Lecture 04 — Asymptotic Notations

COMPSCI 308 — Design and Analysis of Algorithms

Xing Shi Cai https://newptcai.gitlab.io

Duke Kunshan University, Kunshan, China

Jan-Mar 2026

Summary

Asymptotic Analysis of Insertion Sort

Asymptotic Notations in Equations and Inequalities

Small o and Small ω Notations

Comparing Functions

Common Time Complexity

Assignments[1]

Practice makes perfect!

Introduction to Algorithms (ITA)

📖 Read —

- Section 3.1

✏️ Practice —

- [redacted]
- Exercises 3.2: 1–7.
- Problems 3-1, 3-2, 3-3, 3-4.

[1] 💣 Assignments will not be collected; however, quiz problems will be selected from them. (This includes both Readings and Exercises.)

Asymptotic Analysis of Insertion Sort

Insertion Sort

When A.length $= n$, what is $T(n)$, the total running time of the algorithm?

```
INSERTION-SORT(A, n)
1: for i = 2 to n do
2:     key = A[i]
3:     // Insert A[i] into the sorted subarray A[1 : i − 1].
4:     j = i − 1
5:     while j > 0 and A[j] > key do
6:         A[j + 1] = A[j]
7:         j = j − 1
8:     A[j + 1] = key
```

Let's introduce some notation —

- Let c_ℓ be the cost (running time) of line ℓ.
- Let t_i be the number of times line 5 is executed for each i.

INSERTION-SORT(A, n)

```
1: for i = 2 to n do
2:     key = A[i]
3:     // Insert A[i] into the sorted subarray A[1 : i − 1].
4:     j = i − 1
5:     while j > 0 and A[j] > key do
6:         A[j + 1] = A[j]
7:         j = j − 1
8:     A[j + 1] = key
```

cost	times
c_1	n
c_2	$n-1$
0	$n-1$
c_4	$n-1$
c_5	$\sum_{i=2}^{n} t_i$
c_6	$\sum_{i=2}^{n}(t_i-1)$
c_7	$\sum_{i=2}^{n}(t_i-1)$
c_8	$n-1$

Best-case Analysis

Given that

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1)$$

how small can it be?

- What is the smallest possible value for t_i? Answer: 1.

When $t_i = 1$ for all $i \in \{2, \dots n\}$.

we have

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8)n + (-c_2 - c_4 - c_5 - c_8)$$

$= \Theta(n)$

Worst-case Analysis

Given that

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1)$$

how large can it be?

💡 What is the largest possible value for t_i? Answer: $t_i \leq i$

If $t_i = i$ for all $i \in \{2, \dots n\}$.

then $\sum_{i=2}^{n} t_i = \sum_{i=2}^{n} i = \sum_{i=1}^{n} i - 1 = \frac{n(n+1)}{2} - 1$

$\sum_{i=2}^{n} (t_i - 1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$

Given that

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1)$$

how large can it be?

If $t_i = i$ (the largest possible value of t_i), we have

$$\begin{aligned} T(n) =& c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &+ c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ =& \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n \\ &- (c_2 + c_4 + c_5 + c_8). \end{aligned}$$

$\Theta(n^2)$

Let $T(n)$ be the worst case running time of insertion sort on Random Access Machine (RAM). In other words,

$$T(n) = \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8).$$

$= \Theta(n^2)$.

🍰 Which of the following are correct?

1. $T(n) = O(n^3)$ ✓
2. $T(n) = O(2n^2)$ ✓
3. $T(n) = O(n^2)$ ✓
4. $T(n) = \Theta(n^2)$ ✓

4. $T(n) = \Omega(n^2)$ ✓
5. $T(n) = \Omega(n)$ ✓
6. $T(n) = \Omega(2n)$ ✓
7. $T(n) = \Omega(\sqrt{n})$ ✓

Asymptotic Notations in Equations and Inequalities

Asymptotic Notations in Equations

Given equations like

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) \tag{1}$$

we interpret $\Theta(n)$ as an anonymous function $f(n)$ which satisfies

$$f(n) = \Theta(n)$$

How can we show that (1) is correct?

We have

$$2n^2 + 3n + 1 - 2n^2$$

$$= 3n + 1 = \Theta(n)$$

Asymptotic Notations in Equations

Given equations like

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) \tag{1}$$

we interpret $\Theta(n)$ as an anonymous function $f(n)$ which satisfies

$$f(n) = \Theta(n)$$

🍰 Is the following equation correct?

$$2n^2 + 3n + 1 = n^2 + \Omega(n)$$

we have:

$$2n^2 + 3n + 1 - n^2 = n^2 + 3n + 1 = \Omega(n)$$

correct!

The Number of Anonymous Functions

Each appearance of an asymptotic notation in an equation is interpreted as an anonymous function.

Thus,

$$n^3 + O(n^2) + O(n)$$

should be interpreted as

$$n^3 + f(n) + g(n)$$

where

$$f(n) = O(n^2), \qquad g(n) = O(n)$$

However, it is uncommon to see such equations since we can usually combine big-O notations.

Avoid Anonymous Functions in Sums

🧐 Thus, the sum

$$\sum_{i=1}^{n} O(i)$$

should *not* be interpreted as

$$O(1) + O(2) + \cdots + O(n)$$

but

$$\sum_{i=1}^{n} f(i)$$

where $f(i) = O(i)$ for all $i \in \{1, \ldots, n\}$ is an anonymous function.

💣 However, it does not really make sense to write expressions like

$$f(3) = O(3)$$ ← Not useful.

Merge Sort[2]

The worst-case running time of merge sort is

$$T(n) = 2T(n/2) + \Theta(n)$$

Later we will see that this equation is enough to find the asymptotic behaviour of $T(n)$. $= \Theta(n \cdot \log n)$

😆 So there is no need to figure out what the $\Theta(n)$ part is exactly.

[2] Merge sort will be covered in lecture 5.

We interpret the equation

$$2n^2 + \Theta(n) = \Theta(n^2) \tag{2}$$

as — For any $f(n) = \Theta(n)$, $2n^2 + f(n) = \Theta(n^2)$.

Is (2) correct?

If $f(n) = \Theta(n)$, then there exist c_1, c_2, n_0 (>0, ≥ 0) such that $c_1 n < f(n) < c_2 n$ for all $n \geq n_0$.

So $c_1 n + 2n^2 < f(n) + 2n^2 < c_2 n + 2n^2$. ($c_2 n^2 + 2n^2$)

2 $n^2 < c_1 n + 2n^2 < f(n) + 2n^2 < c_2 n + 2n^2 < (c_2 + 2) n^2$

Thus $f(n) + 2n^2 = \Theta(n^2)$

Asymptotic Notations on Both Sides of Equations

We interpret the equation

$$2n^2 + \Theta(n) = \Theta(n^2) \tag{2}$$

as — For any $f(n) = \Theta(n)$, $2n^2 + f(n) = \Theta(n^2)$.

🍰 **Is (2) correct? Answer: Yes.** Since $f(n) = \Theta(n)$, there exist $c_1, c_2, n_0 > 0$ such that

$$c_1 n \le f(n) \le c_2 n, \qquad \text{for all } n \ge n_0.$$

Adding $2n^2$ to the inequality leads to

$$2n^2 + c_1 n \le 2n^2 + f(n) \le 2n^2 + c_2 n.$$

In other words,

$$2n^2 \le 2n^2 + f(n) \le (2 + c_2)n^2, \qquad \text{for all } n \ge n_0.$$

Thus, $2n^2 + f(n) = \Theta(n^2)$ by definition.

Asymptotic Notations on Both Sides of Equations

We interpret the equation

$$2n^2 + \Theta(n) = \Theta(n^2) \tag{2}$$

as — For any $f(n) = \Theta(n)$, $2n^2 + f(n) = \Theta(n^2)$.

Is $n^3 + \Theta(n^2) = \Theta(n^2)$ correct? No !

We interpret the equation

$$2n^2 + \Theta(n) = \Theta(n^2) \tag{2}$$

as — For any $f(n) = \Theta(n)$, $2n^2 + f(n) = \Theta(n^2)$.

🍰 Is $n^3 + \Theta(n^2) = \Theta(n^2)$ correct?

Answer: No. By definition, any $g(n) = \Theta(n^2)$ must satisfy $g(n) \leq c \cdot n^2$ for some $c > 0$ and all $n \geq n_0$ where $n_0 \geq 0$ is a constant.

However, for any $f(n) = \Theta(n^2)$, we have:

$$\lim_{n\to\infty} \frac{n^3 + f(n)}{n^2} = \lim_{n\to\infty} \left(n + \frac{f(n)}{n^2}\right) = \infty.$$

There does not exist $c > 0$ s.t. $n^3 + f(n) < c \cdot n^2$ for all n.

Because this ratio grows without bound, no constant c can satisfy the upper bound requirement. Therefore, $n^3 + f(n) \neq \Theta(n^2)$.

Chaining Asymptotic Equations

We interpret

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2) \tag{3}$$

Correct

as two separate equations

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$$

Correct

and

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

Correct.

🍰 Does (3) hold?

Small o and Small ω Notations

Tight Asymptotic Upper Bound

The upper bound provided O may or may not be *tight*.

For example, $2n^2 = O(n^2)$ is tight in the sense that

$$\lim_{n\to\infty} \frac{2n^2}{n^2} = 2.$$

On the other hand, $n = O(n^2)$ is *not* tight in the sense that

$$\lim_{n\to\infty} \frac{n}{n^2} = 0.$$

n is much smaller compared to n^2

In the latter case, we can replace the O by o and write

$$n = o(n^2).$$

o-Notations

Formally, we write $f(n) = o(g(n))$ if for any $c > 0$, there exists n_0 such that

$$0 \le f(n) < cg(n), \qquad \text{for all } n \ge n_0.$$

Can you see why $f(n) = o(g(n))$ if and only if

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

Answer: By definition of lim, if.

Compare with $f(n) = O(g(n)) \Longleftrightarrow$ There exist $c_2 > 0$, and n_0, such that $f(n) < c_2 g(n)$ for all $n \ge n_0$.

ω-Notations

Formally, we write $f(n) = \omega(g(n))$ if for any $c > 0$, there exists n_0 such that

$$0 \leq cg(n) < f(n), \qquad \text{for all } n \geq n_0.$$

In other words, $f(n) = \omega(g(n))$ if and only if

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

By definition of the limit being ∞.

Comparing Functions

Transitivity in Asymptotic Notations

Transitivity means that if $f(n)$ is related to $g(n)$ and $g(n)$ is related to $h(n)$ by the same notation, then $f(n)$ is also related to $h(n)$ by that notation:

📌 Example — Transitivity in real life:

- You are friends with Alex.
- Alex is friends with Billy.
- Billy is friends with Cindy.
- Cindy is friends with David.
- David is friends with Taylor Swift.
- Therefore, you and Taylor Swift are also friends!

Transitivity in Asymptotic Notations

Transitivity means that if $f(n)$ is related to $g(n)$ and $g(n)$ is related to $h(n)$ by the same notation, then $f(n)$ is also related to $h(n)$ by that notation:

- $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \implies f(n) = \Theta(h(n))$
- $f(n) = O(g(n))$ and $g(n) = O(h(n)) \implies f(n) = O(h(n))$
- $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n)) \implies f(n) = \Omega(h(n))$
- $f(n) = o(g(n))$ and $g(n) = o(h(n)) \implies f(n) = o(h(n))$
- $f(n) = \omega(g(n))$ and $g(n) = \omega(h(n)) \implies f(n) = \omega(h(n))$

Reflexivity in Asymptotic Notations

Reflexivity means that any function $f(n)$ is related to itself by some asymptotic notations:

$$f(n) = \Theta(f(n)),$$
$$f(n) = O(f(n)),$$
$$f(n) = \Omega(f(n))$$

Reflexivity in Asymptotic Notations

Reflexivity means that any function $f(n)$ is related to itself by some asymptotic notations:

$$f(n) = \Theta(f(n)),$$
$$f(n) = O(f(n)),$$
$$f(n) = \Omega(f(n))$$

Do we have

$$f(n) = o(f(n)),$$ False
$$f(n) = \omega(f(n)).$$ False

No reflexivity

Symmetry in Asymptotic Notations

Symmetry means that if $f(n)$ is related to $g(n)$ by an asymptotic notation, then $g(n)$ should be related to $f(n)$ by the same notation.

For example —

$$f(n) = \Theta(g(n)) \implies g(n) = \Theta(f(n)).$$

If $f(n) = \Theta(g(n))$ then there are $c_1, c_2 > 0$, $n_0 \geq 0$.

such that for all $n \geq n_0$,

$$c_1\, g(n) < f(n) < c_2\, g(n).$$

Then $\frac{1}{c_2} f(n) < g(n) < \frac{1}{c_1} f(n)$

for $n \geq n_0$ Thus $g(n) = \Theta(f(n))$.

Symmetry in Asymptotic Notations

Symmetry means that if $f(n)$ is related to $g(n)$ by an asymptotic notation, then $g(n)$ should be related to $f(n)$ by the same notation.

For example —

$$f(n) = \Theta(g(n)) \implies g(n) = \Theta(f(n)).$$

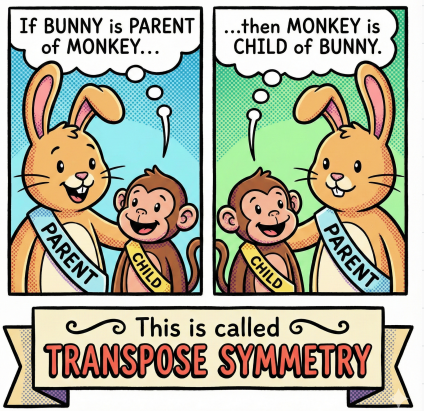
🍰 Do we have

$$f(n) = O(g(n)) \implies g(n) = \Omega(f(n)),$$
$$f(n) = \Omega(g(n)) \implies g(n) = O(f(n))$$

Transpose Symmetry in Asymptotic Notations

Transpose Symmetry means that if $f(n)$ is related to $g(n)$ by one asymptotic notation, then $g(n)$ should be related to $f(n)$ by another notation:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \_\_\_\_\_(f(n)),$$
$$f(n) = o(g(n)) \Leftrightarrow g(n) = \_\_\_\_\_(f(n)).$$



Trichotomy — The division into three mutually exclusive or contrasting groups, categories, or parts.

📌 For example, real numbers have a trichotomy — for any $a, b \in \mathbb{R}$, exactly one of the following must hold:

$$a < b, \qquad a = b, \qquad a > b.$$

Figure 1: A 🐰 facing a trichotomy

Trichotomy in Asymptotic Notations

We say that

- $f(n)$ is asymptotically smaller than $g(n)$ if $f(n) = o(g(n))$, denoted by $f(n) \prec g(n)$;
- $f(n)$ is asymptotically larger than $g(n)$ if $f(n) = \omega(g(n))$, denoted by $f(n) \succ g(n)$.

💣 Not all functions are asymptotically comparable, e.g.:

$$f(n) = n, \qquad g(n) = n^{1+\sin n}.$$

No Trichotomy in Asymptotic Notations

We say that

- $f(n)$ is asymptotically smaller than $g(n)$ if $f(n) = o(g(n))$, denoted by $f(n) \prec g(n)$;
- $f(n)$ is asymptotically larger than $g(n)$ if $f(n) = \omega(g(n))$, denoted by $f(n) \succ g(n)$.

💣 Not all functions are asymptotically comparable, e.g.:

$$f(n) = n, \qquad g(n) = n^{1+\sin n}.$$

🧩 Are monotonically increasing functions always asymptotically comparable?

Common Time Complexity

Common Time Complexity

Let $T(n)$ be the average/worst-case running time of an algorithm given an input of size n.

The time complexity of this algorithm is the rate of growth of $T(n)$.

Time complexity is a way to express how the running time of an algorithm grows as the size of the input to the algorithm increases.

Some common time complexities are:

- constant: $O(1)$
- log-logarithmic: $O(\log \log n)$
- logarithmic: $O(\log n)$ — e.g. $(\log n)^2 + 3 \cdot \log n$.
- polylogarithmic: $O(\text{poly}(\log n))$
- sub-linear: $o(n)$

Common Time Complexity

Let $T(n)$ be the average/worst-case running time of an algorithm given an input of size n.

The time complexity of this algorithm is the rate of growth of $T(n)$.

Time complexity is a way to express how the running time of an algorithm grows as the size of the input to the algorithm increases.

Some common time complexities are:

- constant: $O(1)$
- log-logarithmic: $O(\log \log n)$
- logarithmic: $O(\log n)$
- polylogarithmic: $O(\text{poly}(\log n))$
- sub-linear: $o(n)$
- linear: $O(n)$
- linearithmic: $O(n \log n)$
- quadratic: $O(n^2)$
- cubic: $O(n^3)$
- exponential: $2^{O(n)}$

Constant Time Complexity

A constant time algorithm has running time $O(1)$.

In other words, its running time is bounded by a constant regardless of the input size.

📌 Examples include:

- Conditional branch (e.g., `if (x > 0) {...}`)
- Arithmetic/logic operation (e.g., `y = a + b`)
- Declare/initialize a variable (e.g., `int x = 0`)
- Follow a link in a linked list (e.g., `node = node->next`)
- Access element i in an array (e.g., `A[i]`)
- Compare/exchange two elements in an array (e.g., swap `A[i], A[j]`)

Linear Time Complexity

A linear time algorithm has running time $O(n)$.

In other words, its running time is bounded by a constant factor times the input size.

A common reason for this is the algorithm processes the input in a single pass.

📌 Examples include finding the maximum element in an array.

🍰 Can you think of another example?

Linear Time Complexity

A linear time algorithm has running time $O(n)$.

In other words, its running time is bounded by a constant factor times the input size.

A common reason for this is the algorithm processes the input in a single pass.

📌 Examples include finding the maximum element in an array.

🍰 Can you think of another example?

🧩 Finding the Median

The median of a set of numbers is the value separating the higher half from the lower half.
For example, the median of

$$0, 2, 3, 4, 11$$

is 3.
Can we find the median of an unsorted array in linear time?

A logarithmic time algorithm has running time $O(\log n)$.

📌 Examples include searching for an element in a sorted array.

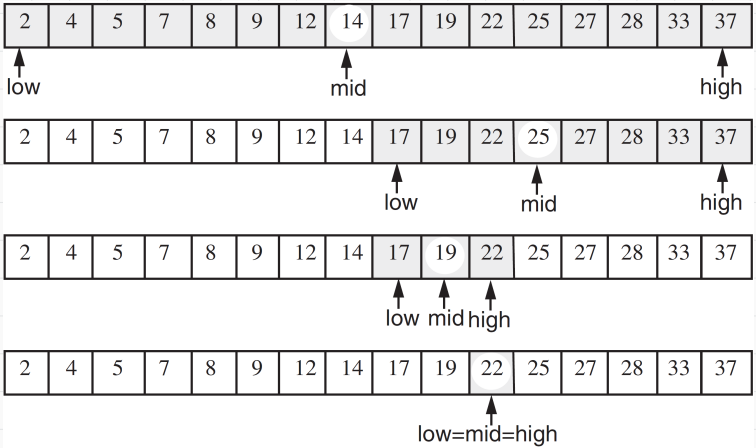


Figure 2: Binary search for **22** in A

Logarithmic Time Complexity

A logarithmic time algorithm has running time $O(\log n)$.

📌 Examples include searching for an element in a sorted array.

```
1:  Binary-Search(A[1..n], key)
2:  low ← 1; high ← n
3:  while low ≤ high do
4:      mid ← (low + high)//2
5:      if A[mid] = key then return mid
6:      else if A[mid] < key then
7:          low ← mid + 1
8:      else
9:          high ← mid − 1
10:     end if
11: return NULL
```

Given the array of length 15,

$$A = \langle 12, 25, 32, 37, 41, 48, 58, 60, 66, 73, 74, 79, 83, 91, 95 \rangle$$

How many comparisons does it take to find the position of 74 with binary search?

💡 You should count = and < as two comparisons.

Linear Time Complexity

A linear time algorithm has running time $O(n)$.

📌 Examples include searching for an element in an unsorted array.

```
1: Linear-Search(A[1..n], key)
2: for i ← 1 to n do
3:     if A[i] = key then
4:         return i
5:     end if
6: return NULL
```

Quadratic Time Complexity

A quadratic time algorithm has running time $O(n^2)$.

📌 A simple example is the brute-force algorithm for finding the closest pair of points in the plane.

```
1:  Closest-Pair(x[1..n], y[1..n])
2:  d ← ∞
3:  cp ← (NULL, NULL)
4:  for i ← 1 to n − 1 do
5:      for j ← i + 1 to n do
6:          dt ← √((x[i] − x[j])² + (y[i] − y[j])²)
7:          if dt < d then
8:              d ← dt
9:              cp ← ((x[i], y[i]), (x[j], y[j]))
10:         end if
11: return cp
```

Exponential Time Complexity

An exponential time algorithm has running time $O(c^n)$ for some constant $c > 1$.

📌 A classic example is the naïve recursive computation of Fibonacci numbers

$$\{F_n\}_{n=0}^{\infty} = \{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots\}$$

which is defined recursively as

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

```
1: Fib(n)
2: if n ≤ 1 then
3:     return n
4: else
5:     return Fib(n − 1) + Fib(n − 2)
6: end if
```

What are your main takeaways today? Any questions?