1 Big-Oh Notation

1.1 Definition

Big-Oh notation is a way to describe the rate of growth of functions. In CS, we use it to describe properties of algorithms (number of steps to compute or amount of memory required) as the size of the inputs to the algorithm increase. The idea is that we want something that is impervious to constant factors; for example, if your new laptop is twice as fast as your old one, you don't want to have to redo all your analysis.

$f(n)$ is $O(g(n))$	if there exist c, N such that $f(n) \leq c \cdot g(n)$ for all $n \geq N$.	f "$\leq$" g
$f(n)$ is $o(g(n))$	if $\lim_{n\to\infty} f(n)/g(n) = 0$.	f "$<$" g
$f(n)$ is $\Theta(g(n))$	if $f(n)$ is $O(g(n))$ and $g(n)$ is $O(f(n))$.	f "$=$" g
$f(n)$ is $\Omega(g(n))$	if there exist c, N such that $f(n) \geq c \cdot g(n)$ for all $n \geq N$.	f "$\geq$" g
$f(n)$ is $\omega(g(n))$	if $\lim_{n\to\infty} g(n)/f(n) = 0$.	f "$>$" g

Note that in the definition above, all we care about is the **long term** behavior of f versus g. That is why in the capital letters O and Ω, we have $n \geq N$, and in the lowercase letters o and ω, we have are taking the limit as $n \to \infty$.

Keep in mind that $o \Rightarrow O$ and $\omega \Rightarrow \Omega$.

1.2 Notes on notation

When we use asymptotic notation within an expression, the asymptotic notation is shorthand for an unspecified function satisfying the relation:

- $n^{O(1)}$ means $n^{f(n)}$ for some function $f(n)$ such that $f(n) = O(1)$.
- $n^2 + \Omega(n)$ means $n^2 + g(n)$ for some function $g(n)$ such that $g(n) = \Omega(n)$.
- $2^{(1-o(1))n}$ means $2^{(1-\epsilon(n))\cdot g(n)}$ for some function $\epsilon(n)$ such that $\epsilon(n) \to 0$ as $n \to \infty$.

When we use asymptotic notation on both sides of an equation, it means that for all choices of the unspecified functions in the left-hand side, we get a valid asymptotic relation:

- $n^2/2 + O(n) = \Omega(n^2)$ because for all f such that $f(n) = O(n)$, we have $n^2/2 + f(n) = \Omega(n^2)$.
- But it is not true that $\Omega(n^2) = n^2/2 + O(n)$ (e.g. $n^2 \neq n^2/2 + O(n)$).

The above is quite subtle, but it is important to think of the equal sign as more of a "belongs to the family of functions described by the right hand side" rather than a strict equality.

Exercise 1. Using Big-Oh notation, describe the rate of growth of each of the following:

- n^{124} vs. 1.24^n
- $\sqrt[124]{n}$ vs. $(\log n)^{124}$
- $n \log n$ vs. $n^{\log n}$
- $\sqrt{n}$ vs. $2^{\sqrt{\log n}}$
- $\sqrt{n}$ vs. $n^{\sin n}$

Exercise 2. Which of the following hold?

- $f(n) = \omega(2^n)$ implies that $f(n) = \Omega(n^2)$
- If $g(n) = o(f(n))$, then $f(n) + g(n) = \Theta(f(n))$ (You can assume f and g are positive functions)

Exercise 3. Give a counterexample to the following statement. Then propose an easy fix.

If $f(n)$ and $g(n)$ are positive functions, then $f(n) + g(n) = \Theta(\min(f(n), g(n))$

Exercise 4. (Challenge) Let $f(n) = 1^k + 2^k + \ldots + n^k$ for some constant $k \in \mathbb{N}$; find a 'simple' function $g : \mathbb{N} \to \mathbb{N}$ such that $f(n) = \Theta(g(n))$. Find a constant c such that $\frac{f(n)}{cg(n)} \to 1$ as $n \to \infty$.

2 Recurrence relations

2.1 General guidelines

There is no single best method for solving recurrences. There's a lot of guesswork and intuition involved. That being said, here are some tips that will help you out most of the time:

- **Guess**! You can get surprisingly far by comparing recurrences to other recurrences you've seen, and by following your gut. Often quadratic, exponential, and logarithmic recurrences are easy to eyeball.
- **Graph it**. Try plugging in some values and graphing the result, alongside some familiar functions. This will often tell you the form of the solution.
- **Substitute**. If a recurrence looks particularly tricky, try substituting various parts of the expression until it begins to look familiar.
- **Solve for constants**. Using the previous methods, you can often determine the form of the solution, but not a specific function. However, there is often enough information in the recurrence to solve for the remainder of the information. For instance, let's say a recurrence looked quadratic. By substituting $T(n) = an^2 + bn + c$ for the recurrence and solving for a, b, and c, you'll likely have enough information to write the exact function.

Exercise 5. Solve the following recurrence relations, assuming that $T(n) = 1$ for $n \leq 1$.

- $T(n) = T(n/3) + 2$
- $T(n) = 3T(n/4)$
- $T(n) = T(n-1) + n^2$

2.2 The Master Theorem

The previous section stressed methods for finding the exact form of a recurrence, but usually when analyzing algorithms, we care more about the asymptotic form and aren't too picky about being exact. For example, if we have a recurrence $T(n) = n^2 \log n + 4n + 3\sqrt{n} + 2$, what matters most is that $T(n)$ is $\Theta(n^2 \log n)$. In cases like these, if you're only interested in proving Θ bounds on a recurrence, the Master Theorem is very useful. The solution to the recurrence relation

$$T(n) = aT(n/b) + cn^k$$

where $a \geq 1$, $b \geq 2$ are integers, and c and k are positive constants, satisfies

$$T(n) \text{ is } \begin{cases} \Theta(n^{\log_b a}) & \text{if } a > b^k \\ \Theta(n^k \log n) & \text{if } a = b^k \\ \Theta(n^k) & \text{if } a < b^k \end{cases}$$

In general, the idea is that the relationship between a and b^k determines which term of the recurrence dominates; you can see this if you expand out a few layers of the recurrence. For more information about the proof, see the supplement posted with the section notes.

Exercise 6. Use the Master Theorem to solve $T(n) = 4T(n/9) + 7\sqrt{n}$.

Exercise 7. How might you remember the above cases if you forgot them during an exam?

Exercise 8. For each of the following algorithms, come up with a recurrence relationship in the form of Master Theorem and solve it. Verify that your answer makes sense.

- Mergesort
- Binary Search
- Finding the maximum element of a list by recursively finding the maximum element of the left and right halves and then taking the max of those two.