CS 341: ALGORITHMS

Lecture 2: background and analysis
Readings: CLRS Chapters 2.1, 2.2, 3

Trevor Brown (co-taught with Anna Lubiw)

https://www.student.cs.uwaterloo.ca/~cs341

trevor.brown@uwaterloo.ca
When your interviewer asks for the time complexity of your algorithm but you have no idea what that means

DaCobalt • 1d
Big Oof notation

True story

BIG-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 \( g \).
\( \Theta \)-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.
\( o \)-notation:

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

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

\( \omega \)-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 \( g \).
EXERCISE

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

• Which of the following are true?
  • $n^2 \in O(n^3)$  YES
  • $n^2 \in o(n^3)$  YES
  • $n^3 \in \omega(n^3)$  NO
  • $\log n \in o(n)$  YES
  • $n \log n \in \Omega(n)$  YES
  • $n \log n \in \omega(n \log n)$  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 $<$ the growth rate of $g$
- $f(n) \in \Omega(g(n))$ means the growth rate of $f$ is $\geq$ the growth rate of $g$
- $f(n) \in \omega(g(n))$ means the growth rate of $f$ is $>$ the growth rate of $g$
- $f(n) \in \Theta(g(n))$ means the growth rate of $f$ is $=$ the growth rate of $g$

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

This is included for your notes
Relationships between Order Notations

\[ f(n) \in \Theta(g(n)) \iff g(n) \in \Theta(f(n)) \]
\[ f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n)) \]
\[ f(n) \in o(g(n)) \iff g(n) \in \omega(f(n)) \]

\[ f(n) \in \Theta(g(n)) \iff f(n) \in O(g(n)) \text{ and } f(n) \in \Omega(g(n)) \]
\[ f(n) \in o(g(n)) \Rightarrow f(n) \in O(g(n)) \]
\[ f(n) \in \omega(g(n)) \Rightarrow f(n) \in \Omega(g(n)) \]
Prove that \( f(n) \in \Theta(g(n)) \) implies \( g(n) \in \Theta(f(n)) \).

**Proof:** Suppose \( f(n) \in \Theta(g(n)) \). Then there exist constants \( c_1, c_2, n_0 \) such that

\[
0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)
\]

if \( n \geq n_0 \). Thus

\[
0 \leq \frac{1}{c_2} f(n) \leq g(n) \leq \frac{1}{c_1} f(n)
\]

if \( n \geq n_0 \). Define \( c'_1 = 1/c_2 \), \( c'_2 = 1/c_1 \) and \( n'_0 = n_0 \). Then

\[
0 \leq c'_1 f(n) \leq g(n) \leq c'_2 f(n)
\]

if \( n \geq n'_0 \).
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 \( f(n) \not\in O(n) \).
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
EXAMPLE 1: $f(n) = n^2 - 7n - 30$

• WTP: there exist constants $c > 0, n_0 > 0$ such that for all $n \geq n_0$, we have $0 \leq f(n) \leq cn^2$

• 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 $n \geq 10$.
    • (or when $n \leq -3$ ... but we want $n_0 > 0$)

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 \( 0 \leq cn^2 \leq f(n) \)

- Solution:
  - 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 \( c = \frac{1}{2} \).
EXAMPLE 2: \( f(n) = n^2 - 7n - 30 \)

- WTP: there exist constants \( c > 0, n_0 > 0 \) such that for all \( n \geq n_0 \), we have \( 0 \leq cn^2 \leq f(n) \)

- Solution:
  
  - Let’s try \( c = \frac{1}{2} \).
  
  - Goal: show \( 0 \leq \frac{1}{2} n^2 \leq n^2 - 7n - 30 \)
  
  - 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 \( \frac{1}{2} n^2 - 7n - 30 \leq 0 \)
  
  - 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:
  
  $\neg(f(n) \in O(n))$

  $\neg(\exists c > 0, n_0 > 0 \quad \forall n \geq n_0 \quad : \quad 0 \leq f(n) \leq cn)$

  $\forall c > 0, n_0 > 0 \quad \exists n \geq n_0 \quad : \quad f(n) < 0 \text{ or } f(n) > cn$

- 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
| $O(n^2)$ | $O(n \log n)$ |

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

**COMPARING GROWTH RATES**
Some Common Growth Rates (in increasing order)

polynomial
- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(\sqrt{n})$
- $\Theta(n)$
- $\Theta(n^2)$
- $\Theta(n^c)$

exponential
- $\Theta(1.1^n)$
- $\Theta(2^n)$
- $\Theta(e^n)$
- $\Theta(n!)$
- $\Theta(n^n)$
LIMIT TECHNIQUE
FOR COMPARING GROWTH RATES

Suppose that $f(n) > 0$ and $g(n) > 0$ for all $n \geq n_0$. Suppose that

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

Then

$$f(n) \in \begin{cases} 
    o(g(n)) & \text{if } L = 0 \\
    \Theta(g(n)) & \text{if } 0 < L < \infty \\
    \omega(g(n)) & \text{if } L = \infty.
\end{cases}$$
**Constant Function Rule**

The limit of a constant function is the constant:

$$\lim_{x \to a} C = C.$$ 

**Sum Rule**

This rule states that the limit of the sum of two functions is equal to the sum of their limits:

$$\lim_{x \to a} [f(x) + g(x)] = \lim_{x \to a} f(x) + \lim_{x \to a} g(x).$$

All of the identities shown hold only if the limits exist.
**Product Rule**

This rule says that the limit of the product of two functions is the product of their limits (if they exist):

$$\lim_{x \to a} [f(x) \cdot g(x)] = \lim_{x \to a} f(x) \cdot \lim_{x \to a} g(x).$$

**Quotient Rule**

The limit of quotient of two functions is the quotient of their limits, provided that the limit in the denominator function is not zero:

$$\lim_{x \to a} \frac{f(x)}{g(x)} = \frac{\lim_{x \to a} f(x)}{\lim_{x \to a} g(x)}, \text{ if } \lim_{x \to a} g(x) \neq 0.$$
**Power Rule**

\[
\lim_{x \to a} [f(x)]^p = \left[ \lim_{x \to a} f(x) \right]^p,
\]

**Limit of an Exponential Function**

\[
\lim_{x \to a} b^{f(x)} = b^{\lim_{x \to a} f(x)}
\]

**Limit of a Logarithm of a Function**

\[
\lim_{x \to a} \log_b f(x) = \log_b \lim_{x \to a} f(x)
\]

(Where base \( b > 0 \))
L’HOSPITAL’S RULE

• Often we take the limit of \( \frac{f(n)}{g(n)} \) where both \( f(n) \) and \( g(n) \) tend to \( \infty \), or both \( f(n) \) and \( g(n) \) tend to 0.

• Such limits require L’Hospital’s rule

  • This rule says the limit of \( \frac{f(n)}{g(n)} \) in this case is the same as the limit of the derivative

• In other words, \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{d}{dn} \frac{f(n)}{g(n)} \)
USING THE LIMIT METHOD: EXERCISE 1

• Compare growth rate of $n^2$ and $n^2 - 7n - 30$

  \[
  \lim_{n \to \infty} \frac{n^2 - 7n - 30}{n^2}
  \]

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

  \[
  = 1
  \]

  So $n^2 - 7n - 30 \in \Theta(n^2)$
USING THE LIMIT METHOD: EXERCISE 2

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

\[
\lim_{n \to \infty} \frac{(\ln n)^2}{n^{1/2}} = \lim_{n \to \infty} \frac{\frac{d}{dn}(\ln n)^2}{\frac{d}{dn}n^{1/2}}
\]
USING THE LIMIT METHOD: EXERCISE 2

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

\[
\lim_{n \to \infty} \frac{d}{dn} \frac{(\ln n)^2}{n^{1/2}}
\]

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

\[
= \lim_{n \to \infty} 4 \frac{\ln n}{n^{1/2}}
\]

\[
= \lim_{n \to \infty} \frac{4}{n^{1/2}}
\]

\[
= 0
\]

• So, \((\ln n)^2 \in o(n^{1/2})\)
Additional Exercises

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

2. Compare the growth rates of the functions \(f(n) = n|\sin \pi n/2| + 1\) and \(g(n) = \sqrt{n}\).
SUMMATIONS AND SEQUENCES
**Algebra of Order Notations**

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

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

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

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

**“Summation” rules:** Suppose \( I \) is a finite set. Then

\[
O\left(\sum_{i \in I} f(i)\right) = \sum_{i \in I} O(f(i))
\]

\[
\Theta\left(\sum_{i \in I} f(i)\right) = \sum_{i \in I} \Theta(f(i))
\]

\[
\Omega\left(\sum_{i \in I} f(i)\right) = \sum_{i \in I} \Omega(f(i))
\]
Summation rules are commonly used in loop analysis.

Example:

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

\[
\sum_{i=0}^{n-1}(a + di) = na + \frac{dn(n-1)}{2} \in \Theta(n^2).
\]

**Geometric sequence:**

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

\[
\sum_{i=0}^{n-1} (a + di) r^i = \frac{a}{1 - r} - \frac{(a + (n - 1)d) r^n}{1 - r} + \frac{dr (1 - r^{n-1})}{(1 - r)^2}
\]

provided that \( r \neq 1 \).

Harmonic sequence:

\[
H_n = \sum_{i=1}^{n} \frac{1}{i} \in \Theta(\log n)
\]
Miscellaneous Formulae

\[ n! \in \Theta \left( n^{n+1/2} e^{-n} \right) \]
\[ \log n! \in \Theta(n \log n) \]

Another useful formula is

\[ \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}, \]

which implies that

\[ \sum_{i=1}^{n} \frac{1}{i^2} \in \Theta(1). \]

A sum of powers of integers when \( c \geq 1 \):

\[ \sum_{i=1}^{n} i^c \in \Theta(n^{c+1}). \]
LOGARITHM RULES
Logarithm Formulae

1. $\log_b xy = \log_b x + \log_b y$
2. $\log_b x/y = \log_b x - \log_b y$
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 $\log_c x \in \Theta(\log_b x)$

We typically omit the base, and just write $\Theta(\log x)$ for this reason.
RUNNING TIME ANALYSIS

WHAT I FEEL LIKE WHEN I RUN...

WHAT I'M PRETTY SURE I ACTUALLY LOOK LIKE...
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: $T_M^{\text{avg}}(n)$ denotes the average running time of program $M$ over all instances of size $n$:

$$T_M^{\text{avg}}(n) = \frac{1}{|\{I : \text{Size}(I) = n\}|} \sum_{\{I : \text{Size}(I) = n\}} T_M(I).$$

But how do we know how much time $M$ will take on input $I$?

We don’t know how much time an individual step in the program takes!
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?

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

  Unlikely... split large integers into **arrays** of words. Watch cost of operations!
LOOP ANALYSIS
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)
TWO BIG-O ANALYSIS STRATEGIES

• **Strategy 1**
  • Prove a $O$-bound and a matching $\Omega$-bound separately to get a $\Theta$-bound.

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

*Often easier (but not always)*
Algorithm: *LoopAnalysis1* (*n* : integer)

1. \( \text{sum} \leftarrow 0 \)
2. for \( i \leftarrow 1 \) to \( n \)
   - for \( j \leftarrow 1 \) to \( i \)
     - do \( \text{do} \)
       - \( \text{sum} \leftarrow \text{sum} + (i - j)^2 \)
       - \( \text{sum} \leftarrow \lfloor \text{sum}/i \rfloor \)
   - return \( (\text{sum}) \)
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 $\sum_{i=1}^{n} i$, with $\Theta(1)$ time per

Upper bound:

$$\sum_{i=1}^{n} O(i) \leq \sum_{i=1}^{n} O(n) = O(n^2).$$

Lower bound:

$$\sum_{i=1}^{n} \Omega(i) \geq \sum_{i=n/2}^{n} \Omega(i) \geq \sum_{i=n/2}^{n} \Omega(n/2) = \Omega(n^2/4) = \Omega(n^2).$$

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

Algorithm: LoopAnalysis1($n : integer$)

1. $sum \leftarrow 0$
2. for $i \leftarrow 1$ to $n$
   do $j \leftarrow 1$
   do $sum \leftarrow sum + (i - j)^2$
   do $sum \leftarrow \lfloor sum/i \rfloor$
3. return ($sum$)
Strategy 2: use Θ-bounds throughout the analysis

**Algorithm: LoopAnalysis1(n : integer)**

(1) $\text{sum} \leftarrow 0$
(2) for $i \leftarrow 1$ to $n$
    for $j \leftarrow 1$ to $i$
    do 
        do 
            $\text{sum} \leftarrow \text{sum} + (i - j)^2$
        $\text{sum} \leftarrow \lfloor \text{sum}/i \rfloor$
    do
(3) return ($\text{sum}$)

**Θ-bound analysis**

<table>
<thead>
<tr>
<th>Step</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>(2)</td>
<td>Complexity of inner for loop: $\Theta(i)$</td>
</tr>
<tr>
<td></td>
<td>Complexity of outer for loop: $\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$</td>
</tr>
<tr>
<td>(3)</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>

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

$$\sum_{i=1}^{n} \Theta(i) = \Theta\left(\sum_{i=1}^{n} i\right) = \Theta\left(\frac{n(n + 1)}{2}\right) = \Theta(n^2).$$
Consider this loop alone... number of loop iterations?

\[ j \] starts at \( i \) and is repeatedly divided by 2... this can happen only \( \Theta(\log i) \) times

So inner loop has runtime \( \Theta(\log i) \)

And the entire inner loop is executed for \( i = 1, 2, \ldots, n \)

So, we have \( T(n) \in \Theta(\sum_{i=1}^{n} \log i) \)

\[
T(n) \in O\left(\sum_{i=1}^{n} \log i\right) \subseteq O(n \log n)
\]

\[
T(n) \in \Omega\left(\sum_{i=1}^{n} \log i\right) \subseteq \Omega(n \log \frac{n}{2}) \subseteq \Omega(n \log n)
\]
Olive Garden waiter: Sir, you've already had 5 baskets of breadsticks
Me:

We're done when I say we're done

... another exercise in loop analysis?
EXAMPLE 3  (BENTLEY’S PROBLEM, SOLUTION 1)

```plaintext
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;
```
Strategy 1: big-O and big-Ω bounds

\[ T(n) \in \Theta(1) + \sum_{i=1}^{n} \sum_{j=i}^{n} \left( \Theta(1) + \sum_{k=i}^{j} \Theta(1) + \Theta(1) \right) \]

\[ T(n) \in \sum_{i=1}^{n} \sum_{j=i}^{n} \Theta(j - i) \in \Theta \left( \sum_{i=1}^{n} \sum_{j=i}^{n} (j - i) \right) \]

\[ T(n) \in O \left( \sum_{i=1}^{n} \sum_{j=i}^{n} (j - i) \right) \leq O \left( \sum_{i=1}^{n} \sum_{j=i}^{n} n \right) \]

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

\[ T(n) \in O(n^3) \]

This is the maximum number of iterations that could be performed in this loop.
Proving a big-Ω bound...

Recall:

\[ T(n) \in \Theta \left( \sum_{i=1}^{n} \sum_{j=i}^{n} (j - i) \right) \]

\[ T(n) \in \Omega \left( \sum_{i=1}^{n} \sum_{j=i}^{n} (j - i) \right) \]

\[ \geq \Omega \left( \sum_{i=1}^{n/2} \sum_{j=i}^{n} (j - i) \right) \]

\[ \geq \Omega \left( \sum_{i=1}^{n/2} \sum_{j=3n/4}^{n} (j - i) \right) \]

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

To get a good \( \Omega \)-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 \( \Omega(n) \) iterations! For example, when \( i \leq n/2 \)...

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

Code snippet:

```cpp
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;
```
Proving a big-Ω bound… continued

Recall:

\[ T(n) \in \Omega \left( \sum_{i=1}^{n/2} \sum_{j=3n/4}^{n} (j - i) \right) \]

\[ \geq \Omega \left( \sum_{i=1}^{n/2} \sum_{j=3n/4}^{n} \left( \frac{3n}{4} - \frac{n}{2} \right) \right) \]

\[ = \Omega \left( \sum_{i=1}^{n/2} \sum_{j=3n/4}^{n} \frac{n}{4} \right) \]

\[ \geq \Omega \left( \frac{n}{2} \cdot \frac{n}{4} \cdot \frac{n}{4} \right) = \Omega(n^3) \]

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 \( O(n^3) \) and \( \Omega(n^3) \), we have **proved** \( \Theta(n^3) \)