

CS 375

September 14, 2026

Lecture focus: Vocabulary and Notations

Big O Notation

We say $f(n)$ is "Big-O" of $g(n)$ if there is some constant c such that $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq c$

example:

1. $f(n) = n$
 $g(n) = n^2$

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

2. $f(n) = n^3$
 $g(n) = n^2$ $\lim_{n \rightarrow \infty} \frac{n^3}{n^2} = +\infty \Rightarrow n^3 \neq O(n^2)$

3. $3n^2 + 5n + 1 \Rightarrow \text{Big-O of each other.}$
 $2n^2 + 10n + 10^6$

Ω - Big Omega

$$f(n) = \Omega(g(n)) \text{ if } g(n) = O(f(n))$$

where " O " is an upper-bound that is $\leq$

Ω is a "lower bound" that is $\geq$

② ~ "Big Theta"

$$f(n) = \Theta(g(n)) \text{ if } f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n))$$

Θ ~ an asymptotic " $\equiv$ " sign

Interesting Questions :

- * What is the worst-case runtime of a given algorithm?
- * What is the average-case runtime of a given algorithm?
- * Is this algorithm correct?
- * What is the best-case runtime of a given algorithm?
- * Is there an algorithm that correctly solves a given problem?
- * What algorithm guarantees the lowest worst-case runtime for a given problem?
- * How hard is this problem?

→ ; All of the above for space efficiency

What does it mean for an algorithm to be correct?

It always gives the right output for every input.

Proof By Induction

$$1 + 2 + 3 + \dots + (n-1) + n = \frac{n(n+1)}{2}$$

Base case: ($n=1$)

$$1 = \frac{1(1+1)}{2}$$

Inductive step:

Assume I know the formula holds for some particular value n .

Now, show it works for $n+1$

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

$$= \frac{n(n+1)}{2} + n+1$$

$$= \frac{n^2 + n + 2n + 2}{2}$$

$$= \frac{n^2 + 3n + 2}{2}$$

$$= \frac{(n+1)(n+2)}{2}$$

if $n+1$ is substituted for n in this formula, we get

$$\frac{(n+1)(n+2)}{2}$$

Example 2.

$$1 + 2 + 4 + 8 + \dots + 2^{n-1} = 2^n - 1$$

Base Case: ($n=1$)

$$2^0 = 1 = 2^1 - 1$$

Inductive Step (show that if it works for n , it works for $n+1$)

$$1 + 2 + \dots + 2^{n-1} + 2^n + 2^n$$

$$\Rightarrow 2^n - 1 + 2^n$$

$$= 2(2^n) - 1$$

$$= 2^{n+1} - 1$$

Insertion Sort (Input $A[1:n+1]$)
 $\nwarrow$ last index is n

for $i = 1, \dots, n$

$j = i$

 while $j > 1$ and $A[j] < A[j-1]$

 swap $A[j]$ and $A[j-1]$

$j \dots$

After the first i iterations, the first i items are sorted

Base Case:

On iteration $i=1$, nothing happens in the while loop, but first "one" element is sorted.

Inductive step:

If it worked for the i^{th} iteration, show it works on the $(i+1)^{\text{th}}$ iteration.