DSC 40B

Lecture 4 :

Big- Theta, Big-Oh and Omega, Formalized

Agenda

Plan for the lecture

- **Formally** define Θ, O, Ω notation.
- Some useful properties.

Formally define Θ, O, Ω notations

So Far

- Time Complexity Analysis: a picture of how an algorithm **scales**.
- Can use Θ-notation to express time complexity.
- Allows us to **ignore** details in a rigorous way:
  - **Saves us work!**
  - **But what exactly can we ignore?**

Theta Notation, Informally

- $\Theta(\cdot)$ **forgets** constant factors, lower-order terms.

$$5n^3 + 3n^2 + 42 = \Theta(n^3)$$

Theta Notation, Informally

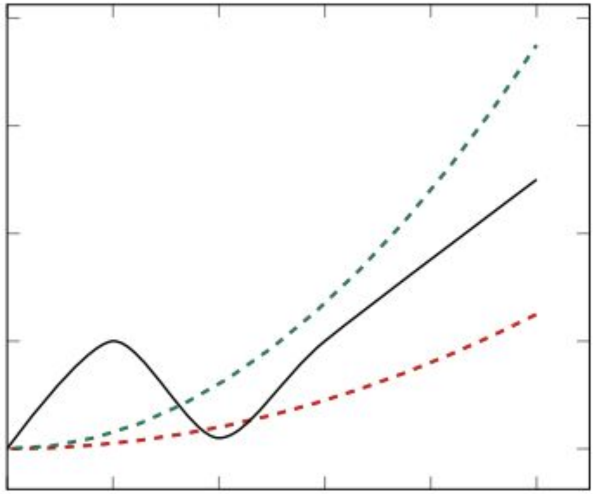
- $f(n) = \Theta(g(n))$ if $f(n)$ **"grows like"** $g(n)$.

  $5n^3 + 3n^2 + 42 = \Theta(n^3)$

Definition

We write $f(n) = \Theta(g(n))$ if there are **positive** constants N, c_1 and c_2 such that for all $n \geq N$:

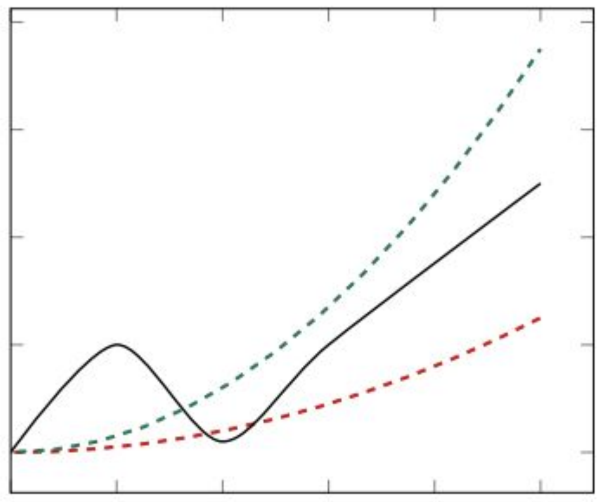
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$



Definition

We write $f(n) = \Theta(g(n))$ if there are **positive** constants N, c_1 and c_2 such that for all $n \geq N$:

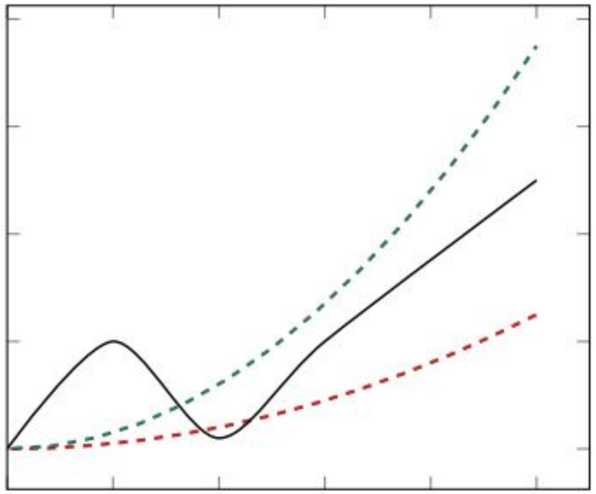
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$



Definition

We write $f(n) = \Theta(g(n))$ if there are **positive** constants N, c_1 and c_2 such that for all $n \geq N$:

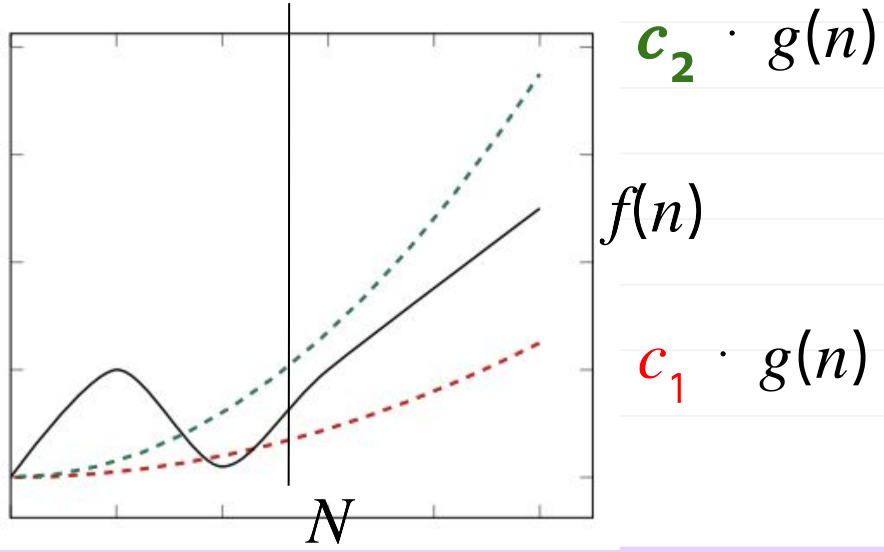
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$



Definition

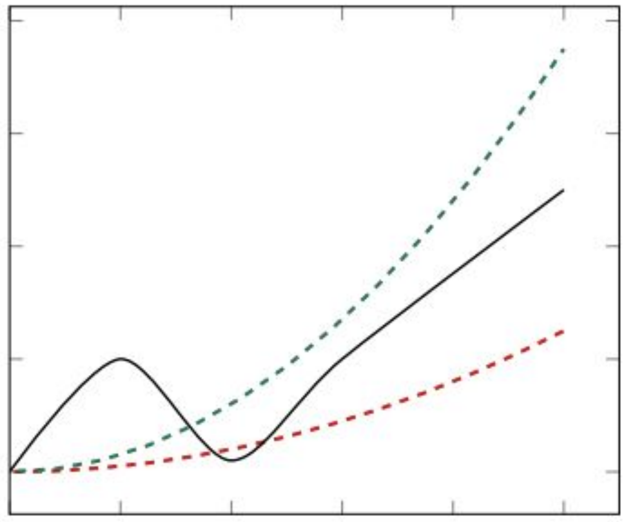
We write $f(n) = \Theta(g(n))$ if there are **positive** constants N, c_1 and c_2 such that for all $n \geq N$:

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$



Main Idea

If $f(n) = \Theta(g(n))$, then when $\boldsymbol{n}$ **is large** f is "*sandwiched*" between *copies* of g.



Proving Big-Theta

- We can prove that $f(n) = \Theta(g(n))$ by finding these constants.

$$c_1\, g(n) \le f(n) \le c_2\, g(n) \qquad (n \ge N)$$

- Requires a **lower bound** and an **upper bound.**

Strategy: Chains of Inequalities

- To show $f(n) \leq c_2 g(n)$, we show:

$$f(n) \leq \textit{(something)} \leq \textit{(another thing)} \leq \ldots \leq c_2\, g(n)$$

- At each step:
  - We can do anything to make value **larger**.
  - But the goal is to simplify it to look like $g(n)$.

Example

- Show that $4n^3 - 5n^2 + 50 = \Theta(n^3)$
- Find constants c_1 , c_2 , N such that for all $n > N$:

  $c_1 n^3 \leq 4n^3 - 5n^2 + 50 \leq c_2 n^3$
- They don't have to be the "best" constants! **Many solutions!**

Example

$$c_1 n^3 \leq 4n^3 - 5n^2 + 50 \leq c_2 n^3$$

- We want to make $4n^3 - 5n^2 + 50$ "look like" cn^3 .
- For the upper bound, can do anything that makes the function **larger**.
- For the lower bound, can do anything that makes the function **smaller**.

Example (n is a positive integer)

$$c_1 n^3 \leq 4n^3 - 5n^2 + 50 \leq c_2 n^3$$

- Upper bound:

$$4n^3 - 5n^2 + 50 \leq 4n^3 + 50 \leq 4n^3 + 50n^3 = 54n^3$$

Upper-Bounding Tips

- "Promote" lower-order **positive** terms:

$$3n^3 + 5n \leq 3n^3 + 5n^3$$

- "Drop" **negative** terms:

$$3n^3 - 5n \leq 3n^3$$

Example

$c_1 n^3 \leq 4n^3 - 5n^2 + 50 \leq c_2 n^3$

- Lower bound:

$4n^3 - 5n^2 + \mathbf{50} \geq 4n^3 - 5n^2$

$\geq 3n^3 + \mathbf{(n^3 - 5n^2)} \geq 3n^3$ (n >=5)

True WHEN $(n^3 - 5n^2) >= 0$ => n>=5

Lower-Bounding Tips

- "Drop" lower-order **positive** terms:

  $3n^3 + 5n \geq 3n^3$

- "Promote and cancel" **negative** lower-order terms if possible:

  $4n^3 - 2n \geq 4n^3 - 2n^3 = 2n^3$

Lower-Bounding Tips

- "Cancel" negative lower-order terms with big constants by "breaking off" a piece of high term.

$$4n^3 - 10n^2 = (3n^3 + n^3) - 10n^2$$

Lower-Bounding Tips

- "Cancel" negative lower-order terms with big constants by "breaking off" a piece of high term.

$$4n^3 - 10n^2 = (3n^3 + n^3) - 10n^2 = 3n^3 + (n^3 - 10n^2)$$

Lower-Bounding Tips

- “Cancel” negative lower-order terms with big constants by “breaking off” a piece of high term.

$$4n^3 - 10n^2 = (3n^3 + n^3) - 10n^2 = 3n^3 + (n^3 - 10n^2)$$

$n^3 - 10n^2 \geq 0$ when

Lower-Bounding Tips

- "Cancel" negative lower-order terms with big constants by "breaking off" a piece of high term.

$$4n^3 - 10n^2 = (3n^3 + n^3) - 10n^2 = 3n^3 + (n^3 - 10n^2)$$

$$n^3 - 10n^2 \geq 0 \text{ when } n^3 \geq 10n^2 \Longrightarrow n \geq 10:$$

Lower-Bounding Tips

- "Cancel" negative lower-order terms with big constants by "breaking off" a piece of high term.

$$4n^3 - 10n^2 = (3n^3 + n^3) - 10n^2 = 3n^3 + (n^3 - 10n^2)$$

$$n^3 - 10n^2 \geq 0 \text{ when } n^3 \geq 10n^2 \Longrightarrow n \geq 10:$$

$$\geq 3n^3 + 0 \quad (n \geq 10)$$

Example

$$c_1 n^3 \leq 4n^3 - 5n^2 + 50 \leq c_2 n^3$$

- We want to make $4n^3 - 5n^2 + 50$ "look like" cn^3 .
- For the upper bound, can do anything that makes the function **larger**.
- For the lower bound, can do anything that makes the function **smaller**.

Example

$3n^3 \leq 4n^3 - 5n^2 + 50 \leq 54n^3$ (n >= 5) (**N is 5**)

- We want to make $4n^3 - 5n^2 + 50$ "look like" cn^3 .
- For the upper bound, can do anything that makes the function **larger**.
- For the lower bound, can do anything that makes the function **smaller**.

$n^2 <= 2n^2 - n <= 2n^2 + n \sin 3n <= 2n^2 + n <= 3n^2$

Exercise

True or False: $2n^2 + n \sin 3n = \Theta(n^2)$.

A: True

B: False

C: Impossible to determine

Exercise

True or False: $f(n) = \Theta(n^2)$

Let

$$f(n) = \begin{cases} n^2 & \text{if } n \text{ is even} \\ 5 & \text{if } n \text{ is odd.} \end{cases}$$

A: True

B: False

C: Impossible to determine

Big-Oh (Big-O) and Big-Omega

Other Bounds

- $f = \Theta(g)$ means that f is both **upper** and **lower** bounded by factors of g.
- Sometimes we only have (or care about) upper bound or lower bound.
- We have notation for that, too.

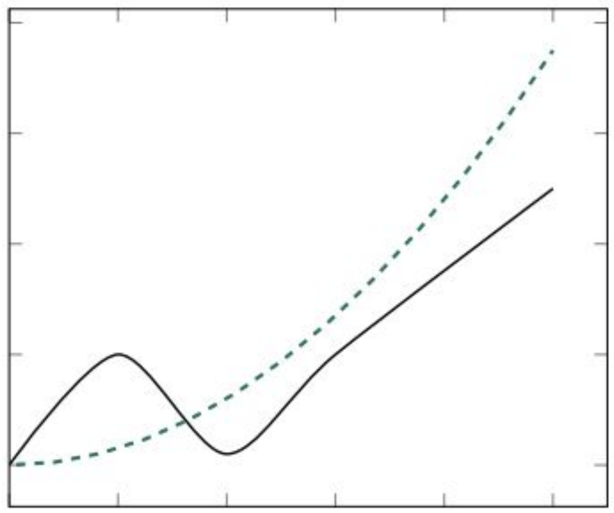
Big-O Notation, Informally

- Sometimes we only care about **upper bound.**
- $f(n) = O(g(n))$ if f "grows at most as fast" as g.
- **Examples**:
  - $4n^2 = O(n^{100})$
  - $4n^2 = O(n^3)$
  - $4n^2 = O(n^2)$ and $4n^2 = \Theta(n^2)$

Formal Definition

- We write $f(n) = O(g(n))$ if there are positive constants $\boldsymbol{N}$ and $\boldsymbol{c}$ such that for all $n \geq N$:

$$f(n) \leq \boldsymbol{c} \cdot g(n)$$



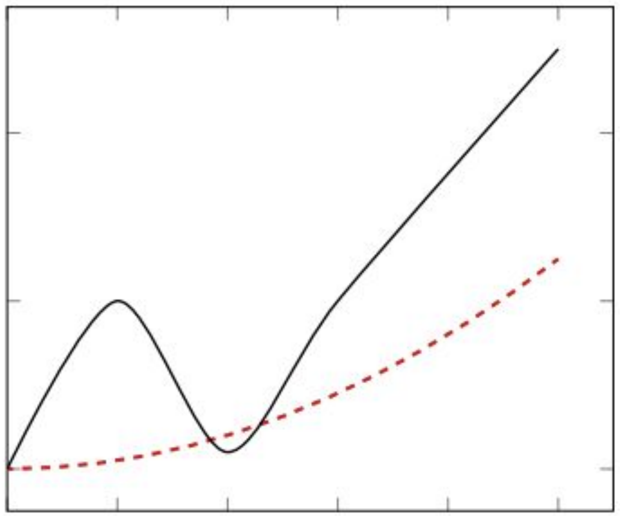
Big-Omega Notation, Informally

- Sometimes we only care about **lower bound.**
- $f(n) = \Omega(g(n))$ if f "grows at least as fast" as g.
- **Examples**:
  - $4n^{100} = \Omega(n^5)$
  - $4n^2 = \Omega(n)$
  - $4n^2 = \Omega(n^2)$ *and* $4n^2 = \Theta(n^2)$

Formal Definition

- We write $f(n) = \Omega(g(n))$ if there are positive constants $\boldsymbol{N}$ and $\boldsymbol{c}$ such that for all $n \geq N$:

$$\boldsymbol{c} \cdot g(n) \leq f(n)$$



FUN FACT

“Omega” in Greek literally means: big O.
So translated, “Big-Omega” means “big big O”.

Theta, Big-O, and Big-Omega

- If $f = \Theta(g)$ then $f = O(g)$ and $f = \Omega(g)$.
- If $f = O(g)$ and $f = \Omega(g)$ then $f = \Theta(g)$.
- Pictorially:
  - $\Theta \Longrightarrow (O \text{ and } \Omega)$
  - $(O \text{ and } \Omega) \Longrightarrow \Theta$

Analogies

- Θ is kind of like =
- O is kind of like ≤ (at most)
- Ω is kind of like ≥ (at least)

Practice: true/false

Let

$$f(n) = \begin{cases} n^2 & \text{if } n \text{ is even} \\ 5 & \text{if } n \text{ is odd.} \end{cases}$$

- True or False: $f(n) = O(n^2)$.
- True or False: $f(n) = \Omega(n^2)$.
- True or False: $f(n) = \Omega(1)$.

A: True

B: False

Why?

mic

- Laziness.
- Sometimes finding an upper or lower bound would take **too much work**, and/or we don’t really care about it anyways.

Big-Oh

- Often used when another part of the code would dominate time complexity anyways.

Exercise: what is the complexity?

```
def foo(n):
    for a in range(n**4):
        print(a)

    for i in range(n):
        for j in range(i**2):
            print(i + j)
```

Exercise: what is the complexity?

```
def foo(n):
    for a in range(n**4):
        print(a)

    for i in range(n):
        for j in range(i**2):
            print(i + j)
```

$\Theta(n^4)$

Exercise: what is the complexity?

```
def foo(n):
    for a in range(n**4):
        print(a)

    for i in range(n):
        for j in range(i**2):
            print(i + j)
```

First loop: $\Theta(n^4)$

Second loop: $O(n^3)$

Big-Omega

- Often used when the time complexity will be **so large** that we don’t care what it is, exactly.

Example: Big-Omega

```
best_separation = float('inf')
best_clustering = None

for clustering in all_clusterings(data):
    sep = calculate_separation(clustering)
    if sep < best_separation:
        best_separation = sep
        best_clustering = clustering

print(best_clustering)
```

Example: Big-Omega

```
best_separation = float('inf')
best_clustering = None

for clustering in all_clusterings(data):
    sep = calculate_separation(clustering)
    if sep < best_separation:
        best_separation = sep
        best_clustering = clustering

print(best_clustering)
```

We have 2^n pairs

Example: Big-Omega

```
best_separation = float('inf')
best_clustering = None

for clustering in all_clusterings(data):
    sep = calculate_separation(clustering)
    if sep < best_separation:
        best_separation = sep
        best_clustering = clustering

print(best_clustering)
```

We have 2^n pairs

$\Theta(?)$

Example: Big-Omega

```
best_separation = float('inf')
best_clustering = None

for clustering in all_clusterings(data):
    sep = calculate_separation(clustering)
    if sep < best_separation:
        best_separation = sep
        best_clustering = clustering

print(best_clustering)
```

We have 2^n pairs

$\Theta(?)$

Example: Big-Omega

```
best_separation = float('inf')
best_clustering = None

for clustering in all_clusterings(data):
    sep = calculate_separation(clustering)
    if sep < best_separation:
        best_separation = sep
        best_clustering = clustering

print(best_clustering)
```

We have 2^n pairs

Θ(?)

$T(n) = \Omega(2^n)$

Other Notations

- $f(n) = o(g(n))$ if f grows *"much slower"* than g.
  - Whatever c you choose, eventually $f < cg(n)$.
  - Example: $n^2 = o(n^3)$

Other Notations

- $f(n) = o(g(n))$ if f grows *"much slower"* than g.
  - Whatever c you choose, eventually $f < cg(n)$.
  - Example: $n^2 = o(n^3)$

- $f(n) = \omega(g(n))$ if f grows *"much faster"* than g
  - Whatever c you choose, eventually $f > cg(n)$.
  - Example: $n^3 = \omega(n^2)$

Other Notations

- $f(n) = o(g(n))$ if f grows *"much slower"* than g.
  - Whatever c you choose, eventually $f < cg(n)$.
  - Example: $n^2 = o(n^3)$
- $f(n) = \omega(g(n))$ if f grows *"much faster"* than g
  - Whatever c you choose, eventually $f > cg(n)$.
  - Example: $n^3 = \omega(n^2)$
- **We won't really use these.**

Properties

Properties

- We don’t usually go back to the definition when using Θ.
- Instead, we use a few basic ***properties***.

Properties of Θ

- **Symmetry**: If $f = \Theta(g)$, then $g = \Theta(f)$.
- **Transitivity**: If $f = \Theta(g)$ and $g = \Theta(h)$ then $f = \Theta(h)$.
- **Reflexivity**: $f = \Theta(f)$

Practice: T/F

- If $f = O(g)$ and $g = O(h)$, then $f = O(h)$.
- If $f = \Omega(h)$ and $g = \Omega(h)$, then $f = \Omega(g)$.
- If $f_1 = \Theta(g_1)$ and $f_2 = O(g_2)$, then $f_1 + f_2 = \Theta(g_1 + g_2)$.
- If $f_1 = \Theta(g_1)$ and $f_2 = \Theta(g_2)$, then $f_1 \times f_2 = \Theta(g_1 \times g_2)$.

A: F, F, F, T

B: F, T, F, F

C: T, F, F, T

D: T, T, T, T

E: None of the above

Practice: T/F

- If $f = O(g)$ and $g = O(h)$, then $f = O(h)$.
- If $f = \Omega(h)$ and $g = \Omega(h)$, then $f = \Omega(g)$.
- If $f_1 = \Theta(g_1)$ and $f_2 = O(g_2)$, then $f_1 + f_2 = \Theta(g_1 + g_2)$.
- If $f_1 = \Theta(g_1)$ and $f_2 = \Theta(g_2)$, then $f_1 \times f_2 = \Theta(g_1 \times g_2)$.

A: F, F, F, T

B: F, T, F, F

C: T, F, F, T

D: T, T, T, T

E: None of the above

Proving/Disproving Properties

- Start by trying to **disprove**.
- Easiest way: find a counterexample.
- **Example**: If $f = \Omega(h)$ and $g = \Omega(h)$, then $f = \Omega(g)$.
  - False! Let $f = n^3$, $g = n^5$, and $h = n^2$.

Proving the Property

- If you can't disprove, *maybe* it is true.
- **Example**:
  - Suppose $f_1 = O(g_1)$ and $f_2 = O(g_2)$.
  - Prove that $f_1 \times f_2 = O(g_1 \times g_2)$.

Step 1: State the assumption

- We know that $f_1 = O(g_1)$ and $f_2 = O(g_2)$.
- So there are constants c_1, c_2, N_1, N_2 so that for all $n \geq N_1$ and $n \geq N_2$:

$$f_1(n) \leq c_1 g_1(n) \qquad (n \geq N_1)$$

$$f_2(n) \leq c_2 g_2(n) \qquad (n \geq N_2)$$

Prove that $f_1 \times f_2 = O(g_1 \times g_2)$.

Use the assumption

- Chain of inequalities, starting with $f_1 \times f_2$, ending with $\leq c\, g_1 \times g_2$.
- Using the following piece of information:

$$f_1(n) \leq c_1 g_1(n) \qquad (n \geq N_1)$$

$$f_2(n) \leq c_2 g_2(n) \qquad (n \geq N_2)$$

Prove that $f_1 \times f_2 = O(g_1 \times g_2)$.

Use the assumption

- Chain of inequalities, starting with $f_1 \times f_2$, ending with $\le c\, g_1 \times g_2$.
- Using the following piece of information:

$$\boldsymbol{f_1(n) \le c_1 g_1(n)} \qquad (n \ge N_1)$$

$$f_2(n) \le c_2 g_2(n) \qquad (n \ge N_2)$$

$f_1(n) \times f_2(n) \le$ …………..≤……………..≤ $c\, g_1 \times g_2$

Prove that $f_1 \times f_2 = O(g_1 \times g_2)$.

Use the assumption

- Chain of inequalities, starting with $f_1 \times f_2$, ending with $\leq c\, g_1 \times g_2$.
- Using the following piece of information:

$$\boldsymbol{f_1(n) \leq c_1 g_1(n)} \quad (n \geq N_1)$$

$$f_2(n) \leq c_2 g_2(n) \quad (n \geq N_2)$$

$$f_1(n) \times f_2(n) \leq \boldsymbol{c_1 g_1(n)} \times f_2(n) \quad (n \geq N_1)$$

Prove that $f_1 \times f_2 = O(g_1 \times g_2)$.

Use the assumption

- Chain of inequalities, starting with $f_1 \times f_2$, ending with $\leq c\, g_1 \times g_2$.
- Using the following piece of information:

$$f_1(n) \leq c_1 g_1(n) \qquad (n \geq N_1)$$

$$\mathbf{f_2(n) \leq c_2 g_2(n)} \qquad (n \geq N_2)$$

$$f_1(n) \times f_2(n) \leq c_1 g_1(n) \times \mathbf{f_2(n)} \qquad (n \geq N_1)$$

$$\leq c_1 g_1(n) \times c_2 g_2(n) \qquad (n \geq N_1) \text{ and } (n \geq N_2)$$

Prove that $f_1 \times f_2 = O(g_1 \times g_2)$.

Use the assumption

- Chain of inequalities, starting with $f_1 \times f_2$, ending with $\leq c\, g_1 \times g_2$.
- Using the following piece of information:

$f_1(n) \leq c_1 g_1(n) \quad (n \geq N_1)$

$\boldsymbol{f_2(n) \leq c_2 g_2(n)} \quad (n \geq N_2)$

$f_1(n) \times f_2(n) \leq c_1 g_1(n) \times f_2(n) \qquad (n \geq N_1)$

$\leq c_1 g_1(n) \times c_2 g_2(n) \qquad (n \geq N_1)$ and $(n \geq N_2)$

$\leq \boldsymbol{c}\, g_1(n) \times g_2(n) \qquad (n \geq \max(N_1, N_2))$ and $\boldsymbol{c = c_1 \cdot c_2}$

Analyzing Code

- The properties of Θ (and O and Ω) are useful when analyzing code.
- We can analyze pieces, put together the results.

Sums of Theta

- Property: If $f_1 = \Theta(g_1)$ and $f_2 = \Theta(g_2)$, then $f_1 + f_2 = \Theta(g_1 + g_2)$
- Used when analyzing sequential code.

Example

```
def foo(n):
    bar(n)
    baz(n)
```

- Say bar takes $\Theta(n^3)$,
- baz takes $\Theta(n^4)$.
- foo takes $\Theta(n^4 + n^3) = \Theta(n^4)$.
- baz is the **bottleneck**.

Products of Theta

- Property: If $f_1 = \Theta(g_1)$ and $f_2 = \Theta(g_2)$, then

$$f_1 \cdot f_2 = \Theta(g_1 \cdot g_2)$$

- Useful when analyzing **nested loops.**

Example

```
def foo(n):
    for i in range(3*n + 4, 5n**2 - 2*n + 5):
        for j in range(500*n, n**3):
            print(i, j)
```

Example

```
def foo(n):
    for i in range(3*n + 4, 5n**2 - 2*n + 5):
        for j in range(500*n, n**3):
            print(i, j)
```

A: $\Theta(n^5)$

B: $\Theta(n^2)$

C: $\Theta(n^4)$

D: $\Theta(n^3)$

E: Something else

Careful!

If inner loop index **depends** on outer loop, you have to be more careful.

```
def foo(n):
    for i in range(n):
        for j in range(i):
            print(i, j)
```

Caution

- To upper bound a fraction A/B, you must:
  - Upper bound the numerator, A.
  - Lower bound the denominator, B.
- And to lower bound a fraction A/B, you must:
  - Lower bound the numerator, A.
  - Upper bound the denominator, B.

Caution

- To upper bound a fraction A/B, you must:
  - Upper bound the numerator, A.
  - Lower bound the denominator, B.
- **Example**:
  - What is larger? ⅗ or ⅘ ?
    - ⅘ since numerator is larger (4 > 3)
  - What is larger? 4/3 or ⅘?
    - 4/3 since the denominator is smaller (3 < 4)

Caution

- To upper bound a fraction A/B, you must:
  - Upper bound the numerator, A.
  - Lower bound the denominator, B.
- To make a fraction **as large as possible**, you should:
  - Make the **numerator (A)** as **large as possible** → that's upper bounding A.
  - Make the **denominator (B)** as **small as possible** → that's lower bounding B.

Caution

- To make a fraction **as large as possible**, you should:
  a. Make the **numerator (A)** as **large as possible** → that's upper bounding A.
  b. Make the **denominator (B)** as **small as possible** → that's lower bounding B.

If **A=3x+2** and **B=x+1** and you know that **1≤x≤4**

What is the best upper bound for A/B?

A: 6

B: 5/4

C: 7

D: 14

Asymptotic Notation Practicalities

In this part...

- Other ways asymptotic notation is used.
- Asymptotic notation slip ups.
- Downsides of asymptotic notation.

Not Just for Time Complexity!

- We most often see asymptotic notation used to express time complexity.
- But it can be used to express any type of growth!

Example: Combinatorics

- Recall: $\binom{n}{k}$ is number of ways of choosing k things from a set of n.
- How fast does this grow with n? For fixed k:

$$\binom{n}{k} = \Theta(n^k)$$

- **Example**: the number of ways of choosing 3 things out of n is $\Theta(n^3)$.

Example: Central Limit Theorem

- Recall (DSC10): the CLT says that the sample mean has a normal distribution with standard deviation $\sigma_{\text{pop}}/\sqrt{n}$
- The **error** in the sample mean is: $O(1/\sqrt{n})$

Common Slip-ups

- Asymptotic notation can be used **improperly**.
  - Might be technically correct, but defeats the purpose.
- Don’t do these in, e.g., interviews!

Slip-up #1

- Don't include constants, lower-order terms in the notation.
- **Bad**: $3n^2 + 2n + 5 = \Theta(3n^2)$.
- **Good**: $3n^2 + 2n + 5 = \Theta(n^2)$.
- It isn't *wrong* to do so, just defeats the purpose.

Slip-up #2

- Don't include base in logarithm.
- **Bad**: $\Theta(\log_2 n)$
- **Good**: $\Theta(\log n)$
- Why? $\log_2 n = c \cdot \log_3 n = c' \log_4 n = \ldots$

Slip-up #3

- Don't misinterpret meaning of $\Theta(\cdot)$.
- $f(n) = \Theta(n^3)$ does **not** mean that there are constants so that $f(n) = c_3 n^3 + c_2 n^2 + c_1 n + c_0$.

Slip-up #4

- Time complexity is not a **complete** measure of efficiency.
- $\Theta(n)$ is not *always* "better" than $\Theta(n^2)$.
- Why?

Slip-up #4

- **Why?** Asymptotic notation "hides the constants".
- $T_1(n) = 1{,}000{,}000n = \Theta(n)$
- $T_2(n) = 0.00001n^2 = \Theta(n^2)$
- But $T_1(n)$ is **worse** for all but really large n.

Main Idea

- TIme Complexity is **not** the only way to measure efficiency, and it can be misleading.
- Sometimes even a $\Theta(2^n)$ algorithm is better than a $\Theta(n)$ algorithm, if the data size is small.

Thank you!

Do you have any questions?

CampusWire!