Big-O: Motivation

What is the running time of this algorithm? How many “primitive steps” are executed for an input of size \( n \)?

```java
sum = 0
for i = 1 to n
do
  for j = 1 to n
do
  end for
end for
```

The running time is \( T(n) = 2n^2 + n + 1 \).

For large values of \( n \), \( T(n) \) is less than some multiple of \( n^2 \). We say \( T(n) \) is \( O(n^2) \) and we typically don’t care about other terms.

Big-O: What it Is and Isn’t

- **Is**: a way to categorize growth rate of (non-negative) functions relative to other functions.
- **Is not**: “the running time of my function”

**Correct usage:**
- The running time of my algorithm in input of size \( n \) is \( T(n) \). Statement about algorithm only.
- \( T(n) \) is \( O(n^3) \). Statement about the function \( T(n) \) only.
- The running time of my algorithm is \( O(n^3) \). About algorithm and \( T(n) \).

**Incorrect usage:**
- \( O(n^3) \) is the running time of my algorithm

Algorithm design

- Formulate the problem precisely
- Design an algorithm to solve the problem
- Prove the algorithm is correct
- Analyze the algorithm’s running time

Big-O: Formal Definition

**Definition**: The function \( T(n) \) is \( O(f(n)) \) if there exist constants \( c \geq 0 \) and \( n_0 \geq 0 \) such that

\[
T(n) \leq cf(n) \quad \text{for all} \quad n \geq n_0
\]

We say that \( f \) is an asymptotic upper bound for \( T \).

**Examples**:
- If \( T(n) = n^2 + 100000n \) then \( T(n) \) is \( O(n^2) \)
- If \( T(n) = n^3 + n \log n \) then \( T(n) \) is \( O(n^3) \)
- If \( T(n) = 2\sqrt{n} \log n \) then \( T(n) \) is \( O(n) \)
- If \( T(n) = n^3 \) then \( T(n) \) is \( O(n^3) \) but it’s also \( O(n^3), O(n^5) \) etc.

Properties of Big-O Notation

**Claim (Transitivity)**: If \( f \) is \( O(g) \) and \( g \) is \( O(h) \), then \( f \) is \( O(h) \).

**Proof**: we know from the definition that

\[
\begin{align*}
  f(n) &\leq cg(n) \quad \text{for all} \quad n \geq n_0 \\
  g(n) &\leq ch(n) \quad \text{for all} \quad n \geq n'_0
\end{align*}
\]

Therefore

\[
\begin{align*}
  f(n) &\leq cg(n) \\
  &\leq c(c'h(n)) \quad \text{if} \quad n \geq n_0 \quad \text{and} \quad n \geq n'_0 \\
  &\leq c^{c'}h(n) \quad \text{if} \quad n \geq \max(n_0, n'_0)
\end{align*}
\]

Know how to do proofs using Big-O definition.
Properties of Big-O Notation

Claims (Additivity):
- If \( f \) is \( O(h) \) and \( g \) is \( O(h) \), then \( f + g \) is \( O(h) \).
- If \( f_1, f_2, \ldots, f_k \) are each \( O(h) \), then \( f_1 + f_2 + \ldots + f_k \) is \( O(h) \).
- If \( f \) is \( O(g) \), then \( f + g \) is \( O(g) \).

We’ll go through a couple of examples...

Other Useful Facts: Log vs. Poly vs. Exp

Fact: \( \log_b(n) \) is \( O(n^d) \) for all \( b \) and \( d \)

All polynomials grow faster than logarithm of any base

Fact: \( n^d \) is \( O(r^n) \) when \( r > 1 \)

Exponential functions grow faster than polynomials

Big-O sorting

Which grows faster?

\[
\begin{align*}
(n \log n)^3 & \text{ vs. } n^{4/3} \\
(\log n)^3 & \text{ vs. } n^{1/3} \\
\log n & \text{ vs. } n^{1/9}
\end{align*}
\]

- We know \( \log n \) is \( O(n^d) \) for all \( d \)
- \( \Rightarrow \) \( \log n \) is \( O(n^{1/9}) \)
- \( \Rightarrow \) \( n(\log n)^3 \) is \( O(n^{4/3}) \)

Apply transformations (monotone, invertible) to both functions. Try taking log.

Consequences of Additivity

- OK to drop lower order terms. E.g., if
  \[
  f(n) = 4.1n^3 + 23n + n \log n
  \]
  then \( f(n) \) is \( O(n^3) \)
- Polynomials: Only highest degree term matters. E.g., if
  \[
  f(n) = a_0 + a_1n + a_2n^2 + \ldots + a_dn^d, \quad a_d > 0
  \]
  then \( f(n) \) is \( O(n^d) \)

Logarithm review

Definition: \( \log_b(a) \) is the unique number \( c \) such that \( b^c = a \)

Informally: the number of times you can divide \( a \) into \( b \) parts until each part has size one

Properties:
- Log of product \( \rightarrow \) sum of logs
  \[
  \log(xy) = \log x + \log y
  \]
- \( \log_b(x^k) = k \log_b x \)

\( \log_b(\cdot) \) is inverse of \( b^{(\cdot)} \)

- \( \log_b(b^n) = n \)
- \( b^{\log_b(n)} = n \)

When using big-O, it’s OK not to specify base. Assume \( \log_2 \) if not specified.

Big-Ω Motivation

Algorithm \( \text{foo} \)

\[
\begin{align*}
\text{for } i=1 \text{ to } n & \text{ do }
\text{for } j=1 \text{ to } n & \text{ do }
\text{do something...}
\text{end for}
\text{end for}
\text{end for}
\end{align*}
\]

Fact: run time is \( O(n^3) \)

What is wrong?

Algorithm \( \text{bar} \)

\[
\begin{align*}
\text{for } i=1 \text{ to } n & \text{ do }
\text{for } j=1 \text{ to } n & \text{ do }
\text{do something else.}
\text{end for}
\text{end for}
\text{end for}
\end{align*}
\]

Fact: run time is \( O(n^3) \)

Conclusion: \( \text{foo} \) and \( \text{bar} \) have the same asymptotic running time.
More Big-Ω Motivation

Algorithm sum-product
sum = 0
for i = 1 to n do
    for j = i to n do
        sum += A[i]*A[j]
    end for
end for

What is the running time of sum-product?
Easy to see it is \( O(n^2) \). Could it be better? \( O(n) \)?

Big-Ω

Informally: \( T \) grows at least as fast as \( f \)

Definition: The function \( T(n) \) is \( \Omega(f(n)) \) if there exist constants \( c \geq 0 \) and \( n_0 \geq 0 \) such that
\[
T(n) \geq cf(n) \quad \text{for all } n \geq n_0
\]
\( f \) is an asymptotic lower bound for \( T \)

Exercise review

Hard way
- Count exactly how many times the loop executes
  \[
  1 + 2 + \ldots + n = \frac{n(n + 1)}{2} = \Omega(n^2)
  \]

Easy way
- Ignore all loop executions where \( i > n/2 \) or \( j < n/2 \)
- The inner statement executes at least \( (n/2)^2 = \Omega(n^2) \) times

Big-Θ example

How do we correctly compare the running time of these algorithms?

Algorithm foo
for i = 1 to n do
    for j = 1 to n do
        do something...
    end for
end for

Algorithm bar
for i = 1 to n do
    for j = 1 to n do
        for k = 1 to n do
            do something else..
        end for
    end for
end for

Answer: foo is \( \Theta(n^2) \) and bar is \( \Theta(n^3) \). They do not have the same asymptotic running time.
Additivity Revisited

Suppose \( f \) and \( g \) are two (non-negative) functions and \( f = O(g) \)

Old version: Then \( f + g = O(g) \)
New version: Then \( f + g = \Theta(g) \)

Example:
\[
\frac{n^2}{g} + 42n + n \log n \leq f \quad \text{is} \quad \Theta(n^2)
\]