Asymptotic Analysis

Big-O

$$O(g(n)) = \{f(n) | \exists c > 0, n_0 > 0$$
$$(\forall n > n_0 (0 \leq f(n) \leq cg(n)))\}$$

"***Big -Oh***" of some function $g(n)$ is the set of all functions $f(n)$, where there exists a positive constant c such that $0 \leq f(n) \leq cg(n)$ for sufficiently large values of n (for values of n above some positive value n_0 ($n \geq n_0 \geq 0$)).

Big-O

$$0 \leq f(n) \leq cg(n)$$

Big-O

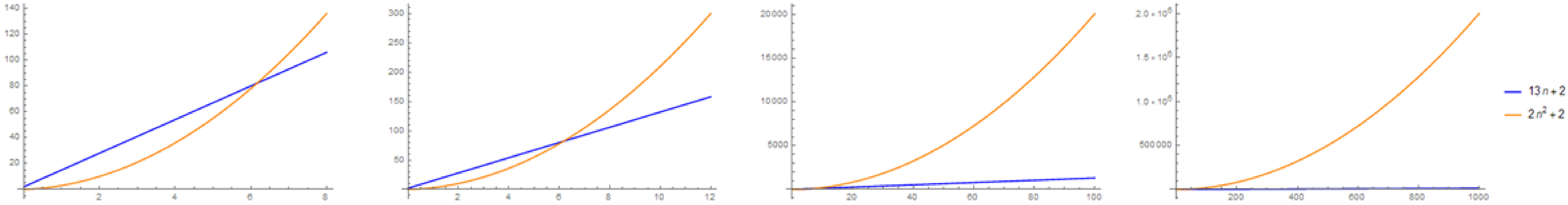
this inequality must hold true for all values n that reach a threshold n_0

Big-O

all images $f(n)$ must be not be greater than the corresponding image $g(n)$ times c

Big-O

$$13n + 2 \in O(2n^2 + n)$$


140
120
100
80
60
40
20
2
4
6
8
300
250
200
150
100
50
2
4
6
8
10
12
20000
15000
10000
5000
20
40
60
80
100
2.0 × 10^6
1.5 × 10^6
1.0 × 10^6
500 000
200
400
600
800
1000
13n + 2
2n^2 + 2

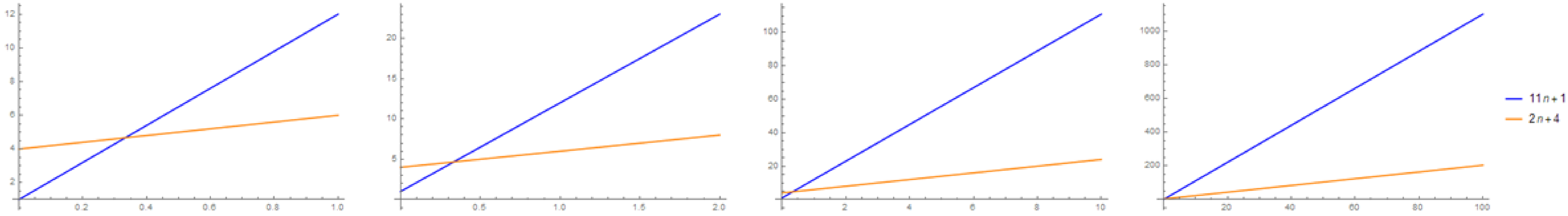
Big-O

Big-O

- As shown in the plot, even though the function $13n + 2$ starts bigger than $2n^2 + n$, after some point, $2n^2 + n$ **overtakes** the function $13n + 2$ and leaves it behind so much that, $13n + 2$ is only barely visible on the last plot
- **there exists an n_0, a threshold for n, where, we are sure that for any n greater than n_0, the inequality $13n + 2 \leq 2n^2 + n$ is always true**

Big-O

$$11n + 1 \in O(2n + 4)$$

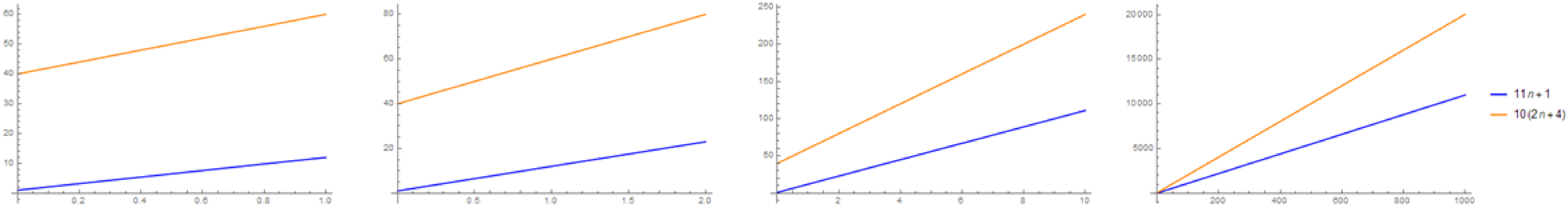


12
10
8
6
4
2
0.2
0.4
0.6
0.8
1.0
20
15
10
5
0.5
1.0
1.5
2.0
100
80
60
40
20
2
4
6
8
10
1000
800
600
400
200
20
40
60
80
100
11 n + 1
2 n + 4

Big-O

Big-O

$2n+4$, starts bigger, but then $11n+1$ overtakes it around $n=0.4$


60
50
40
30
20
10
0.2
0.4
0.6
0.8
1.0
80
60
40
20
0.5
1.0
1.5
2.0
250
200
150
100
50
2
4
6
8
10
20 000
15 000
10 000
5000
200
400
600
800
1000
11n + 1
10(2n + 4)

Big-O

Big-O

or $c = 10$, the function $10(2n + 4)$, starts as bigger than $2n + 4$ and is always bigger than $2n + 4$

Big-O

- $13n+2$ **is upper bounded by** $O(2n^2+n)$ - it's called an upper bound because as n increases to large values, $13n+2$ can never be greater than $2n^2+n$. There is a boundary above $13n+2$ and that boundary is $2n^2+n$.

Big-O

- $13n + 2$ **cannot be more complex than** $2n^2 + n$ - the term complexity is often used when contextualized with programming. When dealing with programming resources such as time, memory, and etc, CS often abstracts these resources as complexities (i.e. time complexity, space complexity). This will be discussed in detail later.

When we say $f(n)$ cannot be more complex than $g(n)$, it means that it is still possible that $f(n)$ is equally as complex as $g(n)$,

Big-Ω

$$\Omega(g(n)) = \{f(n) | \exists c > 0, n_0 > 0$$
$$(\forall n > n_0 (0 \leq cg(n) \leq f(n)))\}$$

"***Big -Omega***" of some function $g(n)$ is the set of all functions $f(n)$, where there exists a positive constant c such that $0 \leq cg(n) \leq f(n)$ for sufficiently large values of n (for values of n above some positive value n_0 ($n \geq n_0 \geq 0$)).

Big-Ω

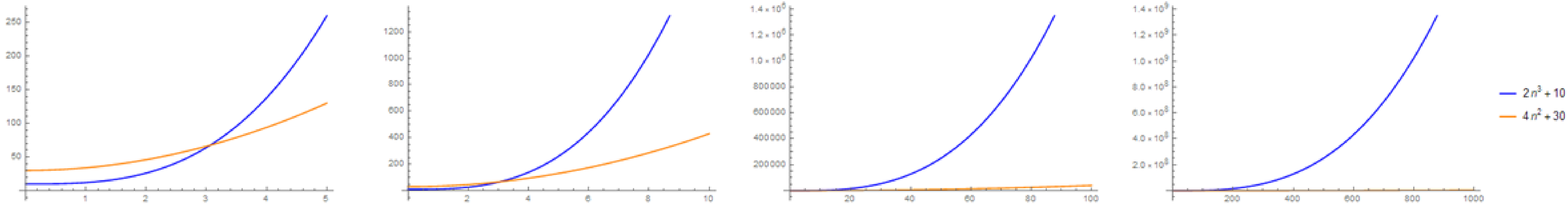
$$0 \leq cg(n) \leq f(n)$$

Big-Ω

$cg(n)$ **is now less than or equal to** $f(n)$

Big-Ω

$$2n^3 + 10 \in \Omega(4n^2 + 30)$$


250
200
150
100
50
1
2
3
4
5
1200
1000
800
600
400
200
2
4
6
8
10
1.4×10^6
1.2×10^6
1.0×10^6
800000
600000
400000
200000
20
40
60
80
100
1.4×10^9
1.2×10^9
1.0×10^9
8.0×10^8
6.0×10^8
4.0×10^8
2.0×10^8
200
400
600
800
1000
$2n^3 + 10$
$4n^2 + 30$

Big-Ω

Big-Ω

- $2n^3 + 10$ **is lower bounded by** $4n^2 + 30$.
- $2n^3 + 10$ **cannot be less complex than** $4n^2 + 30$.

When we say $f(n)$ cannot be less complex than $g(n)$, it means that it is still possible that $f(n)$ is equally as complex as $g(n)$,

Big-θ

$$\Theta(g(n)) = \{f(n) | \exists c_1 > 0, c_2 > 0, n_0 > 0$$
$$(\forall n > n_0 (0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)))\}$$

"***Big -Theta***" of some function $g(n)$ is the set of all functions $f(n)$, where there exists a positive constant c_1 and c_2 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for sufficiently large values of n (for values of n above some positive value n_0 $(n \geq n_0 \geq 0)$).

Big-θ

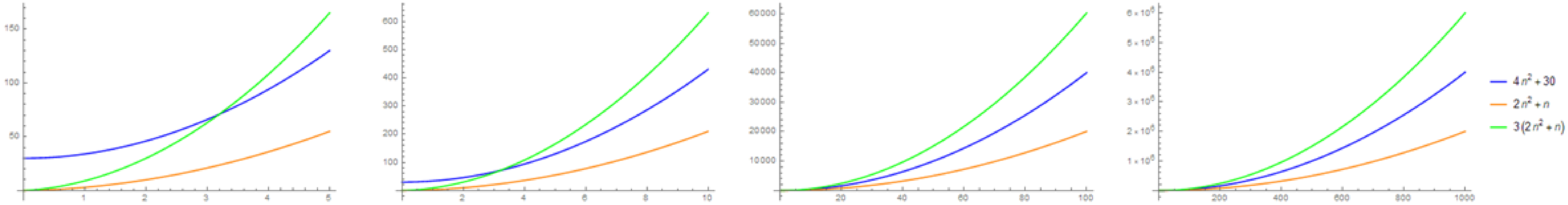
the inequality conditions for Big-O and Big-Ω are combined into one

Big-θ

$$4n^2 + 30 \in \Theta(2n^2 + n)$$

Big-θ

$$c_1(2n^2 + n) \leq 4n^2 + 30 \leq c_2(2n^2 + n)$$



150
100
50
1
2
3
4
5
600
500
400
300
200
100
2
4
6
8
10
60000
50000
40000
30000
20000
10000
20
40
60
80
100
6×10^6
5×10^6
4×10^6
3×10^6
2×10^6
1×10^6
200
400
600
800
1000
$4n^2+30$
$2n^2+n$
$3(2n^2+n)$

Big-θ

Big-θ

- $4n^2 + 30$ is tightly bounded by $2n^2 + n$.
- $4n^2 + 30$ is as complex as $2n^2 + n$.

Before we move on the next section, here are some questions to ponder upon: (don't worry, these questions will be answered and proven in the next sections)

- **Is it true that $f(n) \in O(g(n)) \cup \Omega(g(n))$ for any pair of functions $f(n)$ and $g(n)$?** *(I.e. Is $f(n)$ guaranteed to be a member of either $O(g(n))$ or $\Omega(g(n))$?)*
- **Is it true that if $f(n) \in O(g(n))$, then $g(n) \in \Omega(f(n))$?**
- **What is $O(g(n)) \cap \Omega(g(n))$?** *(What is the intersection of $O(g(n))$ and $\Omega(g(n))$?)*

Proving Asymptotic Relationships using the Formal Definition

is $n^2 + 5 \in O(n^2)$?

Proving Asymptotic Relationships using the Formal Definition

$$2n^3 + 10 \in \Omega(4n^2 + 30)$$

Proving Asymptotic Relationships using the Formal Definition

$$\Omega(g(n)) = \{f(n) | \exists c > 0, n_0 > 0$$
$$(\forall n > n_0 (0 \leq cg(n) \leq f(n)))\}$$

Proving Asymptotic Relationships using the Formal Definition

$$\exists c > 0, n_0 > 0$$
$$(\forall n > n_0 (0 \leq c(4n^2 + 30) \leq 2n^3 + 10)$$

Proving Asymptotic Relationships using the Formal Definition

$$\text{let } c = 1, n_0 = 3.4$$

Proving Asymptotic Relationships using the Formal Definition

$$4n^2 + 30 \leq 2n^3 + 10$$

$$4n^2 + 20 \leq 2n^3$$

$$2 + \frac{10}{n^2} \leq n$$

Proving Asymptotic Relationships using the Formal Definition

[^1]: In fact this is connected to how limit definitions work (which will be discussed in the next section) and why this whole concept is called **asymptotic** analysis.

Proving non-membership

What if we needed to prove that a function is not a member of O, Ω, or Θ

Proving non-membership

$$2n^3 + 10 \notin O(4n^2 + 30)$$

Proving non-membership

$$\neg(\exists c > 0, n_0 > 0$$

$$(\forall n > n_0(0 \leq (2n^3 + 10) \leq c(4n^2 + 30))))$$

$$(\forall c > 0, n_0 > 0$$

$$(\exists n > n_0(0 > (2n^3 + 10) \vee (2n^3 + 10) > c(4n^2 + 30)))$$

note: we do not actually care for cases where $n < 0$ or $f(n) < 0$ or in fact anything that is less than 0. In the context that we are using this in computer science, values are cannot be negative.

Proving non-membership

- it is not enough to find a few values for c and n_0
- prove that for all possible combinations of c and n_0, sometimes there is a value for n that is greater than n_0, that satisfies $(2n^3 + 10) > c(4n^2 + 30)$.

Proving non-membership

$$\begin{aligned} 2n^3 + 10 &> c(4n^2 + 30) \\ 2n^3 + 10 &> 4n^2c + 30c \\ n + \frac{10}{2n^2} &> 2c + \frac{30c}{2n^2} \\ n &> 2c + \frac{30c}{2n^2} - \frac{10}{2n^2} \end{aligned}$$

Proving non-membership

Now, **no matter what combination of c and n_0, you choose, there will always be a value of n that satisfies this because the left hand side (which is also n) increases as n increases.** On the other hand, **the right hand side decreases as n increases**

Limit Definitions

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

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0 \to f(n) \in \Omega(g(n))$$

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \to g(n) \in \Theta(f(n))$$

if the limit of the ratio $\frac{f(n)}{g(n)}$ as n approaches infinity...

Limit Definitions

we're trying to see what happens to the value of that ratio as n becomes very very large (as n approaches infinity).

Approaching Infinity

$$f(n) = 2n^3 + 10$$

$$g(n) = 4n^2 + 30$$

Approaching Infinity

Let $n = 5$

$$\frac{2(5)^3 + 10}{4(5)^2 + 30} = 5$$

Approaching Infinity

But as we **increase** the value of n, we'll notice that the **value of the ratio becomes bigger and bigger**:

$$\frac{2(100)^3 + 10}{4(100)^2 + 30} \approx 50$$

Let $n = 1000$

$$\frac{2(1000)^3 + 10}{4(1000)^2 + 30} \approx 500$$

Let $n = 100000$

Approaching Infinity

When we reach **really** big values of n, (around infinity), **we can expect that that number will be really big** as well.

Approaching Infinity

the numerator of the ratio will increase much faster than the denominator

Approaching Infinity

$$f(n) = 13n + 2$$

$$g(n) = 2n^2 + n$$

$$\frac{13(100)+2}{2(100)^2+100} \approx 0.0647$$

Let $n = 10000$

$$\frac{13(10000)+2}{2(10000)^2+10000} \approx 0.000649$$

Let $n = 10000000$

Approaching Infinity

We directly look at the value of the ratio when the value of n approaches to **infinity** (a very big value).

Approaching Infinity

$$\lim_{n \to \infty} \frac{2n^3 + 10}{4n^2 + 30}$$

Approaching Infinity

$$= \frac{2(\infty)^3 + 10}{4(\infty)^2 + 30}$$

$$= \frac{\infty}{\infty}$$

Approaching Infinity

The fraction infinity over infinity is an undefined value, so instead of automatically substituting infinity to n, we'll need to apply **L'Hopital's rule** first:

Approaching Infinity

$$\begin{aligned}\lim_{n\to\infty}\frac{2n^3+10}{4n^2+30} &= \lim_{n\to\infty}\frac{\frac{d}{dx}(2n^3+10)}{\frac{d}{dx}(4n^2+30)}\\ &= \lim_{n\to\infty}\frac{6n^2}{8n}\\ &= \lim_{n\to\infty}\frac{3n}{4}\end{aligned}$$

Approaching Infinity

$$\lim_{n\to\infty} \frac{2n^3 + 10}{4n^2 + 30} = \lim_{n\to\infty} \frac{3n}{4} = \infty$$

If you don't know what L'Hopital's rule is, its basically a technique for solving otherwise undefined/indeterminate limit ratio values. The exact usage of L'Hopital's rule is as follows:

$$\lim_{n\to c}\frac{f(n)}{g(n)} = \lim_{n\to c}\frac{\frac{d}{dx}f(n)}{\frac{d}{dx}g(n)}$$

Basically, if you are unable to solve limits (because you end up with undefined/indeterminate values like $\frac{0}{0}$, $\frac{\infty}{\infty}$), you derive both the denominator and numerator separately. If after applying L'Hopital's rule, you still end up with undefined/indeterminate values, you can apply it again, until you get an answer that is either 0, some positive constant, or infinity.

Approaching Infinity

$$\begin{aligned}\lim_{n\to\infty}\frac{13n+2}{2n^2+n} &= \lim_{n\to\infty}\frac{\frac{d}{dx}(13n+2)}{\frac{d}{dx}(2n^2+n)}\\ &= \lim_{n\to\infty}\frac{13}{4n+1}\\ &= 0\end{aligned}$$

$$\therefore 13n+2 \in O(2n^2+n)$$

$$\begin{aligned}
\lim_{n\to\infty} \frac{\quad}{6n^3+1} &= \lim_{n\to\infty} \frac{\quad}{\frac{d}{dx}(6n^3+1)} \\
&= \lim_{n\to\infty} \frac{12n^2+6n}{18n^2} \\
&= \lim_{n\to\infty} \frac{\frac{d}{dx}(12n^2+6n)}{\frac{d}{dx}(18n^2)} \\
&= \lim_{n\to\infty} \frac{24n+6}{36n} \\
&= \lim_{n\to\infty} \frac{\frac{d}{dx}(24n+6)}{\frac{d}{dx}(36n)} \\
&= \lim_{n\to\infty} \frac{24}{36}
\end{aligned}$$

$$\lim \frac{4n^3+3n^2}{\quad} = 46$$

Note how L'Hopital's rule had to be applied thrice to solve this.

Also, if we end up with a limit value between 0 and infinity (meaning any positive constant), This means that neither the numerator or denominator is relatively bigger than the other. In cases like these, the numerator is a member of big theta of the denominator.

$$
\begin{aligned}
\lim_{n\to\infty} \frac{n \log_5 n + 3n}{5n^2 + 4n + 2} &= \lim_{n\to\infty} \frac{\frac{d}{dx}(n \log_5 n + 3n)}{\frac{d}{dx}(5n^2 + 4n + 2)} \\
&= \lim_{n\to\infty} \frac{n \frac{1}{n \ln 5} + \log_5 n + 3}{10n + 4} \\
&= \lim_{n\to\infty} \frac{\frac{d}{dx}(\frac{1}{\ln 5} + \log_5 n + 3)}{\frac{d}{dx}(10n + 4)} \\
&= \lim_{n\to\infty} \frac{\frac{1}{n \ln 5}}{10} \\
&= \lim_{n\to\infty} \frac{1}{10n(\ln 5)}
\end{aligned}
$$

$$
\lim_{n\to\infty} \frac{n \log_5 n + 3n}{5n^2 + 4n + 2} = 0
$$

Asymptotically Loose Relationships

You might be wondering why, we read asymptotic notation with the qualifier "Big", why do we have to specify **Big**-O, or **Big**-Omega? Well, you might have guessed it, its because there are "little" versions of O, and Ω

Little-o

$$o(g(n)) = \{f(n) | \forall c > 0 (\exists n_0 > 0 (\forall n > n_0 (0 \leq f(n) \leq cg(n))))\}$$

For all positive constants c, there exists a positive constant n_0, such that $0 \leq f(n) \leq cg(n)$ for all $n > n_0$, (for sufficiently large values of n)

Little-o

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

Little-o

$$5n^2 + 3n \in o(2n^3)$$

Little-ω

$$\omega(g(n)) = \{f(n) | \forall c > 0 (\exists n_0 > 0 (\forall n > n_0 (0 \leq cg(n) \leq f(n))))\}$$

For all positive constants c, there exists a positive constant n_0, such that $0 \leq cg(n) \leq f(n)$ for all $n > n_0$, (for sufficiently large values of n)

Little-ω

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

Little-ω

$$2n^3 \in \omega(5n^2 + 3n)$$

Is there no such thing as little-θ?

Try to form the definition of little-θ based on the definitions we have created earlier. Similar to how O and Ω conjunctively combine into Θ, the little relationships, o and ω must combine into θ. Once you manage to create this definition, you'll realize that for any function $g(n)$, $\theta(g(n)) = \emptyset$. No such function is a member of this set.

Asymptotic Tightness

If $f(n) \in O(g(n))$ but $f(n) \notin o(g(n))$, then we call their relationship **asymptotically tight**, meaning $f(n)$ is shown to be upper bounded by $g(n)$, but at the same time this bound is close/tight to $f(n)$, because they actually turn out to have the exact same complexity, $f(n) \in \Theta(g(n))$

Asymptotic Tightness

- $n^2 + n \in O(n^2)$
- $n^2 + n \notin o(n^2)$
- $n^2 + n \in \Theta(n^2)$

Analogy to inequalities/equalities

if we look at asymptotic relationships as something similar to inequalities/equalities it becomes easier to understand their concepts

Analogy to inequalities/equalities

$$L(x) = \{y | y \leq x\}$$

Analogy to inequalities/equalities

- **he set $L(x)$ is basically the set of all numbers less than or equal to x**
- **$L(x)$ can be seen as the $O(g(n))$ but for numbers instead of functions**

Algebraic Inequalities/Equalities	Asymptotic Notation
$L(x) = \{y \mid y \leq x\}$	$f(n) \in O(g(n))$
$G(x) = \{y \mid y \geq x\}$	$f(n) \in \Omega(g(n))$
$E(x) = \{y \mid y = x\}$	$f(n) \in \Theta(g(n))$

Algebraic Inequalities/Equalities	**Asymptotic Notation**
$l(x) = \{y \| y < x\}$	$f(n) \in o(g(n))$
$g(x) = \{y \| y > x\}$	$f(n) \in \omega(g(n))$

Asymptotically tight binding

- If $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$, then $f(n) \in \Theta(g(n))$.
- If $f(n) \in \Theta(g(n))$ then $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.
- Also, $O(g(n)) \cap \Omega(g(n)) = \Theta(g(n))$.

Asymptotic subsets

- If $f(n) \in o(g(n))$ then $f(n) \in O(g(n))$,
- $o(g(n)) \subset O(g(n))$.
- If $f(n) \in \omega(g(n))$ then $f(n) \in \Omega(g(n))$,
- $\omega(g(n)) \subset \Omega(g(n))$.

Asymptotic non-memberships

- If $f(n) \notin O(g(n))$ then $f(n) \in \omega(g(n))$.
- If $f(n) \notin \Omega(g(n))$ then $f(n) \in o(g(n))$.
- $\overline{O(g(n))} = \omega(g(n))$.
- $\overline{\Omega(g(n))} = o(g(n))$.

Application to Computer Science

- By mastering this your focus shifts from writing algorithms that work to writing algorithms that work **efficiently**
- It's not only important to write programs that work, it becomes important that these programs also work **quicker** and use **lesser resources**.

Application to Computer Science

- How do we judge which algorithm is more efficient
- How do we compare their speeds?

Application to Computer Science

- One thing we can do is to look at empirical evidence and try to look at their **running times** when the algorithm is converted to actual code
- **the running time is dependent on the size of the given array**
- If the given array is **shorter** in length then both algorithms have **less elements** to sort
- Therefore, the algorithm will take less time time to run and the running time will be **faster**
- The inverse is also true, if the given array is **longer**,

Application to Computer Science

- **the running time for an algorithm cannot be simply represented as one number**
- Most of the time, an algorithm's running time is dependent on the **size of the input** [^4]

Application to Computer Science

- A more complete way of representing running time would be through a **mathematical function** in terms of the input size
- This is why you'll usually see running times denoted as $T(n)$, where n is the input size.

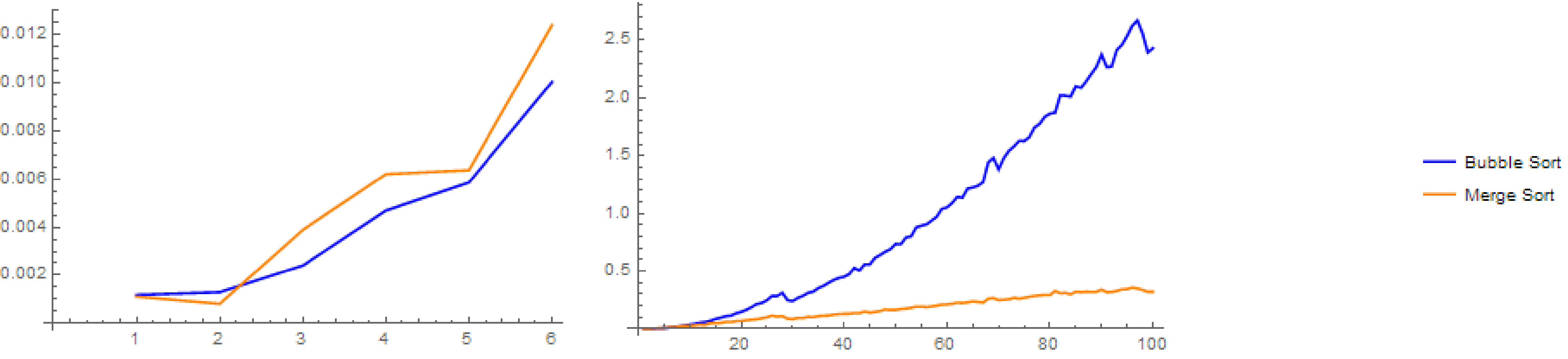
Application to Computer Science

But as it turns out, the process of figuring out the **exact** mathematical function to represent the running time is not an exact science as well

Array Size	Runtime in s (Bubble Sort)	Runtime in s (Merge Sort)
1	0.001188	0.001119
2	0.001301	0.000818
10	0.035985	0.028135

Array Size	Runtime in s (Bubble Sort)	Runtime in s (Merge Sort)
20	0.146644	0.067852
30	0.239448	0.084049
50	0.735932	0.163289

Array Size	Runtime in s (Bubble Sort)	Runtime in s (Merge Sort)
70	1.382356	0.248158
100	2.432715	0.320786



Measuring running time through experiment

Measuring running time through experiment

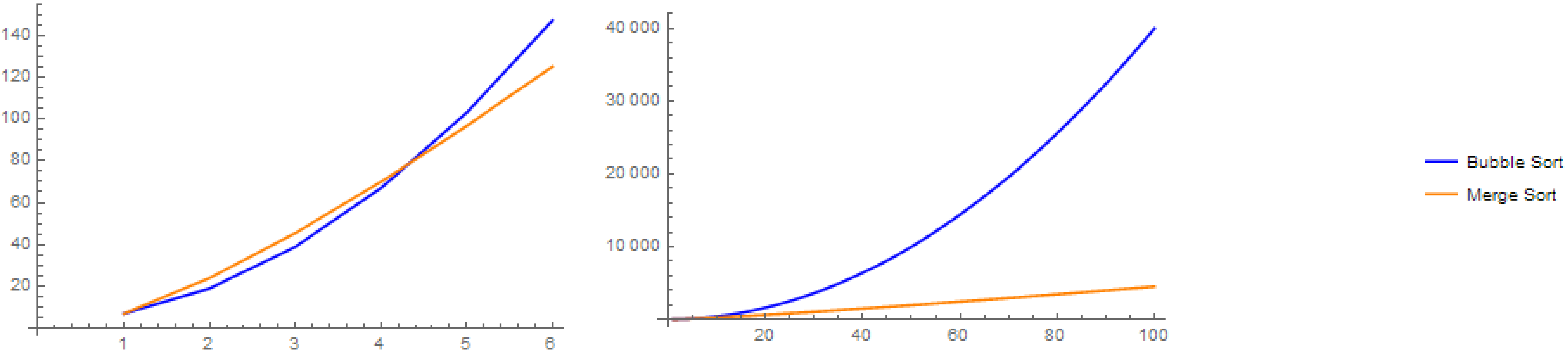
- By comparing the plots of their running times, we can infer that, for **smaller** input sizes, both algorithm's have **similar** running times
- As the input size becomes **bigger** and bigger, both algorithm's running times increase but **bubble sort's running time increases noticeably faster**

Measuring running time through instruction approximation

Another method of approximating the running time functions is by approximating the number of **instructions** executed during the algorithms lifetime[^5]

Measuring running time through instruction approximation

- Bubble Sort: $T(n) = 4n^2 + 3$
- Merge Sort: $T(n) = 6n \log_2 n + 5n + 2$



Measuring running time through instruction approximation

Measuring running time through instruction approximation

- The running time functions that we ended up with also shows that for **smaller input sizes** both algorithms end up executing pretty much the same amount of instructions
- For **larger input sizes**, it also shows that merge sort is clear winner, it ends up executing way fewer instructions compared to bubble sort.

through instruction approximation

- Merge sort is the preferred algorithm between the two
- for cases that matter the most (large input sizes), merge sort is way faster
- we expect that larger input sizes will take a longer time
- we instead focus on the bigger picture that matters most
- Which algorithm performs better in stressful

Measuring running time through instruction approximation

- What we are comparing in these two representations (experimental running time and approximate instruction running time) are running times for the **written code** for the algorithms, not the algorithms themselves
- These running time measurements are attributed specifically to the Python **code** I wrote for the algorithm

Finally, Asymptotic Notation

1. **Algorithm efficiency is highly dependent on input size, the best representations of efficiency is a function.**
2. **There is no reliable way to compare algorithm efficiency. The ways that we have always end up being approximations.**
3. **We care the most about how the algorithm performs in large input sizes**

Finally, Asymptotic Notation

1. **Since algorithm efficiency almost always ends up being functions of input sizes, we can makes use of asymptotic notation, which compare relationships between functions.** Remember that the member's of O, Ω, Θ, o, ω, are functions. We've even shown how asymptotic relationships are similar to inequality/equality relationships, but instead of comparing numerical values, we compare functions.

Finally, Asymptotic Notation

2. **You'll find that asymptotic notation relationships are good at calculating approximated values.** The asymptotic relationships between two functions still hold true even if you change the functions a little bit. For example $2n^2+5n \in \Theta(4n^2+3)$, even if we change the constant coefficients in these functions, their asymptotic relationship still stays the same: $an^2+bn \in \Theta(cn^2+d)$. This characteristic is useful in the case of running time comparisons because as we've shown, our methods of measurements are not precise enough or uniform enough. Asymptotic notation doesn't care about the small mistakes in

Finally, Asymptotic Notation

3. **Asymptotic notation by definition only cares about large values of n.** If we recall, asymptotic relationships only care about what happens on **sufficiently large values of n**, or on values of $n \to \infty$.

Finally, Asymptotic Notation

$$4n^2 + 3 \in \Omega(6n \log_2 n + 5n + 2)$$

Finally, Asymptotic Notation

This leads us to the conclusion that bubble sort's **time complexity** is higher than merge sort's time complexity

Dropping the coefficient

$$c > 0 \to cf(n) \in \Theta(f(n))$$

Dropping the coefficient

Given a function multiplied to some constant, $cf(n)$, its complexity is exactly the same as $f(n)$.

Proof

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

Higher degree polynomial

$$c < d \to f(n)^c \in o(f(n)^d)$$

Higher degree polynomial

Given $f(n)^c$ and $f(n)^d$ where c and d are constants and $c < d$, $f(n)^c$ is less complex than $f(n)^d$

Let $0 < c < d$

$$\begin{aligned}
\lim_{n\to\infty} \frac{f(n)^c}{f(n)^d} &= \lim_{n\to\infty} \frac{cf(n)^{c-1}\frac{d}{dn}f(x)}{df(n)^{d-1}\frac{d}{dn}f(x)} \\
&= \lim_{n\to\infty} \frac{cf(n)^{c-1}}{df(n)^{d-1}} \\
&= \lim_{n\to\infty} \frac{c(c-1)f(n)^{c-2}}{d(d-1)f(n)^{d-2}} \\
&\vdots \\
&= \lim_{n\to\infty} \frac{c(c-1)(c-2)(c-3)\cdots(c-(c-1))f(n)^{c-c}}{d(d-1)(d-2)(d-3)\cdots(d-(c-1))f(n)^{d-c}} \\
&= \lim_{n\to\infty} \frac{c(c-1)(c-2)(c-3)\cdots(c-(c-1))}{d(d-1)(d-2)(d-3)\cdots(d-(c-1))f(n)^{d-c}}
\end{aligned}$$

Sum of functions

$$f(n) \in \Omega(g(n)) \to f(n) + g(n) \in \Theta(f(n))$$

Sum of functions

In a sum of functions, the most complex function dominates the sum therefore the overall complexity will be equal to the dominant function.

Proof

Since $f(n) = \Omega(f(n))$,

for any $n > n_0$

$$f(n) \geq cf(n)$$
$$f(n) + g(n) \geq cf(n)$$

adding $g(n)$ to the left hand side of the inequality does not change the inequality relationship since $g(n)$ is positive (meaning, adding $g(n)$ can only result to increasing the value of the already bigger value)

Therefore,

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

Also since $g(n) = O(f(n))$,

$$g(n) \leq cf(n)$$
$$f(n) + g(n) \leq cf(n) + f(n)$$
$$f(n) + g(n) \leq (c+1)f(n)$$

Therefore,

$$f(n) + g(n) \in O(f(n))$$

And since

$$f(n) + g(n) = \Omega(f(n)) \wedge f(n) + g(n) \in O(f(n)),$$
$$f(n) + g(n) \in \Theta(f(n))$$

Different bases on log

Bases for logarithms doesn't matter in terms of asymptotic notation therefore $\log_b n$ for any base is of complexity $\Theta(\log n)$.

Proof

$$
\begin{aligned}
\lim_{n\to\infty} \frac{\log_a n}{\log_b n} &= \lim_{n\to\infty} \frac{\frac{\log_b n}{\log_b a}}{\log_b n} \\
&= \lim_{n\to\infty} \frac{\log_b n}{\log_b n(\log_b a)} \\
&= \lim_{n\to\infty} \frac{1}{\log_b a}
\end{aligned}
$$

$$
\lim_{n\to\infty} \frac{\log_a n}{\log_b n} = \frac{1}{\log_b a}
$$

Different bases on exponential functions

$$0 < a < b \to a^n \in o(b^n)$$

Different bases on exponential functions

Bases for exponential functions on the other hand matter

Proof

$$\begin{aligned}\lim_{x\to\infty}\frac{a^n}{b^n} &= \lim_{x\to\infty}\frac{a^n}{\left(a^{\log_a b}\right)^n}\\ &= \lim_{x\to\infty}\frac{a^n}{\left(a^n\right)^{\log_a b}}\\ &= \lim_{x\to\infty}\left(a^n\right)^{1-\log_a b}\end{aligned}$$

since $b > a$, then, $1 - \log_a b < 0$. Therefore,

$$\lim_{x\to\infty}\frac{a^n}{b^n} = 0$$

Different bases on exponential functions

Based on these rules a complicated looking function can be **reduced** to the **simplest function** that matches its complexity

Different bases on exponential functions

$$6n\log_2 n + 5n + 2 \in \Theta(6n\log_2 n)$$

Different bases on exponential functions

$$6n \log_2 n + 5n + 2 \in \Theta(n \log_2 n)$$

Different bases on exponential functions

$$6n \log_2 n + 5n + 2 \in \Theta(n \log n)$$

Different bases on exponential functions

we can use $n \log n$ as sort of the representative of $6n \log_2 n + 5n + 2$ on matters regarding its complexity

Complexity Classes

In the domain of computer science we know a few of these *representative* functions, we call these, **complexity classes**

Complexity Class	**Name**	**Example**
$\Theta(1)$	Constant	converting Celsius to Fahrenheit
$\Theta(\log n)$	Logarithmic	binary search
$\Theta((\log n)^c)$	Polylogarithmic	dynamic planarity testing of a graph

Complexity Class	Name	Example
$\Theta(n^c)$ where $0 < c < 1$	Fractional power	primality test
$\Theta(n)$	Linear	finding the smallest number in an array
$\Theta(n \log n)$	Quasilinear	merge sort

Complexity Class	Name	Example
$\Theta(n^2)$	Quadratic	bubble sort
$\Theta(n^c)$ where $c > 1$	Polynomial	naïve matrix multiplication
$\Theta(c^n)$	Exponential	dynamic travelling salesman

Complexity Class	Name	Example
$\Theta(n!)$	Factorial	brute force travelling salesman

Using O versus using Θ

- When specifying time complexity, you'll find that it is usually expressed in terms of **O**
- **expressing the time complexity in terms of Θ is more specific**

Using O versus using Θ

- computer scientists will end up expressing time complexities in terms of O as well because we generally want our algorithms to run on some acceptable standard of complexity
- asymptotic notation in terms of O will usually suffice