6331 - Algorithms, CSE, OSU

Introduction, complexity of algorithms, asymptotic growth of functions

Instructor: Anastasios Sidiropoulos
Why algorithms?

Algorithms are at the core of Computer Science
Why algorithms?

Algorithms are at the core of Computer Science

- Data bases: Data structures.
Why algorithms?

Algorithms are at the core of Computer Science

- Data bases: Data structures.
- Networks: Routing / communication algorithms.
Why algorithms?

Algorithms are at the core of Computer Science

- Data bases: Data structures.
- Networks: Routing / communication algorithms.
- Operating systems: Scheduling, paging, etc.
Why algorithms?

Algorithms are at the core of Computer Science

- Data bases: Data structures.
- Networks: Routing / communication algorithms.
- Operating systems: Scheduling, paging, etc.
- AI: Learning algorithms, etc.
Why algorithms?

Algorithms are at the core of Computer Science

- Data bases: Data structures.
- Networks: Routing / communication algorithms.
- Operating systems: Scheduling, paging, etc.
- AI: Learning algorithms, etc.
- Graphics: Rendering algorithms.
Why algorithms?

Algorithms are at the core of Computer Science

- Data bases: Data structures.
- Networks: Routing / communication algorithms.
- Operating systems: Scheduling, paging, etc.
- AI: Learning algorithms, etc.
- Graphics: Rendering algorithms.
- Robotics: Motion planning / control algorithms, etc.
Why algorithms?

Algorithms are at the core of Computer Science

- Data bases: Data structures.
- Networks: Routing / communication algorithms.
- Operating systems: Scheduling, paging, etc.
- AI: Learning algorithms, etc.
- Graphics: Rendering algorithms.
- Robotics: Motion planning / control algorithms, etc.
- Game development
Algorithms beyond Computer Science

Algorithms are a transformative force in Science & Engineering.
Algorithms beyond Computer Science

Algorithms are a transformative force in Science & Engineering.

- Algorithms for processing complicated data.
Algorithms beyond Computer Science

Algorithms are a transformative force in Science & Engineering.

- Algorithms for processing complicated data.
- Computational Biology.
Algorithms beyond Computer Science

Algorithms are a transformative force in Science & Engineering.

- Algorithms for processing complicated data.
- Computational Biology.
- Medicine: drug design / delivery.
Algorithms beyond Computer Science

Algorithms are a transformative force in Science & Engineering.

- Algorithms for processing complicated data.
- Computational Biology.
- Medicine: drug design / delivery.
- Sociology / Economics: Algorithmic game theory.
Algorithms Beyond Computer Science

Algorithms are a transformative force in Science & Engineering.

▶ Algorithms for processing complicated data.
▶ Computational Biology.
▶ Medicine: drug design / delivery.
▶ Sociology / Economics: Algorithmic game theory.
▶ ...
What makes a good algorithm?

How can we quantify the performance of an algorithm?
What makes a good algorithm?

How can we quantify the performance of an algorithm?

**Computational resources**

- Time complexity: How much time does an algorithm need to terminate?
What makes a good algorithm?

How can we quantify the performance of an algorithm?

**Computational resources**

- **Time complexity**: How much time does an algorithm need to terminate?
- **Space complexity**: How much memory does an algorithm require?
What makes a good algorithm?

How can we *quantify* the performance of an algorithm?

**Computational resources**

- Time complexity: How much time does an algorithm need to terminate?
- Space complexity: How much memory does an algorithm require?
- In other contexts, we might also be interested in different parameters.
What makes a good algorithm?

How can we *quantify* the performance of an algorithm?

**Computational resources**

- **Time complexity:** How much time does an algorithm need to terminate?
- **Space complexity:** How much memory does an algorithm require?
- In other contexts, we might also be interested in different parameters.
  - **Communication complexity** (i.e. the total amount of bits exchanged in a system).
What makes a good algorithm?

How can we *quantify* the performance of an algorithm?

**Computational resources**

- **Time complexity**: How much time does an algorithm need to terminate?
- **Space complexity**: How much memory does an algorithm require?
- In other contexts, we might also be interested in different parameters.
  - Communication complexity (i.e. the total amount of bits exchanged in a system).
  - Waiting / service time (e.g. in queuing systems).
Worst-case complexity

The worst-case time complexity, or worst-case running time of an algorithm is a function $f : \mathbb{N} \to \mathbb{N}$, where

$$f(n) = \text{maximum \# of steps required on any input of size } n$$
Worst-case complexity

The worst-case time complexity, or worst-case running time of an algorithm is a function $f : \mathbb{N} \rightarrow \mathbb{N}$, where

$$f(n) = \text{maximum } \# \text{ of steps required on any input of size } n$$

More precisely:
For any input $x \in \{0, 1\}^n$, let

$$T(x) = \# \text{ of steps required on input } x$$
Worst-case complexity

The worst-case time complexity, or worst-case running time of an algorithm is a function \( f : \mathbb{N} \rightarrow \mathbb{N} \), where

\[
f(n) = \text{maximum number of steps required on any input of size } n
\]

More precisely:
For any input \( x \in \{0, 1\}^n \), let

\[
T(x) = \text{number of steps required on input } x
\]

Then,

\[
f(n) = \max_{x \in \{0,1\}^n} T(x)
\]
Example of worst-case complexity

Finding an element in an array.

Input: integer array $A[1 \ldots n]$, and integer $x$.

Find some $i$, if one exists, such that $A[i] = x$.

Algorithm

for $i = 1$ to $n$
  if $A[i] = x$
    output $i$, and terminate

output "not found"

What is the worst-case time complexity of this algorithm?

What is the best possible time complexity?
Example of worst-case complexity

Finding an element in an array.
Input: integer array $A[1 \ldots n]$, and integer $x$. 

Algorithm for $i = 1$ to $n$
if $A[i] = x$
output $i$, and terminate
end
output "not found"

What is the worst-case time complexity of this algorithm?
What is the best possible time complexity?
Example of worst-case complexity

Finding an element in an array.
Input: integer array $A[1\ldots n]$, and integer $x$.
Find some $i$, if one exists, such that $A[i] = x$. 

Algorithm
for $i = 1$ to $n$
if $A[i] = x$
output $i$, and terminate
end
output "not found"

What is the worst-case time complexity of this algorithm?
What is the best possible time complexity?
Example of worst-case complexity

Finding an element in an array.
Input: integer array $A[1 \ldots n]$, and integer $x$.
Find some $i$, if one exists, such that $A[i] = x$.

**Algorithm**
for $i = 1$ to $n$
    if $A[i] = x$ output $i$, and terminate
end
output “not found”
Example of worst-case complexity

Finding an element in an array.
Input: integer array $A[1 \ldots n]$, and integer $x$.
Find some $i$, if one exists, such that $A[i] = x$.

**Algorithm**
for $i = 1$ to $n$
  if $A[i] = x$ output $i$, and terminate
end
output “not found”

What is the worst-case time complexity of this algorithm?
Finding an element in an array.
Input: integer array $A[1\ldots n]$, and integer $x$.
Find some $i$, if one exists, such that $A[i] = x$.

**Algorithm**
for $i = 1$ to $n$
  if $A[i] = x$ output $i$, and terminate
end
output “not found”

What is the worst-case time complexity of this algorithm?
What is the best possible time complexity?
How do we compare different functions?

We will mostly deal with non-decreasing functions.
How do we compare different functions?

We will mostly deal with non-decreasing functions. In general, we cannot compare functions the same way we compare numbers.
How do we compare different functions?

We will mostly deal with non-decreasing functions. In general, we cannot compare functions the same way we compare numbers.

E.g., $n^2$ vs $1000000n$. Which one is “smaller”? 
\( O(g(n)) = \{ f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0 \} \)
O-notation

\[ O(g(n)) = \{ f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \]
\[ 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0 \} \]

E.g.

\[ 10n^2 + 5n - 100 \in O(n^2) \]
**O-notation**

\[
O(g(n)) = \{ f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0 \}
\]

E.g.

\[
10n^2 + 5n - 100 \in O(n^2)
\]

We write

\[
10n^2 + 5n - 100 = O(n^2)
\]
$O$-notation

$O(g(n)) = \{ f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0 \}$

E.g.

$10n^2 + 5n - 100 \in O(n^2)$

We write

$10n^2 + 5n - 100 = O(n^2)$

Examples:

- $n^2$ vs $1000000n$?
**$O$-notation**

\[ O(g(n)) = \{ f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \]
\[ 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0 \} \]

E.g.

\[ 10n^2 + 5n - 100 \in O(n^2) \]

We write

\[ 10n^2 + 5n - 100 = O(n^2) \]

Examples:

- \( n^2 \) vs \( 1000000n \)?
- \( n^{100} \) vs \( 2^n \)?
**O-notation**

\[ O(g(n)) = \{ f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \]
\[ 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0 \} \]

E.g.

\[ 10n^2 + 5n - 100 \in O(n^2) \]

We write

\[ 10n^2 + 5n - 100 = O(n^2) \]

Examples:

- \( n^2 \) vs \( 1000000n \)?
- \( n^{100} \) vs \( 2^n \)?
- \( n^{\log n} \) vs \( 2^n \)?
O-notation

\[ O(g(n)) = \{ f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0 \} \]

E.g.

\[ 10n^2 + 5n - 100 \in O(n^2) \]

We write

\[ 10n^2 + 5n - 100 = O(n^2) \]

Examples:

- \( n^2 \) vs 1000000\( n \)?
- \( n^{100} \) vs \( 2^n \)?
- \( n^{\log n} \) vs \( 2^n \)?
- \( 2^{2^n} \) vs \( 2^n \)?
Ω-notation

Ω(g(n)) = \{ f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\
0 \leq c \cdot g(n) \leq f(n) \text{ for all } n \geq n_0 \} 

Theorem

f(n) = O(g(n)) \text{ if and only if } g(n) = \Omega(f(n)).
Ω(g(n)) = \{ f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq c \cdot g(n) \leq f(n) \text{ for all } n \geq n_0 \}

**Theorem**

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

**Question:** Suppose that \( f(n) = \Omega(n) \). Does this imply that \( f(n) \) is increasing?
Θ-notation

\[ f(n) = \Theta(g(n)) \] if and only if both of the following hold:

- \[ f(n) = O(g(n)) \]
- \[ f(n) = \Omega(g(n)) \]
Θ-notation

\[ f(n) = \Theta(g(n)) \text{ if and only if both of the following hold:} \]
\[ \bullet \quad f(n) = O(g(n)) \]
\[ \bullet \quad f(n) = \Omega(g(n)) \]

Examples:
\[ \bullet \quad n^2 + n + 5 \text{ vs } 100n^2 + 5n + 3? \]
Θ-notation

\[ f(n) = \Theta(g(n)) \] if and only if both of the following hold:

- \[ f(n) = O(g(n)) \]
- \[ f(n) = \Omega(g(n)) \]

Examples:

- \[ n^2 + n + 5 \text{ vs } 100n^2 + 5n + 3? \]
- \[ n \cdot \log n \text{ vs } n^{1.0001}? \]
\( o(g(n)) = \{ f(n) : \text{for any positive constant } c > 0, \text{there exists a constant } n_0 \text{ such that} \}

\( 0 \leq f(n) < c \cdot g(n) \text{ for all } n \geq n_0 \} \)
$o(g(n)) = \{ f(n) : \text{for any positive constant } c > 0, \text{ there exists a constant } n_0 \text{ such that }$

$0 \leq f(n) < c \cdot g(n) \text{ for all } n \geq n_0 \}$

If $f(n) = o(g(n))$, then

$$\lim_{{n \to \infty}} \frac{f(n)}{g(n)} = 0$$
\( o(g(n)) = \{ f(n) : \text{for any positive constant } c > 0, \]
\( \text{there exists a constant } n_0 \text{ such that } 0 \leq f(n) < c \cdot g(n) \text{ for all } n \geq n_0 \} \)

If \( f(n) = o(g(n)) \), then

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0
\]

Examples:

\( 100n = o(n^2) \)
$o(g(n)) = \{ f(n) : \text{for any positive constant } c > 0, \text{ there exists a constant } n_0 \text{ such that } 0 \leq f(n) < c \cdot g(n) \text{ for all } n \geq n_0 \}$

If $f(n) = o(g(n))$, then

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

Examples:

- $100n = o(n^2)$
- $n^2 \neq o(n^2)$
ω-notation

\[ \omega(g(n)) = \{ f(n) : \text{for any positive constant } c > 0, \]
\[ \text{there exists a constant } n_0 \text{ such that } \]
\[ 0 \leq c \cdot g(n) < f(n) \text{ for all } n \geq n_0 \} \]
\( \omega(\text{g}(n)) = \{ f(n) : \text{for any positive constant } c > 0, \)

\( \text{there exists a constant } n_0 \text{ such that} \)

\( 0 \leq c \cdot \text{g}(n) < f(n) \text{ for all } n \geq n_0 \} \)

\( f(n) = o(\text{g}(n)) \) if and only if \( \text{g}(n) = \omega(f(n)) \).
\( \omega(g(n)) = \{ f(n) : \text{for any positive constant } c > 0, \]
\[ \text{there exists a constant } n_0 \text{ such that } \]
\[ 0 \leq c \cdot g(n) < f(n) \text{ for all } n \geq n_0 \} \]

\[ f(n) = o(g(n)) \text{ if and only if } g(n) = \omega(f(n)). \]

Examples:
- \( 2^n \text{ vs } n^{10} \)?
\( \omega(g(n)) = \{ f(n) : \text{for any positive constant } c > 0, \]
\[ \text{there exists a constant } n_0 \text{ such that}
\]
\[ 0 \leq c \cdot g(n) < f(n) \text{ for all } n \geq n_0 \} \]

\( f(n) = o(g(n)) \) if and only if \( g(n) = \omega(f(n)). \)

Examples:

- \( 2^n \) vs \( n^{10} \)?
- \( n \) vs \( n \cdot \log n \)?
\( \omega(g(n)) = \{ f(n) : \text{ for any positive constant } c > 0, \) 
\[ \text{there exists a constant } n_0 \text{ such that} \]
\[ 0 \leq c \cdot g(n) < f(n) \text{ for all } n \geq n_0 \} \]

\( f(n) = o(g(n)) \) if and only if \( g(n) = \omega(f(n)) \).

Examples:
- \( 2^n \) vs \( n^{10} \)?
- \( n \) vs \( n \cdot \log n \)?
- \( \log(n) \) vs \( \log(\log(n)) \)?