Uses and limitations of asymptotic analysis

Or: How to assume like a computer scientist

[AUT CSC]

AUT Computer Science Club

Koz Ross

April 13, 2017

Overview

- Recap and formalisms
- The usual suspects
- Limitations
- Questions

Recap and formalisms

Where we last left our intrepid heroes...

- Algorithms are recipes for a computer

Where we last left our intrepid heroes. . .

- Algorithms are recipes for a computer
- We want them to be efficient (time and space)

Where we last left our intrepid heroes. . .

- Algorithms are recipes for a computer
- We want them to be efficient (time and space)
- Must have a way to compare algorithms, and explain their performance

Where we last left our intrepid heroes. . .

- Algorithms are recipes for a computer
- We want them to be efficient (time and space)
- Must have a way to compare algorithms, and explain their performance
- Implement-and-measure is *not* a good approach

Where we last left our intrepid heroes. . .

- Algorithms are recipes for a computer
- We want them to be efficient (time and space)
- Must have a way to compare algorithms, and explain their performance
- Implement-and-measure is *not* a good approach
- A model of a computer (RAM)

Where we last left our intrepid heroes. . .

- Algorithms are recipes for a computer
- We want them to be efficient (time and space)
- Must have a way to compare algorithms, and explain their performance
- Implement-and-measure is *not* a good approach
- A model of a computer (RAM)
- Worst-case, asymptotic analysis

Example from last week

function $\text{Max}(\text{arr})$
    $\text{x} \leftarrow \text{arr}[1]$
    for $\text{i} \in 2, 3, \ldots, \text{length}(\text{arr})$ **do**
        if $\text{x} < \text{arr}[\text{i}]$ **then**
            $\text{x} \leftarrow \text{arr}[\text{i}]$
        end if
    end for
    return x
end function

Example from last week

function Max(arr)
  x ← arr[1]
  for i $\in 2, 3, \ldots,$ length(arr) **do**
    if x $<$ arr[i] **then**
      x ← arr[i]
    end if
  end for
  return x
end function

$T_{\text{Max}}(n)$ is $O(n)$

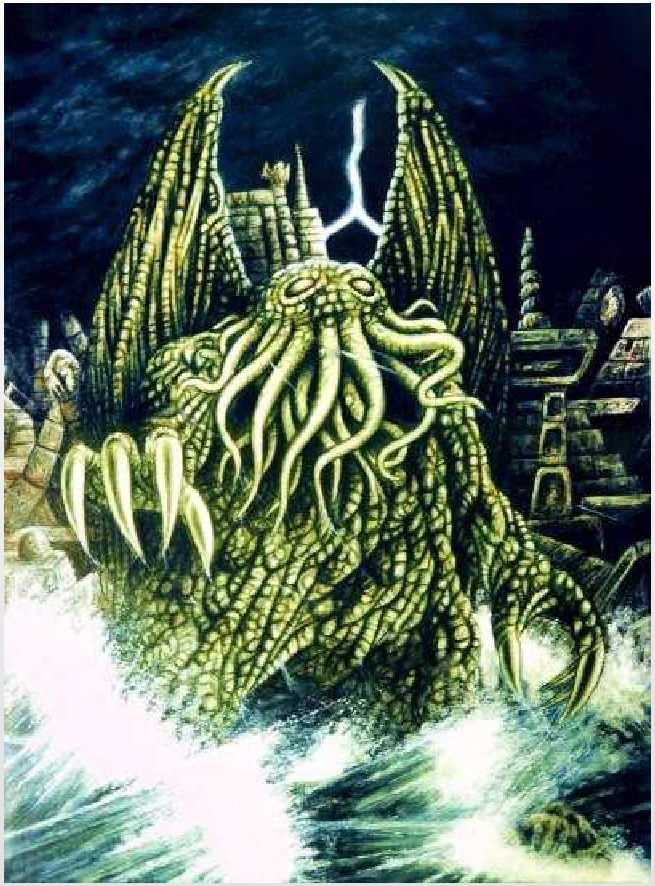
Example from last week

function Max(arr)
  x ← arr[1]
  for i ∈ $2, 3, \ldots,$ length(arr) **do**
    if x < arr[i] **then**
      x ← arr[i]
    end if
  end for
  return x
end function

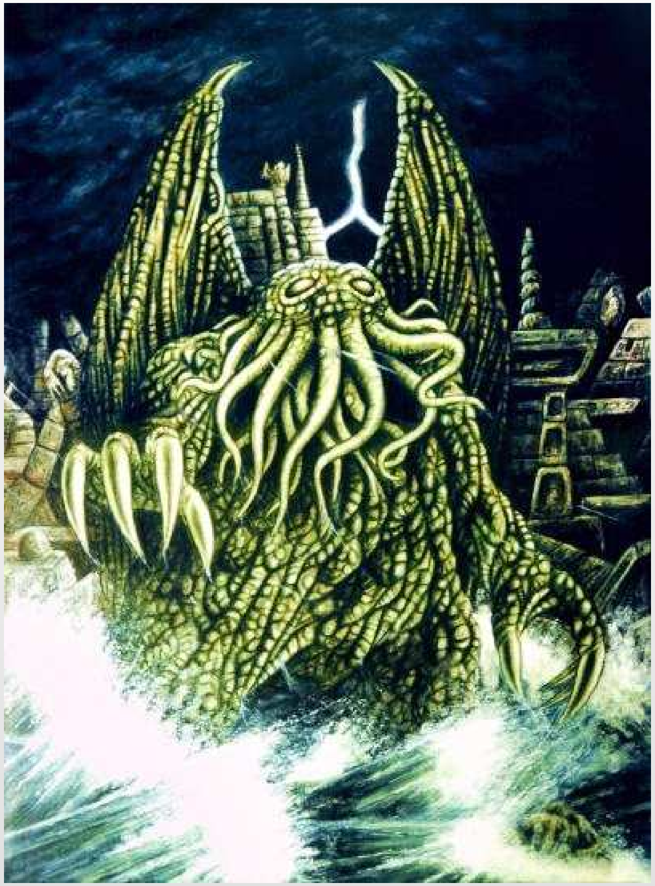
$T_{\mathsf{Max}}(n)$ is $O(n)$

$S_{\mathsf{Max}}(n)$ is $O(1)$

What people see when they look at a formal definition of O



What people see when they look at a formal definition of O



Don’t worry — we’ll go **slow**.

Definition of O (at last!)

Definition. *Let f, g be functions.*

Definition of O (at last!)

Definition. *Let f, g be functions. If there exist $n_0, c > 0$,*

Definition of O (at last!)

Definition. *Let f, g be functions. If there exist $n_0, c > 0$, such that for all $n > n_0$, we have:*

Definition of O (at last!)

Definition. *Let f, g be functions. If there exist $n_0, c > 0$, such that for all $n > n_0$, we have:*

$$f(n) \leq c \cdot g(n)$$

Definition of O (at last!)

Definition. *Let f, g be functions. If there exist $n_0, c > 0$, such that for all $n > n_0$, we have:*

$$f(n) \leq c \cdot g(n)$$

then we say that f is *$O(g)$.*

Definition of O (at last!)

Definition. *Let f, g be functions. If there exist $n_0, c > 0$, such that for all $n > n_0$, we have:*

$$f(n) \leq c \cdot g(n)$$

then we say that f is *$O(g)$.*

We write $O(n)$ as a kind of shorthand — otherwise, we would have to write $O(f$ such that $f(n) = n)$, which is far too long.

Examples of use

Lemma. *$6n - 1$ is $O(n)$.*

Examples of use

Lemma. $6n - 1$ *is* $O(n)$.

Proof. Let $n_0 = 1, c = 7$.

Examples of use

Lemma. $6n - 1$ *is* $O(n)$.

Proof. Let $n_0 = 1, c = 7$. We observe that, for all $n > 1$, we have

$$6n - 1 \leq 7n$$

Examples of use

Lemma. $6n - 1$ *is* $O(n)$.

Proof. Let $n_0 = 1, c = 7$. We observe that, for all $n > 1$, we have

$$6n - 1 \leq 7n$$

Thus, by definition, $6n - 1$ is $O(n)$. $\square$

Examples of use

Lemma. n^2 *is* not $O(n)$.

Examples of use

Lemma. n^2 *is* not $O(n)$.

Proof. Suppose for the sake of a contradition that n^2 is $O(n)$.

Examples of use

Lemma. n^2 *is* not $O(n)$.

Proof. Suppose for the sake of a contradition that n^2 is $O(n)$. By definition, there exist some $n_0, c > 0$ such that for all $n > n_0$, $n^2 \leq c \cdot n$.

Examples of use

Lemma. n^2 *is not* $O(n)$.

Proof. Suppose for the sake of a contradition that n^2 is $O(n)$. By definition, there exist some $n_0, c > 0$ such that for all $n > n_0$, $n^2 \leq c \cdot n$. Consider $n = \max\{n_0 + 1, c + 1\}$.

Examples of use

Lemma. n^2 *is* not $O(n)$.

Proof. Suppose for the sake of a contradition that n^2 is $O(n)$. By definition, there exist some $n_0, c > 0$ such that for all $n > n_0$, $n^2 \leq c \cdot n$. Consider $n = \max\{n_0 + 1, c + 1\}$. There are two cases:

Examples of use

Lemma. n^2 *is* not $O(n)$.

Proof. Suppose for the sake of a contradition that n^2 is $O(n)$. By definition, there exist some $n_0, c > 0$ such that for all $n > n_0$, $n^2 \leq c \cdot n$. Consider $n = \max\{n_0 + 1, c + 1\}$. There are two cases:

Case 1: $n = n_0 + 1$

Examples of use

Lemma. n^2 *is not* $O(n)$*.*

Proof. Suppose for the sake of a contradition that n^2 is $O(n)$. By definition, there exist some $n_0, c > 0$ such that for all $n > n_0$, $n^2 \leq c \cdot n$. Consider $n = \max\{n_0 + 1, c + 1\}$. There are two cases:

Case 1: $n = n_0 + 1$
By substitution, we have $(n_0 + 1)^2 \leq c \cdot (n_0 + 1)$, which simplifies to $n_0 + 1 \leq c$.

Examples of use

Lemma. n^2 *is not* $O(n)$*.*

Proof. Suppose for the sake of a contradition that n^2 is $O(n)$. By definition, there exist some $n_0, c > 0$ such that for all $n > n_0$, $n^2 \leq c \cdot n$. Consider $n = \max\{n_0 + 1, c + 1\}$. There are two cases:

Case 1: $n = n_0 + 1$
By substitution, we have $(n_0 + 1)^2 \leq c \cdot (n_0 + 1)$, which simplifies to $n_0 + 1 \leq c$. However, this is a contradiction, as $c < n_0 + 1$.

Examples of use

Lemma. *n^2 is* not *$O(n)$.*

Proof. Suppose for the sake of a contradition that n^2 is $O(n)$. By definition, there exist some $n_0, c > 0$ such that for all $n > n_0$, $n^2 \leq c \cdot n$. Consider $n = \max\{n_0 + 1, c + 1\}$. There are two cases:

Case 1: $n = n_0 + 1$
By substitution, we have $(n_0 + 1)^2 \leq c \cdot (n_0 + 1)$, which simplifies to $n_0 + 1 \leq c$. However, this is a contradiction, as $c < n_0 + 1$.

Case 2: $n = c + 1$

Examples of use

Lemma. n^2 *is* not $O(n)$.

Proof. Suppose for the sake of a contradition that n^2 is $O(n)$. By definition, there exist some $n_0, c > 0$ such that for all $n > n_0$, $n^2 \leq c \cdot n$. Consider $n = \max\{n_0 + 1, c + 1\}$. There are two cases:

Case 1: $n = n_0 + 1$
By substitution, we have $(n_0 + 1)^2 \leq c \cdot (n_0 + 1)$, which simplifies to $n_0 + 1 \leq c$. However, this is a contradiction, as $c < n_0 + 1$.

Case 2: $n = c + 1$
By substitution, we have $(c + 1)^2 \leq c \cdot (c + 1)$, which simplifies to $c + 1 \leq c$, which is a contradiction.

Examples of use

Lemma. *n^2 is not $O(n)$.*

Proof. Suppose for the sake of a contradition that n^2 is $O(n)$. By definition, there exist some $n_0, c > 0$ such that for all $n > n_0$, $n^2 \leq c \cdot n$. Consider $n = \max\{n_0 + 1, c + 1\}$. There are two cases:

Case 1: $n = n_0 + 1$
By substitution, we have $(n_0 + 1)^2 \leq c \cdot (n_0 + 1)$, which simplifies to $n_0 + 1 \leq c$. However, this is a contradiction, as $c < n_0 + 1$.

Case 2: $n = c + 1$
By substitution, we have $(c + 1)^2 \leq c \cdot (c + 1)$, which simplifies to $c + 1 \leq c$, which is a contradiction.

As both cases lead to contradictions, no such n_0, c can exist. Thus, n^2 is not $O(n)$. $\square$

The usual suspects

The ‘good’ ones

The ‘good’ ones

$O(1)$

Constant: input size does not affect complexity.

The ‘good’ ones

$O(1)$

Constant: input size does not affect complexity.

$O(\log(n))$

Logarithmic: when input size doubles, complexity goes up by a fixed amount.

The ‘good’ ones

$O(1)$

Constant: input size does not affect complexity.

$O(\log{(n)})$

Logarithmic: when input size doubles, complexity goes up by a fixed amount. We don’t care about the base of the logarithm here, because we can change to any base using a multiplication by a constant.

The ‘good’ ones

$O(1)$

Constant: input size does not affect complexity.

$O(\log{(n)})$

Logarithmic: when input size doubles, complexity goes up by a fixed amount. We don’t care about the base of the logarithm here, because we can change to any base using a multiplication by a constant.

$O(n)$

Linear: when input size doubles, complexity also doubles.

The ‘acceptable’ ones

The ‘acceptable’ ones

$O(n \log{(n)})$

Linearithmic: when input size doubles, complexity doubles, then also increases by a fixed amount.

The ‘acceptable’ ones

$O(n \log{(n)})$

Linearithmic: when input size doubles, complexity doubles, then also increases by a fixed amount. Its ‘wordy’ name is rarely used outside of a textbook.

The ‘acceptable’ ones

$O(n \log(n))$

Linearithmic: when input size doubles, complexity doubles, then also increases by a fixed amount. Its ‘wordy’ name is rarely used outside of a textbook.

$O(n^2)$

Quadratic: when input size doubles, complexity *quadruples*.

The ‘acceptable’ ones

$O(n \log(n))$

Linearithmic: when input size doubles, complexity doubles, then also increases by a fixed amount. Its ‘wordy’ name is rarely used outside of a textbook.

$O(n^2)$

Quadratic: when input size doubles, complexity *quadruples*.

$O(n^3)$

Cubic: when input size doubles, complexity increases by a factor of eight.

The ‘*un*acceptable’ ones

The ‘*un*acceptable’ ones

$\mathbf{O(2^n)}$

Exponential: when input size goes up by 1, complexity doubles.

The ‘*un*acceptable’ ones

$\mathbf{O(2^n)}$

Exponential: when input size goes up by 1, complexity doubles. If your algorithm has exponential time or space complexity, it may as well not exist.

The ‘*un*acceptable’ ones

$\mathbf{O(2^n)}$

Exponential: when input size goes up by 1, complexity doubles. If your algorithm has exponential time or space complexity, it may as well not exist.

Only gets *worse* from here — but we usually don’t bother at this point.

Limitations

Limitations of worst-case analysis

Worst-case analysis is *pessimistic* — this can be a good thing, because it stops us from having unrealistic expectations in the face of unknown inputs.

Limitations of worst-case analysis

Worst-case analysis is *pessimistic* — this can be a good thing, because it stops us from having unrealistic expectations in the face of unknown inputs. However, it is possible for it to be *too* pessimistic:

Limitations of worst-case analysis

Worst-case analysis is *pessimistic* — this can be a good thing, because it stops us from having unrealistic expectations in the face of unknown inputs. However, it is possible for it to be *too* pessimistic:

- If worst cases are rare or improbable, an algorithm might look much worse than it actually is

Limitations of worst-case analysis

Worst-case analysis is *pessimistic* — this can be a good thing, because it stops us from having unrealistic expectations in the face of unknown inputs. However, it is possible for it to be *too* pessimistic:

- If worst cases are rare or improbable, an algorithm might look much worse than it actually is
- Focus on the wrong places (exactly what we sought to *avoid*)

Limitations of worst-case analysis

Worst-case analysis is *pessimistic* — this can be a good thing, because it stops us from having unrealistic expectations in the face of unknown inputs. However, it is possible for it to be *too* pessimistic:

- If worst cases are rare or improbable, an algorithm might look much worse than it actually is
- Focus on the wrong places (exactly what we sought to *avoid*)
- Won't match reality (and we won't know why)

Limitations of worst-case analysis

Worst-case analysis is *pessimistic* — this can be a good thing, because it stops us from having unrealistic expectations in the face of unknown inputs. However, it is possible for it to be *too* pessimistic:

- If worst cases are rare or improbable, an algorithm might look much worse than it actually is
- Focus on the wrong places (exactly what we sought to *avoid*)
- Won't match reality (and we won't know why)

Is it really sensible to *always* be *uncompromisingly* pessimistic?

Limitations of RAM

"RAM describes computers, and if we design algorithms around it, their actual performance should reflect what RAM says it should be."

Limitations of RAM

"RAM describes computers, and if we design algorithms around it, their actual performance should reflect what RAM says it should be."

That statement was certainly true in the 1950s, but things have changed *considerably* since then:

Limitations of RAM

"RAM describes computers, and if we design algorithms around it, their actual performance should reflect what RAM says it should be."

That statement was certainly true in the 1950s, but things have changed *considerably* since then:

- Processing units are no longer singular

Limitations of RAM

“RAM describes computers, and if we design algorithms around it, their actual performance should reflect what RAM says it should be.”

That statement was certainly true in the 1950s, but things have changed *considerably* since then:

- Processing units are no longer singular
- Memory is not uniform or uniformally-addressable nowadays

Limitations of RAM

"RAM describes computers, and if we design algorithms around it, their actual performance should reflect what RAM says it should be."

That statement was certainly true in the 1950s, but things have changed *considerably* since then:

- Processing units are no longer singular
- Memory is not uniform or uniformally-addressable nowadays
- Processing units haven't been sequential since the mid-70s

Limitations of RAM

> *"RAM describes computers, and if we design algorithms around it, their actual performance should reflect what RAM says it should be."*

That statement was certainly true in the 1950s, but things have changed *considerably* since then:

- Processing units are no longer singular
- Memory is not uniform or uniformally-addressable nowadays
- Processing units haven't been sequential since the mid-70s

Is it really sensible to analyze (and *design*) algorithms based on such an old view of our machines?

Limitations of asymptotic analysis

The assumptions of asymptotic analysis are self-consistent, but are they consistent with *reality*?

Limitations of asymptotic analysis

The assumptions of asymptotic analysis are self-consistent, but are they consistent with *reality*?

- There *is* a limit on how big inputs can get

Limitations of asymptotic analysis

The assumptions of asymptotic analysis are self-consistent, but are they consistent with *reality*?

- There *is* a limit on how big inputs can get
- $2n$ and $2^{10}n$ are both $O(n)$, but practically, are *very* different, especially for space

Limitations of asymptotic analysis

The assumptions of asymptotic analysis are self-consistent, but are they consistent with *reality*?

- There *is* a limit on how big inputs can get
- $2n$ and $2^{10}n$ are both $O(n)$, but practically, are *very* different, especially for space
- Constants can, and *do*, dominate algorithm performance

Limitations of asymptotic analysis

The assumptions of asymptotic analysis are self-consistent, but are they consistent with *reality*?

- There *is* a limit on how big inputs can get
- $2n$ and $2^{10}n$ are both $O(n)$, but practically, are *very* different, especially for space
- Constants can, and *do*, dominate algorithm performance

By clever 'asymptotic abuse', we can easily come up with algorithms that 'look good', but are worthless in practice:

> *"Our algorithms have theoretical interest only; the constant factors involve in the execution times preclude practicality."*

Limitations of asymptotic analysis

The assumptions of asymptotic analysis are self-consistent, but are they consistent with *reality*?

- There *is* a limit on how big inputs can get
- $2n$ and $2^{10}n$ are both $O(n)$, but practically, are *very* different, especially for space
- Constants can, and *do*, dominate algorithm performance

By clever 'asymptotic abuse', we can easily come up with algorithms that 'look good', but are worthless in practice:

> *"Our algorithms have theoretical interest only; the constant factors involve in the execution times preclude practicality."*

This seems to fly in the face of why we're doing this in the first place. But where to draw the line?

Should we bother with it, then?

- Asymptotic worst-case analysis based on RAM underpins many results, and would be worth knowing just for that reason alone

Should we bother with it, then?

- Asymptotic worst-case analysis based on RAM underpins many results, and would be worth knowing just for that reason alone
- Alternatives to it exist — but to understand them, you need a foundation (guess what *that* is?)

Should we bother with it, then?

- Asymptotic worst-case analysis based on RAM underpins many results, and would be worth knowing just for that reason alone
- Alternatives to it exist — but to understand them, you need a foundation (guess what *that* is?)
- In many cases (*taking assumptions into account*), still works very well (similar to classical mechanics)

Should we bother with it, then?

- Asymptotic worst-case analysis based on RAM underpins many results, and would be worth knowing just for that reason alone
- Alternatives to it exist — but to understand them, you need a foundation (guess what *that* is?)
- In many cases (*taking assumptions into account*), still works very well (similar to classical mechanics)
- ‘The beginning, but not the end’

Should we bother with it, then?

- Asymptotic worst-case analysis based on RAM underpins many results, and would be worth knowing just for that reason alone
- Alternatives to it exist — but to understand them, you need a foundation (guess what *that* is?)
- In many cases (*taking assumptions into account*), still works very well (similar to classical mechanics)
- ‘The beginning, but not the end’
- “If you don’t have an implementation, you don’t have an algorithm”

Should we bother with it, then?

- Asymptotic worst-case analysis based on RAM underpins many results, and would be worth knowing just for that reason alone
- Alternatives to it exist — but to understand them, you need a foundation (guess what *that* is?)
- In many cases (*taking assumptions into account*), still works very well (similar to classical mechanics)
- ‘The beginning, but not the end’
- “If you don’t have an implementation, you don’t have an algorithm”

“Trust, but verify.”

Questions