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

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

Worst-case complexity

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

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

Then,

$$f(n) = \max_{x \in \{0,1\}^n} T(x)$$

Example of worst-case complexity

Finding an element in an array.

Example of worst-case complexity

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

Example of worst-case complexity

Finding an element in an array.
Input: integer array $A[1 \dots n]$, 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?

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?

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-notation

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

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

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

E.g.

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

We write

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

O-notation

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

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 $1000000n$?
- n^{100} vs 2^n?
- $n^{\log n}$ vs 2^n?
- 2^{2^n} vs 2^n?

Ω-notation

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

Theorem

$f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$.

Ω-notation

$$
\begin{aligned}
\Omega(g(n)) = \{f(n) : \ & \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\
& 0 \le c \cdot g(n) \le f(n) \text{ for all } n \ge n_0\}
\end{aligned}
$$

Theorem

$f(n) = O(g(n))$ 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

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

- $f(n) = O(g(n))$
- $f(n) = \Omega(g(n))$

Examples:

- $n^2 + n + 5$ 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:

- $n^2 + n + 5$ vs $100n^2 + 5n + 3$?
- $n \cdot \log n$ vs $n^{1.0001}$?

o-notation

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

o-notation

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

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

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

o-notation

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

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

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

Examples:

- $100n = o(n^2)$

o-notation

$$
o(g(n)) = \{f(n) : \begin{array}{l} \text{for any positive constant } c > 0, \\ \text{there exists a constant } n_0 \text{ such that} \\ 0 \le f(n) < c \cdot g(n) \text{ for all } n \ge n_0\} \end{array}
$$

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

$$\omega(g(n)) = \{f(n) : \begin{array}{l} \text{for any positive constant } c > 0, \\ \text{there exists a constant } n_0 \text{ such that} \\ 0 \le c \cdot g(n) < f(n) \text{ for all } n \ge n_0\} \end{array}$$

ω-notation

$$\omega(g(n)) = \{f(n) : \begin{array}{l} \text{for any positive constant } c > 0, \\ \text{there exists a constant } n_0 \text{ such that} \\ 0 \le c \cdot g(n) < f(n) \text{ for all } n \ge n_0\} \end{array}$$

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

ω-notation

$$\omega(g(n)) = \{f(n) : \begin{array}{l} \text{for any positive constant } c > 0, \\ \text{there exists a constant } n_0 \text{ such that} \\ 0 \le c \cdot g(n) < f(n) \text{ for all } n \ge n_0\} \end{array}$$

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

Examples:

- 2^n vs n^{10}?

ω-notation

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

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

Examples:

- 2^n vs n^{10}?
- n vs $n \cdot \log n$?

ω-notation

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

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

Examples:

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