When your interviewer asks for the time complexity of your algorithm but you have no idea what that means.

### Big-O Notation

**O-notation:**

\[ f(n) \in O(g(n)) \] if there exist constants \( c > 0 \) and \( n_0 > 0 \) such that \( 0 \leq f(n) \leq cg(n) \) for all \( n \geq n_0 \).

Here the complexity of \( f \) is not higher than the complexity of \( g \).

**Ω-notation:**

\[ f(n) \in \Omega(g(n)) \] if there exist constants \( c > 0 \) and \( n_0 > 0 \) such that \( 0 \leq cg(n) \leq f(n) \) for all \( n \geq n_0 \).

Here the complexity of \( f \) is not lower than the complexity of \( g \).

**Θ-notation:**

\[ f(n) \in \Theta(g(n)) \] if there exist constants \( c_1, c_2 > 0 \) and \( n_0 > 0 \) such that \( 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \) for all \( n \geq n_0 \).

Here \( f \) and \( g \) have the same complexity.

**ω-notation:**

\[ f(n) \in \omega(g(n)) \] if for all constants \( c > 0 \), there exists a constant \( n_0 > 0 \) such that \( 0 \leq cg(n) \leq f(n) \) for all \( n \geq n_0 \).

Here \( f \) has lower complexity than \( g \).

**ω-notation:**

\[ f(n) \in \omega(g(n)) \] if for all constants \( c > 0 \), there exists a constant \( n_0 > 0 \) such that \( 0 \leq cg(n) \leq f(n) \) for all \( n \geq n_0 \).

Here \( f \) has higher complexity than \( g \).

Note: \( O + \Omega = \Theta \).
EXERCISE

• Which of the following are true?
  • $n^2 \in O(n^3)$
  • $n^2 \in o(n^3)$
  • $n^3 \in \omega(n^2)$
  • $\log n \in o(n)$
  • $n \log n \in \Omega(n)$
  • $n \log^2 n \in \omega(n \log n)$
  • $n \in \Theta(n \log n)$

EXERCISE

• Which of the following are true?
  • $n^2 \in O(n^3)$  YES
  • $n^2 \in o(n^3)$  YES
  • $n^3 \in \omega(n^2)$  NO
  • $\log n \in o(n)$  YES
  • $n \log n \in \Omega(n)$  YES
  • $n \log^2 n \in \omega(n \log n)$  NO
  • $n \in \Theta(n \log n)$  NO

Intuitively, we have the following correspondences between order notation and growth rates:

- $f(n) \in O(g(n))$ means the growth rate of $f$ is $\leq$ the growth rate of $g$.
- $f(n) \in o(g(n))$ means the growth rate of $f$ is $< \text{the growth rate of } g$.
- $f(n) \in \Omega(g(n))$ means the growth rate of $f$ is $\geq \text{the growth rate of } g$.
- $f(n) \in \omega(g(n))$ means the growth rate of $f$ is $> \text{the growth rate of } g$.
- $f(n) \in \Theta(g(n))$ means the growth rate of $f$ is $= \text{the growth rate of } g$.

4n \in O(n^2)  7n^2 \in \Omega(n)
4n \in o(n^2)  7n^2 \in \omega(n)
4n \in O(n^2)  4n \in \Omega(n)
7n^2 \in o(n^2)  4n \in \omega(n)

This is included for your notes.

Relationships between Order Notations

- $f(n) \in O(g(n)) \Rightarrow g(n) \in \Theta(f(n))$
- $f(n) \in o(g(n)) \Rightarrow g(n) \in \Omega(f(n))$
- $f(n) \in \omega(g(n)) \Rightarrow g(n) \in \omega(f(n))$
- $f(n) \in \Theta(g(n)) \Rightarrow f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$
- $f(n) \in o(g(n)) \Rightarrow f(n) \in O(g(n))$
- $f(n) \in \omega(g(n)) \Rightarrow f(n) \in \Omega(g(n))$

WORKED EXERCISES

1. Let $f(n) = n^2 - 7n - 30$. Prove from first principles that $f(n) \in O(n^2)$.
2. Let $f(n) = n^2 - 7n - 30$. Prove from first principles that $f(n) \in \Omega(n^2)$.
3. Suppose $f(n) = n^2 + n$. Prove from first principles that $f(n) \notin O(n)$.
EXAMPLE 1: \( f(n) = n^2 - 7n - 30 \)

- Want to prove (WTP) from first principles: \( f(n) \in O(n^2) \)
  - More formally: there exist constants \( c > 0, n_0 > 0 \)
    such that for all \( n \geq n_0 \), we have \( 0 \leq f(n) \leq cn^2 \)
- Pick a value for \( c \). How about 1?
- Let’s visualize \( c = 1 \)

Seems plausible that \( c = 1 \) will work.

Let’s prove this algebraically.

Solution:

When is \( f(n) \leq cn^2 \)

\[ n^2 - 7n - 30 \leq n^2 \] (for all \( n \geq 0 \))

When is \( 0 \leq f(n) \)

\[ f(n) = n^2 - 7n - 30 = (n - 10)(n + 3) \]

When is \( (n - 10)(n + 3) \geq 0 \) when \( n \geq 10 \).

So, the claim holds with \( c = 1, n_0 = 10 \).

EXAMPLE 2: \( f(n) = n^2 - 7n - 30 \)

- WTP from first principles: \( f(n) \in \Omega(n^2) \)
  - More formally: there exist constants \( c > 0, n_0 > 0 \)
    such that for all \( n \geq n_0 \), we have \( 0 \leq f(n) \leq cn^2 \)
- Pick a value for \( c \).
- How about 1?
- Must show \( n^2 \leq n^2 - 7n - 30 \)
- Impossible! \( c = 1 \) is too large.
- Let’s try \( c = \frac{1}{2} \).

Solution:

When is \( f(n) \leq \frac{1}{2}n^2 \)

\[ n^2 - 7n - 30 \leq \frac{1}{2}n^2 \] (for all \( n \geq 0 \))

First part \( 0 \leq \frac{1}{2}n^2 \) is easy: satisfied for all \( n \geq 0 \) (i.e., for any \( n_0 \)).

Second part \( \frac{1}{2}n^2 \leq n^2 - 7n - 30 \) holds when \( \frac{1}{2}n^2 - 7n - 30 \geq 0 \)

Roots are \( 7 \pm \sqrt{109} \), which are < 18

Result: \( c = \frac{1}{2}, n_0 = 18 \) works!

EXAMPLE 3: \( f(n) = n^2 + n \)

- WTP from first principles \( f(n) \notin O(n) \).
  - Formally:
    \[ \neg(\exists c > 0, \forall n_0 > 0, \forall n \geq n_0 : f(n) \leq cn) \]
    \[ \forall c > 0, \forall n_0 > 0, \exists n \geq n_0 : f(n) < 0 \text{ or } f(n) > cn \]
- Consider any arbitrary \( c > 0, n_0 > 0 \)
- We find some \( n \geq n_0 \) such that \( n^2 + n < 0 \) or \( n^2 + n > cn \)
  - \( n^2 + n > cn \) iff \( n^2 + n - cn > 0 \) iff \( n(n + 1 - c) > 0 \)
  - For \( n \geq n_0 > 0 \), this holds iff \( n + 1 - c > 0 \), equivalently \( n > c - 1 \)
  - So, \( n = \max(c, n_0) \) will suffice.

You vs. the guy she tells you not to worry about

\[ \begin{array}{c|c}
  \text{O}(n^2) & \text{O}(n \log n) \\
\end{array} \]

Comparing Growth Rates
All of the identities shown hold only if the limits exist.
USING THE LIMIT METHOD: EXERCISE 1

• Compare growth rate of \( n^2 \) and \( n^2 - 7n - 30 \)

\[
\lim_{n \to \infty} \frac{n^2 - 7n - 30}{n^2} = \lim_{n \to \infty} \left(1 - \frac{7}{n} - \frac{30}{n^2}\right) = 1
\]

\( \therefore \) \( n^2 - 7n - 30 \in \Theta(n^2) \)

When you derive \( e^x \)

Try these at home...

This is included for your notes

USING THE LIMIT METHOD: EXERCISE 2

• Compare growth rate of \((\ln n)^2\) and \(n^{1/2}\)

\[
\lim_{n \to \infty} \frac{\ln n}{n^{1/2}} = \lim_{n \to \infty} \frac{\frac{1}{n}}{2n^{-1/2}} = \lim_{n \to \infty} \frac{\ln n}{n^{1/2}} = \lim_{n \to \infty} \frac{1}{n^{1/2}} = 0
\]

\( \therefore (\ln n)^2 \in o(n^{1/2}) \)

Additional Exercises

1. Compare the growth rate of the functions \((3 + (-1)^n)\) \(n\) and \(n\).

2. Compare the growth rates of the functions \(f(n) = n [\sin n/2] + 1\) and \(g(n) = \sqrt{n}\).

Algebra of Order Notations

"Maximum" rules: Suppose that \( f(n) > 0 \) and \( g(n) > 0 \) for all \( n \geq n_0 \). Then

\[
O(f(n) + g(n)) = O(\max\{f(n), g(n)\})
\]

\[
\Omega(f(n) + g(n)) = \Omega(\max\{f(n), g(n)\})
\]

\[
\Theta(f(n) + g(n)) = \Theta(\max\{f(n), g(n)\})
\]

"Summation" rules: Suppose \( f \) is a finite set. Then

\[
O\left(\sum_{i=1}^{n} f(i)\right) = \sum_{i=1}^{n} O(f(i))
\]

\[
\Omega\left(\sum_{i=1}^{n} f(i)\right) = \sum_{i=1}^{n} \Omega(f(i))
\]

\[
\Theta\left(\sum_{i=1}^{n} f(i)\right) = \sum_{i=1}^{n} \Theta(f(i))
\]

SUMMATIONS

AND SEQUENCES

\[\sum\]

\[\int\]
Summation rules are commonly used in loop analysis.

Example:
\[
\sum_{i=1}^{n} O(i) = O\left(\sum_{i=1}^{n} i\right) = O\left(\frac{n(n+1)}{2}\right) = O(n^2).
\]

**SEQUENCES**

Arithmetic sequence:
\[
\sum_{i=0}^{n-1} (a + dt) = na + d\frac{n(n-1)}{2} \in \Theta(n^2).
\]

Geometric sequence:
\[
\sum_{i=0}^{n-1} ar^i = \begin{cases} 
\frac{ar^{n+1}-1}{(r-1)} & \text{if } r > 1 \\
na & \text{if } r = 1 \\
\frac{a(1-r^n)}{1-r} & \text{if } 0 < r < 1.
\end{cases}
\]

**SEQUENCES CONTINUED**

Arithmetic-geometric sequence:
\[
\sum_{i=0}^{n-1} (a_i + dr^i) = \frac{a}{1-r} - \frac{(a+(a-1)d)(1-r)^n}{1-r} + dr(1-r^{n-1}) (1-r)^2.
\]

provided that \(r \neq 1\).

Harmonic sequence:
\[
H_n = \sum_{i=1}^{n} \frac{1}{i} \in \Theta(\log n)
\]

**Miscellaneous Formulae**

\(a^n \in \Theta(n^{(n+1)/e^n})\)

\(\log_2 n \in \Theta(n \log n)\)

Another useful formula is
\[
\sum_{i=1}^{n} \frac{1}{i^2} = \frac{\pi^4}{6},
\]

which implies that
\[
\sum_{i=1}^{n} \frac{1}{i} \in \Theta(1).
\]

A sum of powers of integers when \(e \geq 1\):
\[
\sum_{i=1}^{n} r^i \in \Theta(n^{r+1}).
\]

**Logarithm Formulae**

1. \(\log_b xy = \log_b x + \log_b y\)
2. \(\log_b \frac{x}{y} = \log_b x - \log_b y\)
3. \(\log_b 1/x = -\log_b x\)
4. \(\log_b x^y = y \log_b x\)
5. \(\log_b a = \frac{1}{\log_a b}\)
6. \(\log_b a = \frac{\log_c a}{\log_c b}\)
7. \(a^{\log_b c} = c^{\log_b a}\)
BASE OF LOGARITHM DOES NOT MATTER!

• Big-O notation does not distinguish between log bases
• Proof:
  • Fix two constant logarithm bases \( b \) and \( c \)
  • From log rules, we can change from \( \log_c x \) to \( \log_b x \)
    by using formula: \( \log_b x = \frac{\log_c x}{\log_c b} \)
  • But \( \log_c b \) is a constant!
  • So \( \log_b x \in \Theta(\log_c x) \)

We typically omit the base, and just write \( \Theta(\log x) \) for this reason

RUNNING TIME ANALYSIS

MODEL OF COMPUTATION

• Before we can analyze the running time of code, we need a precise model of computation
  • We use the Word-RAM model
    • Each memory location is a word that can hold an integer
    • Accessing a word of memory takes constant time
    • Basic operations (arithmetic, shifting, logical operators) take constant time
• Is a word large enough to hold any integer? Unlikely… split large integers into arrays of words. Watch cost of operations!

But, large enough to hold an address of an object in a data structure? Yes! If the data structure fits in RAM…

META-ALGORITHM FOR ANALYZING LOOPS

• Identify operations that require only constant time
• The complexity of a loop is the sum of the complexities of all iterations
• Analyze independent loops separately and add the results
• If loops are nested, it often helps to start at the innermost, and proceed outward… but,
  • sometimes you must express several nested loops together in a single equation (using nested summations),
  • and actually evaluate the nested summations… (can be hard)

LOOP ANALYSIS

Running Time of a Program: \( T_M(I) \) denotes the running time (in seconds) of a program \( M \) on a problem instance \( I \).

Worst-case Running Time as a Function of Input Size: \( T_M(n) \) denotes the maximum running time of program \( M \) on instances of size \( n \):

\[
T_M(n) = \max\{ T_M(I) : \text{Size}(I) = n \}.
\]

Average-case Running Time as a Function of Input Size: \( \bar{T}_M^a(n) \) denotes the average running time of program \( M \) over all instances of size \( n \):

\[
\bar{T}_M^a(n) = \frac{1}{|\{ I : \text{Size}(I) = n \}|} \sum_{|\{ I : \text{Size}(I) = n \}|} T_M(I).
\]
TWO BIG-O ANALYSIS STRATEGIES

- **Strategy 1**
  - Prove a $O$-bound and a matching $\Omega$-bound separately to get a $\Theta$-bound.
  - Often easier (but not always)

- **Strategy 2**
  - Use $\Theta$-bounds throughout the analysis and thereby obtain a $\Theta$-bound for the complexity of the algorithm:

```
Algorithm: LoopAnalysis\(n : integer\)
1. \(\text{sum} \leftarrow 0\)
2. for \(i \leftarrow 1 \text{ to } n\)
   do \(\text{for } j \leftarrow 1 \text{ to } i\)
      do \(\text{sum} \leftarrow \text{sum} + (i - j)^2\)
         \(\text{sum} \leftarrow \text{sum}/i\)
   \(\text{return} (\text{sum})\)
```

EXAMPLE 1

```
Strategy 1: big-O and big-\(\Omega\) bounds

We focus on the two nested for loops \((i.e., (2))\).
The total number of iterations is \(\sum_{i=1}^{n} j\), with $\Theta(1)$ time per iteration.
Upper bound:
\[
\sum_{i=1}^{n} O(i) = \sum_{i=1}^{n} O(n) = O(n^2).
\]
Lower bound:
\[
\sum_{i=1}^{n} \Omega(i) \geq \sum_{i=1}^{n} \Omega(n/2) = \Omega(n^2/4) = \Omega(n^2).
\]

Since the upper and lower bounds match, the complexity is $\Theta(n^2)$.
```

```
EXAMPLE 2

```
sum := 0;
for i := 1 to n do
  j := 1;
  while j <= i do
    sum := sum + j;
    j := floor(j/2);
print(sum)
```

```
Consider this loop alone. number of loop iterations?

\[\sum_{i=1}^{n} \log(j) \leq \sum_{i=1}^{n} \log(i) = O(n \log n)\]
```

```
... ANOTHER EXERCISE IN LOOP ANALYSIS?

Olive Garden waiter: Sir, you've already had 5 baskets of breadsticks. Me:

We're going to have 20.
Example 3 (Bentley's Problem, Solution 1)

```java
max := 0;
for i := 1 to n do
    for j := i to n do
        sum := 0;
        for k := i to j do
            sum := sum + A[k];
        if sum > max then max := sum;
```

Try to analyze this yourself!
One possible solution is given in these slides...

Strategy 1: big-O and big-Ω bounds

```
T(n) ∈ Θ(1) + \sum_{i=2}^{n} \sum_{j=1}^{i-1} (j-i) = Θ(n^2)
```

```
T(n) ∈ Θ(1) + \sum_{i=2}^{n} \sum_{j=1}^{i-1} (j-i) ≤ O\left(\sum_{i=2}^{n} \sum_{j=1}^{i-1} n^2\right)
```

```
This is the maximum number of iterations that could be performed in this loop
```

Proving a big-Ω bound...

Recall:
```
T(n) ∈ Θ(1) + \sum_{i=2}^{n} \sum_{j=1}^{i-1} (j-i) = Ω(n^2)
```

Intuition: `j - i` is \(Ω(n)\) in some iterations. How many iterations? Lots!

To get a good Ω-bound, we ask questions like:
When do our loops have many iterations?
When is our dominant term large?

Many iterations: when our `j` loop does \(Ω(n)\) iterations! For example, when \(i ≤ n/2\).

Large dominant term: when `j` is much larger than `i` (i.e., by a factor of `n`).

Recall:
```
\sum_{j=1}^{i} 2(i-j) = \sum_{j=1}^{i} 2i - 2j = 2i^2 - 2i
```

```
for i := 1 to n do
    for j := 1 to n do
        sum := 0;
        for k := i to j do
            sum := sum + A[k];
        if sum > max then max := sum;
```

Smallest possible value of `j - i` for these bounds on `i, j`:

```
Smallest possible value of `j - i` for these bounds on `i, j`:
```

We will perform at least this much work in every iteration!

This term does not depend on the loop indexes, so just multiply by the total number of loop iterations...

Since we have \(Θ(n^2)\) and \(Ω(n^2)\), we have proved \(Θ(n^2)\).

Proving a big-Ω bound... continued