Asymptotic Notation, Recurrences

Asymptotic growth rates

Let \( g : \mathbb{N} \rightarrow \mathbb{R} \).

**O-notation:** \( O(g) \) is the set of all functions \( f : \mathbb{N} \rightarrow \mathbb{R} \) for which there are constants \( c > 0 \) and \( n_0 \geq 0 \) such that

\[
0 \leq f(n) \leq c \cdot g(n), \quad \text{for all } n \geq n_0.
\]

"Rate of change of \( f(n) \) is at most that of \( g(n) \)"

**Ω-notation:** \( \Omega(g) \) is the set of all functions \( f : \mathbb{N} \rightarrow \mathbb{R} \) for which there are constants \( c > 0 \) and \( n_0 \geq 0 \) such that

\[
0 \leq c \cdot g(n) \leq f(n), \quad \text{for all } n \geq n_0.
\]

"Rate of change of \( f(n) \) is at least that of \( g(n) \)"

**Θ-notation:** \( \Theta(g) \) is the set of all functions \( f : \mathbb{N} \rightarrow \mathbb{R} \) for which there are constants \( c_1, c_2 > 0 \) and \( n_0 \geq 0 \) such that

\[
0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n), \quad \text{for all } n \geq n_0.
\]

"Rate of change of \( f(n) \) and \( g(n) \) are about the same"

Examples

▶ Let \( f(n) = 0.01 \cdot n^2 \) and \( g(n) = n \). Then \( g = O(f) \).
▶ Let \( f(n) = \ln(n) \) and \( g(n) = n \). Then \( g = \Omega(f) \).
▶ Let \( f(n) = 10n + \ln(n) \) and \( g(n) = n \). Then \( g = \Theta(f) \).

Sometimes \( O(\ldots) \) appears within a formula, rather than simply forming the right hand side of an equation. We make sense of this by thinking of \( O(\ldots) \) as standing for some anonymous (but fixed) function from the set of the same name.

For example, \( h(n) = 2^{O(n)} \) means \( \exists c > 0, n_0 \in \mathbb{N} \) such that

\[
h(n) \leq 2^{cn} \text{ for all } n > n_0.
\]

Consequences

Suppose \( f(n) = O(g(n)) \) AND \( g(n) = O(f(n)) \). What can we say?

What if \( f(n) = O(g(n)) \) AND \( f(n) = \Omega(g(n)) \)?

Various consequences of the above conventions:

\[
\Theta(n) \times \Theta(n^2) = \Theta(n^3),
\]

\[
\Theta(n) + \Theta(n^2) = \Theta(n^2),
\]

\[
\Theta(n) + \Theta(n) = \Theta(n).
\]
Array A is indexed from $j = 1$ to $n = \text{length}[A]$ (different from Java).

---

**Algorithm** Insertion-Sort($A$)

1. for $j \leftarrow 2$ to $\text{length}[A]$ do
2.    key $\leftarrow A[j]$
   (now insert $A[j]$ into the sorted sequence $A[1 \ldots j - 1]$)
3. $i \leftarrow j - 1$
4. while $i > 0$ and $A[i] > key$ do
5.         $A[i + 1] \leftarrow A[i]$
6.         $i \leftarrow i - 1$
7. $A[i + 1] \leftarrow key$

Array $A$ is indexed from $j = 1$ to $n = \text{length}[A]$ (different from Java).

---

**Algorithm** Merge-Sort($A, p, r$)

1. if $p < r$ then
2.    $q \leftarrow \lfloor \frac{p + r}{2} \rfloor$
3.    Merge-Sort($A, p, q$)
4.    Merge-Sort($A, q + 1, r$)
5.    Merge($A, p, q, r$)

The for-loop on line 1 is iterated $n - 1$ times
- For each execution of the for, the while does $\leq j$ iterations;
- Each of the comparisons/assignments requires only $O(1)$ basic steps;
- Therefore the total number of steps (=time) is at most
  \[ O(1) \sum_{j=1}^{n} j = O(1) \frac{n(n + 1)}{2} = O(n^2). \]
- This is essentially tight - sorting the list $n, n - 1, n - 2, \ldots, 3, 2, 1$ takes $\Omega(n^2)$ time.

---

**Algorithm** Merge($A, p, q, r$)

1. $n \leftarrow r - p + 1$, $n_1 \leftarrow q - p + 1$, $n_2 \leftarrow r - q$
2. create an array $B$ of length $n$
3. $i \leftarrow p$, $j \leftarrow q + 1$, $k \leftarrow 1$
4. while $((i \leq q) \&\& (j \leq r))$
5.    if $((j > r) \&\& ((i \leq q) \&\& (A[i] \leq A[j])))$
6.        $B[k] \leftarrow A[i]$
7.        $i \leftarrow i + 1$
8.    else
10.       $j \leftarrow j + 1$
11.      $k \leftarrow k + 1$
12. for $i = 1$ to $n$
13.    $A[(p - 1) + i] \leftarrow B[i]$
Analysis of Merge

We have \( n = (r - p) + 1, n_1 = (q - p) + 1, n_2 = r - q \) (note \( n = n_1 + n_2 \)).

Merge carries out the following steps:

- Initialisation/maintenance work in steps 1., 2., 3., uses \( 3 + n + 3 \) operations (\( n \) for setting up \( B \)).
- Over all \( n \) iterations of while, line 4. will carry out between \( n \) and \( n + n_2 \) index comparisons.
- Over all \( n \) iterations of while, line 5 will carry out between \( n \) and \( n + n_1 \) index comparisons and between \( n_1 \) and \( n \) key comparisons.
- Over all \( n \) iterations of while, lines 6.-11. will carry out \( 2n \) index updates and \( n \) copy operations (keys being copied into \( B \)).
- Lines 12.-13. take 2\( n \) steps.

Therefore the running-time of Merge satisfies the following:

\[
8n + n_1 + 6 \leq T_{\text{Merge}}(n : n_1, n_2) \leq 10n + n_1 + n_2 + 6
\]

We can express a neater bound as

\[
8n \leq T_{\text{Merge}}(n : n_1, n_2) \leq 14n.
\]

Solving recurrences

Methods for deriving/verifying solutions to recurrences:

- Induction Guess the solution and verify by induction on \( n \).
  Lovely if your recurrence is “NICE” enough that you can guess-and-verify. Rare.

- Unfold and sum “Unfold” the recurrence by iterated substitution on the “neat” values of \( n \) (often power of 2 case). At some point a pattern emerges. The “solution” is obtained by evaluating a sum that arises from the pattern.
  Since the pattern is just for the “neat” \( n \), the method is rigorous only if we verify the solution (e.g., by a direct induction proof).
  Often the only way to do the PROOF neatly is to RELATE to “neat” values of \( n \) ... sometimes powers-of-2

- “Master Theorem” Match the recurrence against a template. Read off the solution from the Master Theorem.

Upper bounds by first principles

Proof by “first principles”

When working from first principles, need to replace “extra work” terms (\( \Theta(n) \) for MergeSort) by terms with explicit constants.

So we check slide 10 again.

\[
T_{\text{MS}}(n) \leq \begin{cases} 
1 & \text{if } n = 1, \\
T_{\text{MS}}(\lceil n/2 \rceil) + T_{\text{MS}}(\lfloor n/2 \rfloor) + 14n & \text{if } n > 1.
\end{cases} \tag{1}
\]
Upper bounds by first principles

Proof by “first principles”
When working from first principles, need to replace “extra work” terms ($\Theta(n)$ for MergeSort) by terms with explicit constants.
So we check slide 10 again.

Unfold-and-sum will give a “guess” for the upper bound:

$$T_{MS}(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ T_{MS}([n/2]) + T_{MS}([n/2]) + 14n & \text{if } n > 1. \end{cases}$$  \hspace{1cm} (1)

**Upper bound for MergeSort ($n$ a power-of-2)**

$T'_{MS}(n) = \begin{cases} 1 & \text{if } n = 1, \\ T'_{MS}([n/2]) + T'_{MS}([n/2]) + 14n & \text{if } n > 1. \end{cases}$  \hspace{1cm} (2)

Claim (powers of 2): $T'_{MS}(n) = 14n \lg(n) + n$ if $n = 2^k$ for some $k \in \mathbb{N}$

ADS (2018/19) – Lectures 2 and 3 – slide 12

Proof by “first principles”
When working from first principles, need to replace “extra work” terms ($\Theta(n)$ for MergeSort) by terms with explicit constants.
So we check slide 10 again.

Unfold-and-sum will give a “guess” for the upper bound:

$$T_{MS}(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ T_{MS}([n/2]) + T_{MS}([n/2]) + 14n & \text{if } n > 1. \end{cases}$$  \hspace{1cm} (1)

**Upper bound for MergeSort ($n$ a power-of-2)**

$T_{MS}'(n) = \begin{cases} 1 & \text{if } n = 1, \\ T_{MS}'([n/2]) + T_{MS}'([n/2]) + 14n & \text{if } n > 1. \end{cases}$  \hspace{1cm} (2)
Upper bound for $\text{MergeSort}$ \((n \text{ a power-of-2})\)

\[
T'_{\text{MS}}(n) = \begin{cases} 
1 & \text{if } n = 1, \\
T'_{\text{MS}}(\lceil n/2 \rceil) + T'_{\text{MS}}(\lfloor n/2 \rfloor) + 14n & \text{if } n > 1.
\end{cases}
\] (2)

**Claim (powers of 2):** \(T'_{\text{MS}}(n) = 14n \lg(n) + n\) if \(n = 2^k\) for some \(k \in \mathbb{N}\)

**Proof (for powers of 2):**
Base case \(k = 0\): direct from recurrence \((14 \cdot 2^0 \cdot \lg(2^0) + 2^0 = 14 \cdot 1 \cdot 0 + 1 = 1, \) as required).
Induction Hypothesis (IH): Upper bound holds for \(n = 2^{k-1}\).

Induction Step: Now consider \(n = 2^k\) and apply the recurrence:

\[
T'_{\text{MS}}(n) = T'_{\text{MS}}(\lceil 2^{k-1} \rceil) + T'_{\text{MS}}(\lfloor 2^{k-1} \rfloor) + 14n
\]
Upper bound for MergeSort \((n \text{ a power-of-2})\)

\[
T'_\text{MS}(n) = \begin{cases} 
1 & \text{if } n = 1, \\
T'_\text{MS}(\lceil n/2 \rceil) + T'_\text{MS}(\lfloor n/2 \rfloor) + 14n & \text{if } n > 1. 
\end{cases}
\tag{2}
\]

**Claim** (powers of 2): \(T'_\text{MS}(n) = 14n \lg(n) + n\) if \(n = 2^k\) for some \(k \in \mathbb{N}\)

**Proof** (for powers of 2):

Base case \(k = 0\): direct from recurrence \((14 \cdot 2^0 \cdot \lg(2^0) + 2^0 = 14 \cdot 1 \cdot 0 + 1 = 1,\) as required.

Induction Hypothesis (IH): Upper bound holds for \(n = 2^{k-1}\).

Induction Step: Now consider \(n = 2^k\) and apply the recurrence:

\[
T'_\text{MS}(n) = T'_\text{MS}(\lceil 2^{k-1} \rceil) + T'_\text{MS}(\lfloor 2^{k-1} \rfloor) + 14n
\]
\[
= 2 \cdot T'_\text{MS}(2^{k-1}) + 14n
\]
\[
= 2 \cdot 2^{k-1}(14 \lg(2^{k-1}) + 1) + 14n \quad \text{(using (IH))}
\]
\[
= n \cdot 14 \lg(n/2) + n + 14n
\]

AS REQUIRED.
Upper bounds for general $n$

Three steps for turning a “proof for the neat case” into a “proof for all $n$”.

- **STEP 1**: Prove an exact expression for “neat” $n$ for an equality version $T'(\cdot)$ of the recurrence.
  Done for $T'_{MS}(n)$ (the proof for $T'_{MS}(n)$ on slide 14). “Neat” was powers-of-2.

- **STEP 2**: Prove that the equality version of the recurrence is monotone increasing; i.e., that we have $T'(n) \leq T'(m)$ for all $n, m$ with $n < m$ (not just for “neat” $n, m$).
  This step is why we need to introduce an “equality version” (to prove STEP 2 we will need to work with $T'(n) = T'(m)$).

- **STEP 3**: For “not-neat $n$”, choose a close-by “neat $\hat{n}$” (for proving $O(\cdot)$ bounds, $\hat{n}$ should be larger; for $\Omega(\cdot)$ bounds, $\hat{n}$ should be smaller).
  Then apply monotonicity (STEP 2) to show a relationship between $T'(n)$ and $T'(\hat{n})$, and then substitute the exact expression (from STEP 1) to $T'(\hat{n})$ to work out an upper bound for $T'(n)$.

**Upper bound for MergeSort (general $n$)**

**STEP 2**: Prove that $T'_{MS}(n)$ is monotone increasing.

The proof is by Induction.

**Claim:**
If $n \in \mathbb{N}$ then $T'_{MS}(n) < T'_{MS}(m)$ for all $n < m$.

**Upper bound for MergeSort (general $n$)**

**STEP 2**: Prove that $T'_{MS}(n)$ is monotone increasing.

The proof is by Induction.

**Claim:**
If $n \in \mathbb{N}$ then $T'_{MS}(n) < T'_{MS}(m)$ for all $n < m$.

**Induction Hypothesis (IH):** Claim holds for all $n = 1, \ldots, h$ (with any $m > n$).

**Base Case ($h = 1$):**
$T'_{MS}(1) = 1$.

For $m \geq 2$, $T'_{MS}(m) \geq 14m \geq 28$, and $28 > T'_{MS}(1)$, as needed.
Upper bound for **MergeSort** (general $n$) cont’d.

**STEP 2 cont’d.**

**Induction Step ($n$):** Suppose true for all $n \in \mathbb{N}, n = 1, \ldots, h$. We know $n \geq 2$, so the recurrence for $n$ is

$$T'_{\text{MS}}(n) = T'_{\text{MS}}(\lfloor n/2 \rfloor) + T'_{\text{MS}}(\lfloor n/2 \rfloor) + 14n.$$  \hfill (3)

We are considering $m > n$ (so definitely $m \geq 2$), and the recurrence for $m$ is

$$T'_{\text{MS}}(m) = T'_{\text{MS}}(\lfloor m/2 \rfloor) + T'_{\text{MS}}(\lfloor m/2 \rfloor) + 14m.$$
Upper bound for MergeSort (general $n$) cont’d.

**STEP 2 cont’d.**

**Induction Step ($n$):** Suppose true for all $n \in \mathbb{N}$, $n = 1, \ldots, h$. Consider $n = h + 1$.

We know $n \geq 2$, so the recurrence for $n$ is

$$T'_{\text{MS}}(n) = T'_{\text{MS}}(\lfloor n/2 \rfloor) + T'_{\text{MS}}(\lceil n/2 \rceil) + 14n.$$  \hspace{1cm} (3)

We are considering $m > n$ (so definitely $m \geq 2$), and the recurrence for $m$ is

$$T'_{\text{MS}}(m) = T'_{\text{MS}}(\lfloor m/2 \rfloor) + T'_{\text{MS}}(\lceil m/2 \rceil) + 14m.$$  \hspace{1cm} (3)

$n \geq 2$ implies $\lfloor \frac{n}{2} \rfloor = \lfloor \frac{h+1}{2} \rfloor < n$ (need strict $<$) so $\lfloor \frac{n}{2} \rfloor \in \{1, \ldots, h\}$. So the (IH) can be applied to $\lfloor \frac{n}{2} \rfloor$ with appropriate $m$-values. $m > n$ implies $\lfloor \frac{m}{2} \rfloor \geq \lfloor \frac{n}{2} \rfloor$, so

- either $\lfloor \frac{n}{2} \rfloor = \lfloor \frac{m}{2} \rfloor$, and hence $T'_{\text{MS}}(\lfloor \frac{n}{2} \rfloor) = T'_{\text{MS}}(\lfloor \frac{m}{2} \rfloor)$.
- or else $\lfloor \frac{m}{2} \rfloor > \lfloor \frac{n}{2} \rfloor$ and taking this together with $\lfloor \frac{n}{2} \rfloor \leq h$, the (IH) implies that $T'_{\text{MS}}(\lfloor \frac{n}{2} \rfloor) < T'_{\text{MS}}(\lfloor \frac{m}{2} \rfloor)$.

Same argument goes through with $\lfloor \frac{m}{2} \rfloor$. Hence the (IH) shows that each of the first two terms for $T'_{\text{MS}}(n)$ are $\leq$ the corresponding terms for $T'_{\text{MS}}(m)$.

But also $14n < 14m$, so $\ldots \Rightarrow T'_{\text{MS}}(n) < T'_{\text{MS}}(m)$. 

\hspace{1cm} ADS (2018/19) – Lectures 2 and 3 – slide 16
Upper bound for MergeSort (general \( n \)) cont’d.

STEP 3: Choose a “power of 2” to relate to \( n \).

- Want an upper bound, so need a power of 2 greater than \( n \).
- So define \( \hat{n} = 2^{\lceil \lg(n) \rceil} \) (this will be “\( m \)”).

Hence \( T_{\text{MS}}(n) = O(n \log(n)) \), and (of course) \( T_{\text{MS}}(n) = O(n \log(n)) \).

ADS (2018/19) – Lectures 2 and 3 – slide 17
STEP 3: Choose a “power of 2” to relate to $n$.

- Want an upper bound, so need a power of 2 greater than $n$.
- So define $\hat{n} = 2^{\lceil \log(n) \rceil}$ (this will be “$m$”).
- We know $n \leq \hat{n}$ but $\hat{n} < 2n$.
- Monotonicity property from STEP 2 tells us $T_{MS}'(n) \leq T_{MS}'(\hat{n})$
- Proof of Upper bound for POWERS of 2 tells us $T_{MS}'(\hat{n}) \leq 14\hat{n}\log(\hat{n}) + \hat{n}$.
- By $\hat{n} < 2n$, we get
  $$T_{MS}'(n) \leq T_{MS}'(\hat{n}) = 14\hat{n}(\log(\hat{n})) + \hat{n} < 14(2n)\log(2n) + 2n = 28n\log(n) + 30n.$$
Upper bound for MergeSort (general $n$) cont’d.

**STEP 3:** Choose a “power of 2” to relate to $n$.

- Want an upper bound, so need a power of 2 greater than $n$.
- So define $\hat{n} = 2^{\lceil \lg(n) \rceil}$ (this will be “$m$”).
- We know $n \leq \hat{n}$ but $\hat{n} < 2n$.
- Monotonicity property from STEP 2 tells us $T_{\text{MS}}'(n) \leq T_{\text{MS}}'(\hat{n})$
- Proof of Upper bound for POWERS of 2 tells us $T_{\text{MS}}'(\hat{n}) \leq 14\hat{n}\lg(\hat{n}) + \hat{n}$.
- By $\hat{n} < 2n$, we get

$$T_{\text{MS}}'(n) \leq T_{\text{MS}}'(\hat{n}) = 14\hat{n}(\lg(\hat{n})) + \hat{n} < 14(2n)\lg(2n) + 2n = 28n\lg(n) + 30n.$$  

So for any $n \in \mathbb{N}$ we have $T_{\text{MS}}'(n) \leq 28n\lg(n) + 30n$.
Hence $T_{\text{MS}}'(n) = O(n\lg(n))$, and (of course) $T_{\text{MS}}(n) = O(n\lg(n))$.

Proving a lower bound

The “first principles” proof is essentially a direct proof of a sub-case of the Master Theorem.

Slide 15 described the usual structure of proving $O(\cdot)$ bounds for general $n \in \mathbb{N}$. When wanting to instead give a “first principles” proof of $\Omega(\cdot)$ for a recurrence $T(n)$, there are slight differences:

- (different) Consider an equality version $T'(\cdot)$ of the recurrence $T(\cdot)$ such that $T(n) \geq T'(n)$ holds for all $n \in \mathbb{N}$.
- (same) **STEP 1:** Prove an exact expression for $T'$ for the “NEAT” case (power-of-2 here, but would be power-of-$d$ if $\lceil n/d \rceil, \lceil n/d \rceil$ was involved)
- (same) **STEP 2:** Prove $T'(n)$ is monotonically increasing with $n$ for general $n$.
- (different) **STEP 3:** Consider the closest power-of-$d$ less than $n$, say $\hat{n}$, for a non-neat $n \in \mathbb{N}$. Then exploit $T(n) \geq T'(n)$ (by definition).

$T'(n) \geq T'(\hat{n})$ (from STEP 2), and then substitute in the exact expression for $T'(\hat{n})$ (because $\hat{n}$ is “NEAT”) and work from there.

Reading Assignment

Inf2B ADS Lecture Notes 2 and 8.  
[CLRS] Sections 2.1, 2.2 and 2.3 (of 3rd or 2nd edition). Also Section 3.1 (omitting the bits on the little-o and little-$\omega$ notation at the end).  
(all this material should be familiar from Inf2B and your math classes)