \int_1^n f(x)dx,$$

which proves $\sum_{i=1}^{n} f(i) = \Omega(\int_1^n f(x)dx)$.

Therefore,

$$\sum_{i=1}^{n} f(i) = \Theta(\int_1^n f(x)dx).$$

Example 2.10. Determine the big-O growth of the series

$$\frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}.$$

Solution.

Exercises

1. An algorithm takes 0.5 seconds to run on an input of size 100. How long will it take to run on an input of size 1000 if the algorithm has a running time that is linear? quadratic? cubic?

2. An algorithm is to be implemented and run on a processor that can execute a single instruction in an average of 10^{-9} seconds. What is the largest problem size that can be solved in one hour by the algorithm on this processor if the number of steps needed to execute the algorithm is n? n^2?, n^3? 1.3^n? Assume n is the input size.

3. Suppose that the Insertion Sort sorting algorithm has a running time of $T(n) = 8n^2$, while the Counting Sort algorithm has a running time of $T(n) = 64n$. Find the largest positive input size for which Insertion Sort runs at least as fast as Counting Sort.

4. If you were given a full week to run your algorithm, which has running time $T(n) = 5 \cdot 10^{-9}(n^3)$ seconds, what would be the largest input size that could be used, and for which your algorithm would terminate after one week? Explain and show work.

5. Use big-O notation to state the growth of $f(n) = n + n \log n^2$. Defend your answer.

6. Which function grows faster: $f(n) = n \log^2 n$ or $g(n) = n^{0.3} \log^{36} n$. Defend your answer

7. Prove that $f(n) = \mathrm{O}(g(n))$ if and only if $g(n) = \Omega(f(n))$.

8. Prove that $(n + a)^b = \Theta(n^b)$, for all real a and $b > 0$. Hint: take a limit.

9. Prove that if $\text{limit}_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = \mathrm{O}(g(n))$, but $g(n) \neq \mathrm{O}(f(n))$.

10. True or false: $2^{n+1} = \Theta(2^n)$? Defend your answer.

11. True or false: $2^{2n} = \Theta(2^n)$? Defend your answer.

12. Use big-O notation to state the growth of the expression

$$\log^{50}(n)n^2 + \log(n^4)n^{2.1} + 1000n^2 + 100000000n.$$

    Defend your answer.

13. True or false: for any two functions f and g, if $f(n) = \Theta(g(n))$, then $2^{f(n)} = \Theta(2^{g(n)})$?

14. If $g(n) = o(f(n))$, then prove that $f(n) + g(n) = \Theta(f(n))$.

15. Use L'Hospital's rule to prove that $a^n = \omega(n^k)$, for every real $a > 1$ and integer $k \geq 1$. Hint: take k derivatives of the ratio a^n/n^k.

16. Prove that $\log_a n = \Theta(\log_b n)$ for all $a, b > 0$. Hint: use change of base formula for logarithms.

17. Suppose function $f(n)$ defined as follows.

$$f(n) = \begin{cases} 4n^{2.3} \log^3 n & \text{if } n \bmod 3 = 0 \\ 2^{\log^{2.2} n} & \text{if } n \bmod 3 = 1 \\ 10n^{2.2} \log^{35} n & \text{if } n \bmod 3 = 2 \end{cases}$$

    Provide a big-O upper bound and big-Ω lower bound for $f(n)$.

18. Let $T(n)$ denote the CPU time needed to execute the following code as a function of program variable n. Hint: assume x is some integer input that is capable of assuming any integer value.

```
    for(i=0; i < n; i++)
    {
        if(x % 2 == 0)
        {
            for(j=0; j < n; j++)
                sum++;
        }
        else
            sum += x;
    }
}
```

    Provide a big-O upper bound and big-Ω lower bound for $T(n)$.

19. Prove that $f(n) = 2^{\sqrt{2 \log n}}$ is $\omega(g(n))$, for any poly-log function $g(n)$, but is $o(g(n))$ for any sub-linear function $g(n)$. Explain and show work.

20. Determine the big-O growth of the function $f(n) = (\log n)^{\log n}$. Explain and show work.

21. Determine the big-O growth of the function $f(n) = (\sqrt{2})^{\log n}$. Explain and show work.

22. Determine the big-O growth of the function $f(n) = n^{1/\log n}$.

23. Use the integral theorem to establish that $1^k + 2^k + \cdots + n^k = \Theta(n^{k+1})$, where $k \geq 1$ is an integer constant.

24. Use the Integeral Theorem to prove that $\log 1 + \log 2 + \cdots + \log n = \Theta(n \log n)$.

25. Show that $\log(n!) = \Theta(n \log n)$.

26. Prove that $n! = \omega(2^n)$.

27. Determine the big-O relationship between 2^{2^n} and $n!$.

28. Determine the big-O growth of the function $f(n) = (\log n)!$.

29. Determine the big-O growth of $n + n/2 + n/3 + \cdots + 1$.

30. Suppose we want to prove that statement $P(n)$ is true for every $n \geq 1$. The **principle of mathematical induction** can establish this using the following two steps. **Basis step**: show $P(1)$ is true. **Inductive step**: assume $P(n)$ is true for *some* $n \geq 1$, and show that this implies $P(n+1)$ is true. The assumption in Step ii is called the **inductive assumption**. Use mathematical induction to prove that

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

31. Use mathematical induction to prove that $1 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$.

32. Use mathematical induction to prove that, for all integers $k \geq 1$ and some $\epsilon > 0$, $\log^k n = o(n^\epsilon)$.

Exercise Hints and Solutions

1. The input size has grown by a factor of 10. For linear time, the running time will also grow by a factor of 10. For quadratic, it will grow by a factor of $10^2 = 100$, to 50 seconds. For cubic, it will grow by a factor of $10^3 = 1000$, to 500 seconds.

2. The elapsed time will be $10^{-9}T(n)$ seconds which must not exceed 3600. Thus we must have $T(n) \leq (3600)(10^9)$, which implies $n = \lfloor T^{-1}((3600)(10^9)) \rfloor$. For example, if $T(n) = n^2$, then $T^{-1}(n) = \sqrt{n}$. In this case $n = \lfloor \sqrt{(3600)(10^9)} \rfloor = 1897366$. In the case when $T(n) = n \log n$, use a graphing calculator to determine the value of n for which $T(n)$ is approximately $(3600)(10^9)$. We must do this because there is no formula for computing T^{-1}.

3. Solve the quadratic equation $8n^2 = 64n$ to get $n = 8$. For $n \geq 9$, Counting Sort will run faster.

4. One week contains $(3600)(24)(7) = 604800$ seconds. Thus, the largest input that can be solved in one week is $n = \lfloor (((604800)(10^9)/5)^{1/3} \rfloor = 49455$. So even though you have an entire week to run the program, the largest problem that can be solved will be in the tens of thousands.

5. We have
$$f(n) = n + n \log n^2 = n + 2n \log n = \Theta(n \log n),$$
since $n = o(n \log n)$.

6. Function $f(n)$ grows faster since
$$\lim_{n \to \infty} \frac{n \log^2 n}{n^{0.3} \log^{36} n} = \lim_{n \to \infty} \frac{n^{0.7}}{\log^{34} n} = \infty$$
since powers of logarithms grow more slowly than power functions such as $n^{0.7}$.

7. Set up the inequality for big-O and divide both sides by C.

8. Take the limit of the ratio of the two functions, and use the fact that, for any two functions, $f^k(n)/g^k(n) = (f(n)/g(n))^k$.

9. Since
$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0,$$
we know that $f(n) \leq g(n)$ for sufficiently large n. Thus, $f(n) = \mathrm{O}(g(n))$.

   Now suppose it was true that $g(n) \leq Cf(n)$ for some constant $C > 0$, and n sufficiently large. Then dividing both sides by $g(n)$ yields $\frac{f(n)}{g(n)} \geq 1/C$ for sufficiently large n. But since
$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0,$$
we know that $\frac{f(n)}{g(n)} < 1/C$, for sufficiently large n, which is a contradiction. Therefore, $g(n) \neq \mathrm{O}(f(n))$.

10. True, since $2^{n+1} = 2 \cdot 2^n$.

11. False. $2^{2n} = 4^n$ and
$$\lim_{n\to\infty} \frac{4^n}{2^n} = \lim_{n\to\infty} 2^n = \infty,$$
and so the two functions have different orders of growth.

12. $\Theta(n^{2.1} \log n)$.

13. False. Consider $f(n) = 2n$ and $g(n) = n$.

14. Use Theorem 1.7 and the fact that $g(n) < f(n)$ for n sufficiently large.

15. Since the derivative (as a function of n) of a^n equals $(\ln a)a^n$, it follows that the k th derivative of a^n divided by the k th derivative of n^k equals $(\ln^k a)a^n/k!$, which tends to infinity. Therefore, $a^n = \omega(n^k)$.

16. By the Change of Base formula,
$$\log_a n = \frac{\log_b n}{\log_b a},$$
and so $\log_a n = C \log_b n$, where $C = 1/\log_b a$. Therefore, $\log_a n = \Theta(\log_b n)$.

17. $f(n) = \Omega(n^{2.2} \log^{35} n)$ and $f(n) = \mathrm{O}(2^{\log^{2.2} n})$.

18. $T(n) = \Omega(n)$ and $T(n) = \mathrm{O}(n^2)$

19. First use the Log-ratio test with $g(n) = \log^k n$, then use it again with $g(n) = n^k$.

20. Superpolynomial but not exponential growth. Use the Log-ratio test against an arbitrary polynomial function n^k. Then use it again against an arbitrary exponential function a^n, where $a > 1$. Or use the fact that $(\log n)^{\log n} < n^{\log n}$, and that $n^{\log n}$ was shown to *not* have exponential growth.

21. Sublinear. Use logarithm identities.

22. Constant growth. Take the logarithm of $f(n)$ and analyze its growth.

23. By the Integral Theorem,
$$\sum_{i=1}^{n} i^k = \Theta(\int_1^n x^k dx) = \Theta(\frac{x^{k+1}}{k+1}\Big|_1^n) = \Theta(n^{k+1}).$$

24. By the Integral Theorem,
$$\sum_{i=1}^{n} \ln i = \Theta(\int_1^n \ln x dx).$$
Moreover,
$$\int_1^n \ln x dx = x \ln x|_1^n - \int_1^n 1 dx =$$
$$n \ln n - n + 1 = \Theta(n \ln n).$$

25. Since $\log ab = \log a + \log b$, we have

$$\log(n!) = \log(n(n-1)(n-2)\cdots 1) = \log n + \log(n-1) + \log(n-2) + \cdots \log 1.$$

Therefore, from the previous exercise, we have $\log(n!) = \Theta(n \log n)$.

26. Use the Log-ratio test, and the fact that $\sum_{i=1}^{n} \log(i) = \Theta(n \log n)$.

27. Use the Log-ratio test to show that the first function is little-ω of the second.

28. Superpolynomial. First analyze the growth of $\log(f(n))$. Then consider the growth of $2^{\log(f(n))}$ which has the same growth as $f(n)$.

29. We have

$$n + n/2 + n/3 + \cdots + 1 = n(1 + 1/2 + 1/3 + \cdots + 1/n).$$

Therefore, by Example 2.10, $n + n/2 + n/3 + \cdots + 1 = \Theta(n \log n)$.

30. **Basis step** $1 = \frac{(1)(2)}{2}$.

Inductive step Assume $1 + 2 + \ldots n = \frac{n(n+1)}{2}$ for some $n \geq 1$. Show:

$$1 + 2 + \ldots n + (n+1) = \frac{(n+1)(n+2)}{2}.$$

$$1 + 2 + \ldots n + (n+1) = \frac{n(n+1)}{2} + (n+1) = (n+1)(\frac{n}{2} + 1) = \frac{(n+1)(n+2)}{2},$$

where the first equality is due to the inductive assumption.

31. **Basis step** $1 = \frac{(1)(2)(3)}{6}$.

Inductive step Assume $1^2 + 2^2 + \ldots n^2 = \frac{n(n+1)(2n+1)}{6}$ for some $n \geq 1$. Show:

$$1^2 + 2^2 + \ldots n^2 + (n+1)^2 = \frac{(n+1)(n+2)(2n+3)}{6}.$$

$$1^2 + 2^2 + \ldots n^2 + (n+1)^2 = \frac{n(n+1)(2n+1}{6} + (n+1)^2 =$$

$$\frac{(n+1)}{6}(n(2n+1) + 6(n+1)) = \frac{(n+1)}{6}(2n^2 + 7n + 6) = \frac{(n+1)}{6}(n+2)(2n+3) =$$

$$\frac{(n+1)(n+2)(2n+3)}{6},$$

where the first equality is due to the inductive assumption.

32. **Basis step** $\log^1 n = o(n^\epsilon)$ by Example 2.5.

Inductive step Assume $\lim_{n\to\infty} \frac{\log^k(n)}{n^\epsilon} = 0$ for some $k \geq 1$. Show:

$$\lim_{n\to\infty} \frac{\log^{k+1}(n)}{n^\epsilon} = 0.$$

For simplicity, we assume that $\log(n)$ has base e, so that $(\log n)' = \frac{1}{n}$. This will not affect the big-O growth of the ratio, since, by the change of base formula,

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

and so logarithms with different bases only differ by a constant factor, which does not affect big-O growth. Then by L'Hospital's Rule,

$$\lim_{n\to\infty} \frac{\log^{k+1}(n)}{n^\epsilon} = \lim_{n\to\infty} \frac{(\log^{k+1}(n))'}{(n^\epsilon)'} =$$

$$\lim_{n\to\infty} \frac{\log^k(n)}{n\epsilon \cdot n^{\epsilon-1}} = \frac{1}{\epsilon} \lim_{n\to\infty} \frac{\log^k(n)}{n^\epsilon} = 0,$$

where the last equality is due to the inductive assumption.