

DSC40B: Theoretical Foundations of Data Science II

Lecture 2: *Nested loops and
asymptotic time complexity*

Instructor: Yusu Wang

More examples: Nest loops



The median problem

- ▶ Design an algorithm for the following problem
 - ▶ Input: given n numbers $X = \{x_1, x_2, \dots, x_n\}$
 - ▶ Output: find h minimizing the total absolute loss:
 - ▶ $Loss(h) = \sum_{i=1}^n |x_i - h|$
- ▶ Recall from DSC40A
 - ▶ Solution: h^* is a median of numbers in X
 - ▶ i.e, the number whose order in X (if sorted) is $\lfloor \frac{n}{2} \rfloor$ or $\lceil \frac{n}{2} \rceil$
- ▶ Question:
 - ▶ How to compute this median?



One algorithm

- ▶ Idea:
 - ▶ h^* has to be one of $X = \{x_1, x_2, \dots, x_n\}$
 - ▶ compute all $\text{Loss}(x_1), \dots, \text{Loss}(x_n)$
 - ▶ return the one whose loss is smallest



One algorithm

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

Inner-for loop:

Outer-for
loop:

Total:



One algorithm

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

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

We will see how
to do better later
in class.



Caution!

- ▶ Not all nested loop takes $\Theta(n^2)$ time

```
def foo_1(n):
    for x in range(n):
        for y in range(n, n+10):
            print(x + y)
```

$$T_1(n) = \Theta(n)$$

```
def foo_2(n):
    for x in range(n):
        for y in range(n, 2n-10):
            print(x + y)
```

$$T_2(n) = \Theta(n^2)$$



Another example

- ▶ Alex is given n sticks. He needs to design an algorithm to compute the tallest pole he can make by stacking two sticks

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

On outer iter.#1, inner body runs _____ times

On outer iter.#2, inner body runs _____ times

On outer iter.#3, inner body runs _____ times

⋮

On outer iter.#n, inner body runs _____ times

For the k -th iteration of outer loop, the inner loop takes $c(n - k)$ time.

- ▶ How many times does line 5 – 7 (inner body) run?

Tallest_pole, cont.

- ▶ Total times line 5-7 (inner body) is executed:

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

=

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

=

- ▶ Algorithm **tallest_pole** has $T(n) = \Theta(n^2)$

- ▶ Another way to see the time complexity:

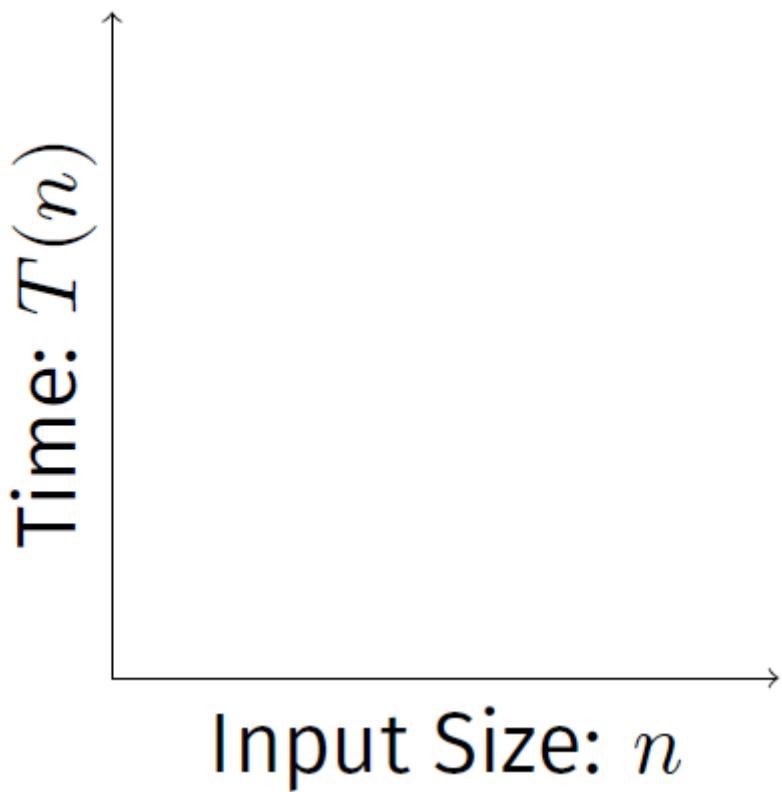
- ▶ Number of pairs of n objects is $\binom{n}{2} = \frac{n(n-1)}{2} = \Theta(n^2)$

Exercise: Find a linear-time algorithm for this problem.

Linear vs quadratic growth



Scaling



- ▶ $T(n) = \Theta(n)$
 - ▶ means " $T(n)$ grows like n "
 - ▶ linear growth

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



-
- ▶ Suppose your algorithm runs 5 sec on 1000 points
 - ▶ Linear growth
 - ▶ If the input has 100,000 points, then it takes 500 seconds (8.3 min)
 - ▶ Quadratic growth
 - ▶ If the input has 100,000 points, then it will take 50,000 seconds (~14 hours)



Some 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



Asymptotic notations



-
- ▶ We have seen $\Theta(\cdot)$ allows us to ignore unnecessary details and focus on dominating terms of growth.
 - ▶ Simpler, easier to analyze, and focus on key growth rate
 - ▶ What exactly are we ignoring?
 - ▶ Before we introduce this formally
 - ▶ First introduce big- O and big- Ω notations
 - ▶ Then big- Θ
 - ▶ Intuitively, big- O , big- Ω , and big- Θ “roughly” corresponds to \leq , \geq , and $=$, respectively (upto constants)
-

Big-O notation



Big-O notation

Definition

We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that for all $n \geq n_0$:

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

- ▶ More precisely, should be $f(n) \in O(g(n))$
- ▶ Intuitively, $f(n) = O(g(n))$ means that
 - ▶ $f(n)$ grows at most as fast as $g(n)$ (up to multiplicative constant factor)
 - ▶ We also say $g(n)$ is **an asymptotic upper bound** for $f(n)$ in this case

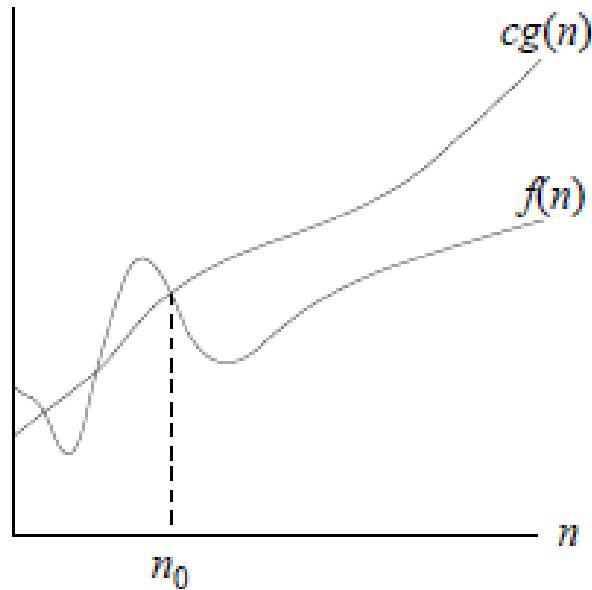


Big-O notation

Definition

We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that for all $n \geq n_0$:

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



Examples

► $n^3 - 3n^2 + 5n - 1 = O(n^3)$

► Proof:

► $5n^2 + 6\sqrt{n} + 8 = O(n^2)?$

► Proof:



More examples

- ▶ $\sqrt{6n^3 + 7n^2 + 3n} = O(n^2)$?
- ▶ $\sqrt{6n^3 + 7n^2 + 3n} = O(n^{1.5})$?
- ▶ $\frac{n}{\lg n} = O(n)$?
- ▶ $\frac{100 n}{\lg n} = O\left(\frac{n}{\lg n}\right)$?
- ▶ $n \lg n = O(n)$?
- ▶ $\log_2 n = O(\log_{10} n)$?



A useful result

Lemma [Upper Bound]

If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ exists, then $f(n) = O(g(n))$ if and only if

$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq c$, where c is a positive constant.

Corollary [Upper Bound]

If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = O(g(n))$.

If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = +\infty$, then $f(n) = O(g(n))$ does not hold.



More examples

- ▶ $n^{100} = O(n^2)$?
- ▶ $50 \lg n = O(n)$?
- ▶ $n^{100} = O(2^n)$?
- ▶ $2^n = O(3^n)$?



Big- Ω notation



Big- Ω notation

Definition

We write $f(n) = \Omega(g(n))$ if there are **positive constants** n_0 and c such that for all $n \geq n_0$:

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

- ▶ More precisely, should be $f(n) \in \Omega(g(n))$
- ▶ Intuitively, $f(n) = \Omega(g(n))$ means that
 - ▶ $f(n)$ grows at least as fast as $g(n)$ (up to multiplicative constant factor)
 - ▶ We also say $g(n)$ is **an asymptotic lower bound** for $f(n)$ in this case

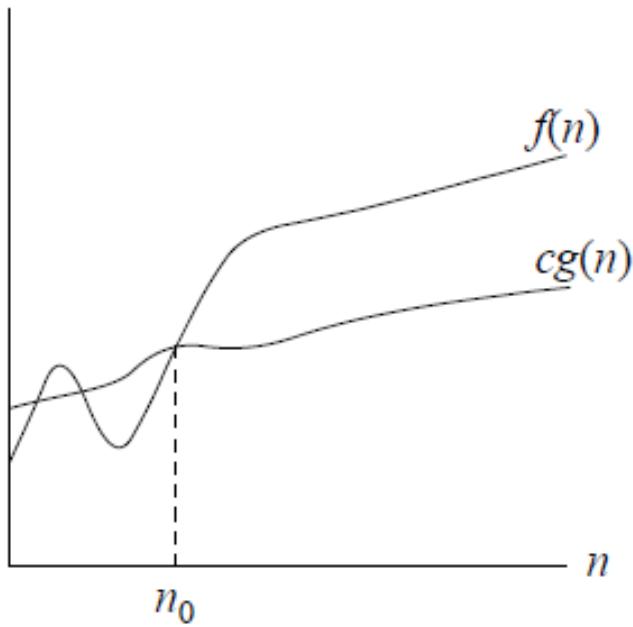


Big- Ω notation

Definition

We write $f(n) = \Omega(g(n))$ if there are **positive constants** n_0 and c such that for all $n \geq n_0$:

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



Examples

- ▶ $n^3 - 3n^2 + 5n = \Omega(n^3)$?
- ▶ Proof:



A useful result

Lemma [Lower Bound]

If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ exists, then $f(n) = \Omega(g(n))$ if and only if

$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \geq c$, where c is a **positive** constant.

Corollary [Lower Bound]

If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = +\infty$, then $f(n) = \Omega(g(n))$.

If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = \Omega(g(n))$ cannot hold.



More Examples

- ▶ $5n^2 + 6n + 8 = \Omega(n^3)$?
- ▶ $5n^2 + 6n + 8 = \Omega(n^2)$?
- ▶ $n^2 = \Omega(100 n^2)$?
- ▶ $\sqrt{6n^3 - 7n^2 + 3n} = \Omega(n^{1.5})$?
- ▶ $2^n = \Omega(n^2)$?
- ▶ $3\lg n = \Omega(n)$?
- ▶ $3^n = \Omega(2^n)$?
- ▶ $2^n = \Omega(3^n)$?
- ▶ $\log_{10} n = \Omega(\log_2 n)$?



Big- Θ notation



Big- Θ notation

Definition

We write $f(n) = \Theta(g(n))$ if there are **positive constants** n_0 , c_1 , and c_2 such that for all $n \geq n_0$:

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

- ▶ More precisely, should be $f(n) \in \Theta(g(n))$
- ▶ Intuitively, $f(n) = \Theta(g(n))$ means that
 - ▶ $f(n)$ grows like $g(n)$ (up to multiplicative constant factor)

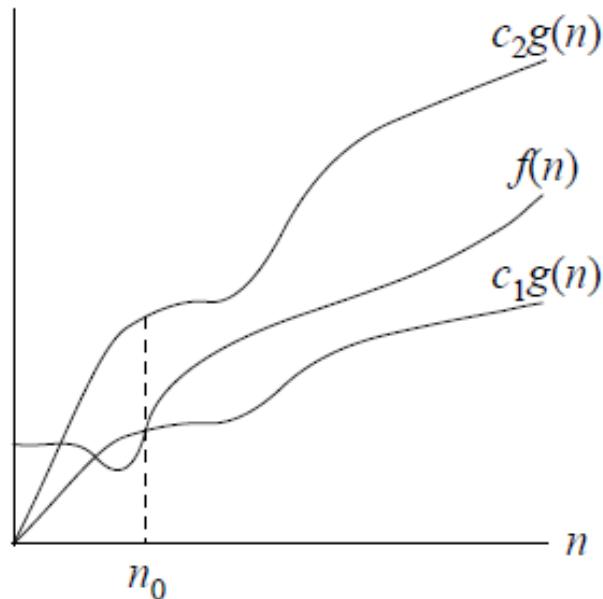


Big-Θ notation

Definition

We write $f(n) = \Theta(g(n))$ if there are **positive constants** n_0 , c_1 , and c_2 such that for all $n \geq n_0$:

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



A useful result

Lemma [Big-Theta]

If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ exists, then $f(n) = \Theta(g(n))$ if and only if

$c_1 \leq \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq c_2$, where c_1 and c_2 are **positive constants**.

Corollary [Big-Theta]

If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$ for some positive constant c , then $f(n) = \Theta(g(n))$.



Examples

► $2n^3 - 3n^2 = \Theta(n^3)$

► Proof:



More Examples

- ▶ $5n^2 + 6n + 8 = \Theta(n^2)$?
- ▶ $\sqrt{6n^3 - 7n^2 + 3n} = \Theta(n^{1.5})$?
- ▶ $n^2 - \lg n = \Theta(n^2)$?
- ▶ $3^n = \Theta(2^n)$?
- ▶ $\log_{10} n = \Theta(\log_2 n)$?

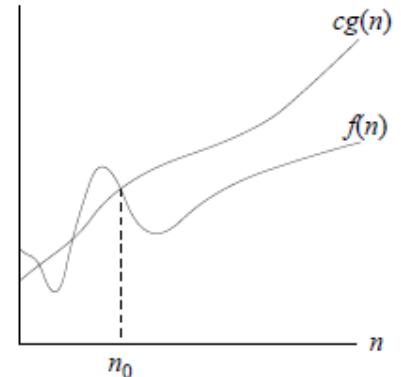


In summary

Big-O (upper bounded)

We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that for all $n \geq n_0$:

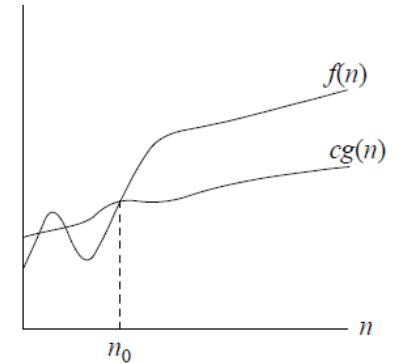
$$f(h) \leq c \cdot g(n)$$



Big-Ω (lower bounded)

We write $f(n) = \Omega(g(n))$ if there are **positive** constants n_0 and c such that for all $n \geq n_0$:

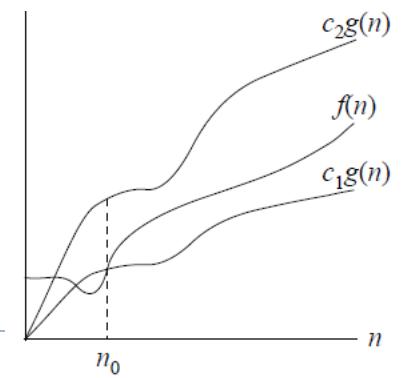
$$f(h) \geq c \cdot g(n)$$



Big-Θ (asymptotically the same)

We write $f(n) = \Theta(g(n))$ if there are **positive** constants n_0 , c_1 , and c_2 such that for all $n \geq n_0$:

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



More on nested loops analysis



A pseudo-code example

```
function func(n)
    1 x ← 0;
    2 i ← 7;
    3 while (i ≤ n) do
        4     | x ← x + i;
        5     | i ← i + 3;
    6 end
    7 return (x);
```

#iteration	value of i	Cost of this iteration



A second example

```
function func(n)

1 x ← 0;

2 i ← 7;

3 while ( $i \leq n$ ) do
4   | x ← x + i;
5   | i ← i * 3;

6 end

7 return (x);
```

#iteration	value of i	Cost of this iteration



A third example

```
function func(n)

1 x ← 0;

2 for i ← 1 to n do
3     j ← 1;
4     while (j ≤ n) do
5         x ← x + (i - j);
6         j ← 2 * j;
7     end
8 end

9 return (x);
```



A third example

```
function func(n)

1 x ← 0;

2 for i ← 1 to n do
3     j ← 1;
4     while (j ≤ i) do
5         x ← x + (i - j);
6         j ← 2 * j;
7     end
8 end

9 return (x);
```



FIN

