DSC 40B

Theoretical Foundations II

Lecture 2 | Part 1

News

News

- Homework 01 posted, due Tuesday @ 11:59 pm PST on Gradescope.
- LaTeX template available.

DSC 40B

Theoretical Foundations II

Lecture 2 | Part 2

Nested Loops

Example: Interview Problem

Example: Interview Problem

- Design an algorithm to solve the following problem...
- Given the heights of n people, what is the height of the tallest doctor you can make by stacking two of them?

Exercise

Design a brute force solution to the problem. What is its time complexity?

Time/exec. **# of execs.**

```python
def tallest_doctor(heights):
    max_height = -float('inf')
    n = len(heights)
    for i in range(n):
        for j in range(n):
            if i == j:
                continue
            height = heights[i] + heights[j]
            if height > max_height:
                max_height = height
    return max_height
```

Time Complexity

- Time complexity of this is $\Theta(n^2)$.
- Note: this algorithm considers each pair of people **twice**.

Shortcut

- Making a table is getting tedious.
- Usually we'll find a line that **dominates** time complexity; i.e., yields the leading term of $T(n)$.

Shortcut

```
def f(n):
    for i in range(10*n, 3*n**3 + 5*n**2 - 100):
        for j in range(n**5, n**6):
            print(i, j)
```

Example: Recall from 40A

- **Given:** real numbers $x_1, \ldots, x_n$.
- **Compute:** h minimizing the **total absolute loss**

$$R(h) = \sum_{i=1} |x_i - h|$$

Example: Recall from 40A

- **Solution**: the **median**.
- That is, a **middle** number.
- But how do we actually **compute** a median?

A Strategy

- **Recall**: one of $x_1, \dots, x_n$ must be a median.
- **Idea**: compute $R(x_1), R(x_2), \dots, R(x_n)$, return x_i that gives the smallest result.
- Basically a **brute force** approach.

Exercise

Write a function which computes a median using this strategy. What is its time complexity?

```python
def median(numbers):
    min_h = None
    min_value = float('inf')
    for h in numbers:
        total_abs_loss = 0
        for x in numbers:
            total_abs_loss += abs(x - h)
        if total_abs_loss < min_value:
            min_value = total_abs_loss
            min_h = h
    return min_h
```

The Median

- The brute force approach has $\Theta(n^2)$ time complexity.
- Is there a better algorithm?

Careful!

- Not every nested loop has $\Theta(n^2)$ time complexity!

Example

```
def foo(n):
    for x in range(n):
        for y in range(10):
            print(x + y)
```

Example: Tallest Doctor, Again

- Our previous algorithm for the tallest doctor computed height for each pair of people **twice**.
  - `i = 3` and `j = 7` is the same as `i = 7` and `j = 3`
- **Idea**: consider each pair only once:

```
for i in range(n):
    for j in range(i + 1, n):
```

```python
1 def tallest_doctor_2(heights):
2     max_height = -float('inf')
3     n = len(heights)
4     for i in range(n):
5         for j in range(i + 1, n):
6             height = heights[i] + height[j]
7             if height > max_height:
8                 max_height = height
```

How many times does line 6 run in total?

```python
for i in range(n):
    for j in range(i + 1, n):
        height = heights[i] + heights[j]
```

- On outer iter. # 1, inner body runs _______ times.
- On outer iter. # 2, inner body runs _______ times.
- On outer iter. # k, inner body runs _______ times.
- The outer loop body runs _______ times.

$$\underbrace{(n-1)}_{\text{1st outer iter}} + \underbrace{(n-2)}_{\text{2nd outer iter}} + \ldots + \underbrace{(n-k)}_{\text{kth outer iter}} + \underbrace{(n-(n-1)}_{\text{(n-1)th outer iter}} + \underbrace{(n-n)}_{\text{nth outer iter}}$$

=

$$1 + 2 + 3 + \ldots + (n-3) + (n-2) + (n-1)$$

=

Time Complexity

- `tallest_doctor_2` has $\Theta(n^2)$ time complexity
- Same as original `tallest_doctor`!

Exercise

Should we have been able to guess this? Why?

Number of Pairs

- Recall from 40A: number of pairs of n objects is

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$$

Exercise

Design a linear time algorithm for this problem.

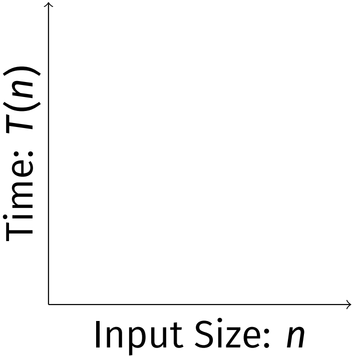
DSC 40B

Theoretical Foundations II

Lecture 2 | Part 3

Linear vs. Quadratic

Scaling



- $T(n) = \Theta(n)$ means "$T(n)$ grows like n"
- $T(n) = \Theta(n^2)$ means "$T(n)$ grows like n^2"

Definition

An algorithm is said to run in **linear time** if $T(n) = \Theta(n)$.

Definition

An algorithm is said to run in **quadratic time** if $T(n) = \Theta(n^2)$.

Linear Growth

- If input size doubles, time *doubles*.
- If code takes 5 seconds on 1,000 points...
- ...on 100,000 data points it takes ≈ 500 seconds.
- i.e., 8.3 minutes

Quadratic Growth

- If input size doubles, time *quadruples*.
- If code takes 5 seconds on 1,000 points...
- ...on 100,000 points it takes ≈ 50,000 seconds.
- i.e., ≈ 14 hours

Common Growth Rates

- $\Theta(1)$: **constant**
- $\Theta(\log n)$: **logarithmic**
- $\Theta(n)$: **linear**
- $\Theta(n \log n)$: **linearithmic**
- $\Theta(n^2)$: **quadratic**
- $\Theta(n^3)$: **cubic**
- $\Theta(2^n)$: **exponential**

Next Time

- What does $\Theta(\cdot)$ notation mean, exactly?

DSC 40B

Theoretical Foundations II

Lecture 2 | Part 4

Big Theta

Last Time

- 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?**

Today

- A deeper look at **asymptotic notation**:
  - What does $\Theta(\cdot)$ mean, exactly?
  - Related notations: $O(\cdot)$ and $\Omega(\cdot)$.
  - How these notations save us work.

Theta Notation, Informally

- $\Theta(\cdot)$ forgets constant factors, lower-order terms.
- $f(n) = \Theta(g(n))$ if $f(n)$ "grows like" $g(n)$.

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

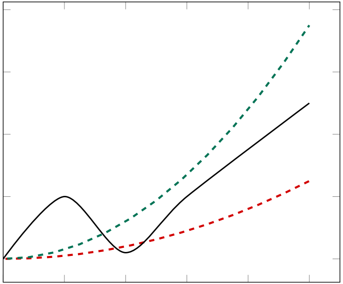
Theta Notation Examples

- $4n^2 + 3n - 20 = \Theta(n^2)$
- $3n + \sin(4\pi n) = \Theta(n)$
- $2^n + 100n = \Theta(2^n)$

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 f can be "sandwiched" between copies of g when n is large.

Example

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

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

- They don't have to be the "best" constants!

Strategy 1: Guess and Check

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

- Guess $c_1 = 1, c_2 = 100$. Solve for N.
- Prove upper bound, lower bound, then combine.

Strategy 1: Guess and Check

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

- Upper bound:

Strategy 1: Guess and Check

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

- Lower bound:

Strategy 1: Guess and Check

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

- Combine:

Strategy 2: Be More Clever

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

- We want to make $4n^2 - 5n^2 + 50$ “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**.

Strategy 2: Be More Clever

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

- Upper bound:

Strategy 2: Be More Clever

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

- Lower bound:

DSC 40B

Theoretical Foundations II

Lecture 2 | Part 5

Big-Oh 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.

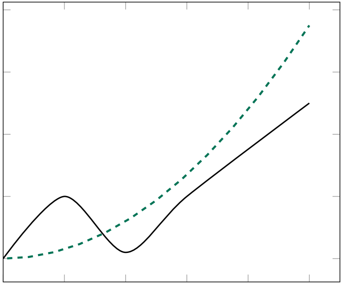
Big-O Notation, Informally

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

Definition

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

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



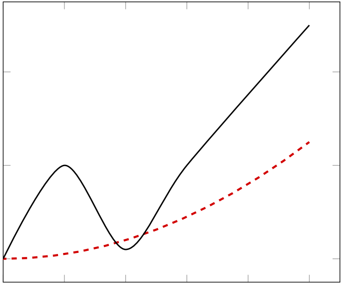
Big-Omega Notation

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

Definition

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

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



FUN FACT

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

Why?

- 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.

Example: Big-Oh

```python
def tallest_doctor(heights):
    max_height = -float('inf')
    n = len(heights)
    for i in range(n):
        for j in range(n):
            if i == j:
                continue
            height = heights[i] + heights[j]
            if height > max_height:
                max_height = height
    return max_height
```

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)
```

DSC 40B

Theoretical Foundations II

Lecture 2 | Part 6

Limit Definitions

Limits and Θ, O, Ω

- You might prefer to use limits when reasoning about asymptotic notation.
- **Warning!** There are some tricky subtleties.
- Be able to "fall back" to the formal definitions.

Theta and Limits

- **Claim**: If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, then $f(n) = \Theta(g(n))$.

- Example: $4n^3 - 5n^2 + 50$.

Warning!

- Converse **isn't true**: if $f(n) = \Theta(g(n))$, it need not be that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$.
- The limit can be **undefined**.
- Example: $5 + \sin(n) = \Theta(1)$, but the limit d.n.e.

Big-O and Limits

- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$, then $f(n) = O(g(n))$.
- Namely, the limit can be zero. e.g., $n = O(n^2)$.
- **Warning!** Converse not true. Limit may not exist.

Big-Omega and Limits

- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$, then $f(n) = O(g(n))$.
- Namely, the limit can be ∞. e.g., $n^2 = \Omega(n)$.
- **Warning!** Converse not true. Limit may not exist.

Good to Know

- $\log_b n$ grows slower than n^p, as long as $p > 0$.
- Example:

$$\lim_{n \to \infty} \frac{\log_2 n}{n^{0.000001}} = 0$$

Exercise

Which grows faster, $n!$ or 2^n?