Solving Recurrences I
COMS10007 2020, Lecture 13

Dr. John Lapinskas
(substituting for Dr. Christian Konrad)

March 16th 2020
Divide-and-conquer algorithms

Many algorithms in this course (and in general!) follow the **divide-and-conquer** approach:

1. **Divide** the problem into smaller instances of the same problem.
2. **Conquer** the subproblems by solving them, either recursively or directly.
3. **Combine** the solutions to the subproblems into a solution for the original problem.

For example:
- Mergesort.
- Quicksort.
- The maximum subarray algorithm.
- Binary search.
- Fast-Peak-Finding.
Many algorithms in this course (and in general!) follow the \textit{divide-and-conquer} approach:

1. **Divide** the problem into smaller instances of the same problem.
2. **Conquer** the subproblems by solving them, either recursively or directly.
3. **Combine** the solutions to the subproblems into a solution for the original problem.

For example:
- Mergesort.
- Quicksort.
- The maximum subarray algorithm.
- Binary search.
- \texttt{Fast-Peak-Finding}.
Example: Merge sort

Recall: Merge Sort

1. Divide
   Split input array $A$ of length $n$ into subarrays $A_1 = A[0, \lfloor n/2 \rfloor]$ and $A_2 = A[\lceil n/2 \rceil + 1, n - 1]$
Example: Merge sort

Recall: Merge Sort

1. **Divide** $A \rightarrow A_1$ and $A_2$

2. **Conquer**
   Sort $A_1$ and $A_2$ recursively using the same algorithm

![Diagram showing the division and sorting process](image)

**Runtime:**
- $T(1) = O(1)$
- $T(n) = 2T(n/2) + O(n)$
Example: Merge sort

Recall: Merge Sort

1 Divide $A \rightarrow A_1$ and $A_2$

2 Conquer Solve $A_1$ and $A_2$

3 Combine

Combine sorted subarrays $A_1$ and $A_2$ and obtain sorted array $A$

\[
\begin{array}{cccccc}
2 & 3 & 7 & 7 & 8 & 9 & 12 & 15 \\
\end{array}
\]

\[
\begin{array}{cccc}
2 & 7 & 9 & 12 \\
\end{array} \quad \begin{array}{cccc}
3 & 7 & 8 & 15 \\
\end{array}
\]
Example: Merge sort

Recall: Merge Sort

1 Divide $A \rightarrow A_1$ and $A_2$

2 Conquer Solve $A_1$ and $A_2$

3 Combine

Combine sorted subarrays $A_1$ and $A_2$ and obtain sorted array $A$

![Diagram of array division and combination]

Runtime: (assuming that $n$ is a power of 2)

$$T(1) = O(1)$$
$$T(n) = 2T(n/2) + O(n)$$
Recurrences

- Divide-and-conquer algorithms naturally lead to recurrences (or “recurrence relations”) like that one.
- How can we solve them? Or at least get a decent upper bound?

Methods for solving recurrences

- Recursion-tree method (as used for mergesort and max subarray). Often has too many awkward details (e.g. floors and ceilings, pivots), but great for getting intuition.
- Substitution method (this lecture). Very powerful, but needs a reasonable initial guess.
- The “Master Theorem”. Only applies to recurrences of the form $T(n) = aT(n/b) + f(n)$, but makes things trivial when it does apply. Not covered in this course.

Generally: use recursion-tree to get a guess for substitution!
How to solve recurrences?

Recurrences

- Divide-and-conquer algorithms naturally lead to recurrences (or “recurrence relations”) like that one.
- How can we solve them? Or at least get a decent upper bound?

Methods for solving recurrences

- Recursion-tree method (as used for mergesort and max subarray).
  Often has too many awkward details (e.g. floors and ceilings, pivots), but great for getting intuition.
- Substitution method (this lecture).
  Very powerful, but needs a reasonable initial guess.
- The “Master Theorem”.
  Only applies to recurrences of the form \( T(n) = aT(n/b) + f(n) \), but makes things trivial when it does apply. Not covered in this course.

Generally: use recursion-tree to get a guess for substitution!
The substitution method

1. Remove the O-notation from the recurrence.
2. Guess a partial form of the solution (with some unknown constants).
3. Use mathematical induction to show the solution works for the right choice of constants.

Dealing with O-notation can introduce some added complications...
The substitution method

1. Remove the O-notation from the recurrence.
2. Guess a partial form of the solution (with some unknown constants).
3. Use mathematical induction to show the solution works for the right choice of constants.

Dealing with O-notation can introduce some added complications...

Example: The recurrence from mergesort (when $n$ is a power of two).

\[ T(1) = O(1), \]
\[ T(n) = 2T(n/2) + O(n). \]
The substitution method

1. Remove the O-notation from the recurrence.
2. Guess a partial form of the solution (with some unknown constants).
3. Use mathematical induction to show the solution works for the right choice of constants.

Dealing with O-notation can introduce some added complications...

**Example:** The recurrence from mergesort (when $n$ is a power of two).

\[
T(1) = O(1),
\]
\[
T(n) = 2T(n/2) + O(n).
\]

**Step 1:** Replace the O-notation by constants.
The substitution method

1. Remove the O-notation from the recurrence.
2. Guess a partial form of the solution (with some unknown constants).
3. Use mathematical induction to show the solution works for the right choice of constants.

Dealing with O-notation can introduce some added complications...

Example: The recurrence from mergesort (when $n$ is a power of two).

$$T(1) = O(1), \quad \rightarrow \quad T(n) \leq c_1 \quad \text{for all } n \leq n_0,$$

$$T(n) = 2T(n/2) + O(n). \quad T(n) \leq 2T(n/2) + c_2n \quad \text{for all } n > n_0.$$

Step 1: Replace the O-notation by constants. Remember, $f(n) \in O(g(n))$ means that there exist $C$ and $n_0$ such that for all $n \geq n_0, f(n) \leq Cg(n)$. 
The substitution method

The substitution method

1. Remove the O-notation from the recurrence.
2. Guess a partial form of the solution (with some unknown constants).
3. Use mathematical induction to show the solution works for the right choice of constants.

Dealing with O-notation can introduce some added complications...

Example: The recurrence from mergesort (when \(n\) is a power of two).

\[
\begin{align*}
T(1) &= O(1), & \quad \rightarrow \quad T(n) &\leq c_1 \quad \text{for all } n \leq n_0, \\
T(n) &= 2T(n/2) + O(n), & \quad T(n) &\leq 2T(n/2) + c_2 n \quad \text{for all } n > n_0.
\end{align*}
\]

Step 1: Replace the O-notation by constants. Remember, \(f(n) \in O(g(n))\) means that there exist \(C\) and \(n_0\) such that for all \(n \geq n_0\), \(f(n) \leq Cg(n)\).

For mergesort specifically, we can take \(n_0 = 1\).
The substitution method

1. Remove the O-notation from the recurrence.
2. Guess a partial form of the solution (with some unknown constants).
3. Use mathematical induction to show the solution works for the right choice of constants.

Dealing with O-notation can introduce some added complications...

**Example:** The recurrence from mergesort (when $n$ is a power of two).

\[
T(1) = O(1), \quad \Rightarrow \quad T(1) \leq c_1,
\]
\[
T(n) = 2T(n/2) + O(n). \quad T(n) \leq 2T(n/2) + c_2n \text{ for all } n > 1.
\]

**Step 1:** Replace the O-notation by constants. Remember, $f(n) \in O(g(n))$ means that there exist $C$ and $n_0$ such that for all $n \geq n_0$, $f(n) \leq Cg(n)$.

For mergesort specifically, we can take $n_0 = 1$. 
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1. \]

**Step 2:** Guess a bound. Here, guess \( T(n) \leq Cn \log n \) for some \( C > 0 \).

**Step 3:** Prove it works by induction.

**Base case \( n = 1 \):** \( T(1) \leq c_1 \), and \( C \cdot 1 \log(1) = 0 > c_1 \).
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1. \]

**Step 2:** Guess a bound. Here, guess \( T(n) \leq Cn \log n \) for some \( C > 0 \).

**Step 3:** Prove it works by induction.

**Base case** \( n = 1 \): \( T(1) \leq c_1 \), and \( C \cdot 1 \log(1) = 0 > c_1 \)... wait, no. :-(
The substitution method

\[
T(1) \leq c_1, \\
T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1.
\]

**Step 2:** Guess a bound. Here, guess \( T(n) \leq Cn \log n \) for some \( C > 0 \).

**Step 3:** Prove it works by induction.

**Base case \( n = 1 \):** \( T(1) \leq c_1 \), and \( C \cdot 1 \log(1) = 0 > c_1 \)...

Wait, no. :-(

But it’s fine! We’re only trying to prove \( T(n) = O(n \log n) \), which means we need \( T(n) \leq Cn \log n \) for all \( n \geq n_0 \) (for some \( C, n_0 \) of our choice).

We **don’t** need \( T(1) \leq C \cdot 1 \log 1 \). We can just take \( n_0 = 2 \).

**Key point:** Since we’re only going for asymptotic results, not exact results, we can choose any base case we want.
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1. \]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \) for all \( n \geq 2 \).

Note that we haven’t fixed a value for \( C \) yet — we’ll see what values work over the course of the proof.
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1. \]

\[ \text{Step 3: Prove by induction that } T(n) \leq Cn \log n \text{ for all } n \geq 2. \]

Note that we haven’t fixed a value for \( C \) yet — we’ll see what values work over the course of the proof.

**Base case \( n = 2 \):** We have

\[ T(2) \leq 2T(1) + c_2 \cdot 2 \leq 2(c_1 + c_2). \]
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1. \]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \text{ for all } n \geq 2. \)

Note that we haven’t fixed a value for \( C \) yet — we’ll see what values work over the course of the proof.

**Base case \( n = 2 \):** We have

\[ T(2) \leq 2T(1) + c_2 \cdot 2 \leq 2(c_1 + c_2), \]
\[ C \cdot 2 \log 2 = 2C. \]
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1. \]

\textbf{Step 3:} Prove by induction that \( T(n) \leq Cn \log n \text{ for all } n \geq 2. \)

Note that we haven’t fixed a value for \( C \) yet — we’ll see what values work over the course of the proof.

\textbf{Base case } \( n = 2 \): We have

\[ T(2) \leq 2T(1) + c_2 \cdot 2 \leq 2(c_1 + c_2), \]
\[ C \cdot 2 \log 2 = 2C. \]

So \( T(2) \leq C \cdot 2 \log 2 \) as long as we choose \( C \geq c_1 + c_2. \) \( \checkmark \)
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2n \quad \text{for all } n > 1. \]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \) for all \( n \geq 2 \).

**Base case \( n = 2 \):** Requires \( C \geq c_1 + c_2 \).
The substitution method

\[
T(1) \leq c_1, \\
T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1.
\]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \) for all \( n \geq 2 \).

**Base case** \( n = 2 \): Requires \( C \geq c_1 + c_2 \).

\[\checkmark\]

**Inductive step:** Suppose that for all \( 2 \leq n' < n \), \( T(n') \leq Cn' \log n' \). Then we must prove \( T(n) \leq Cn \log n \).
The substitution method

\[
T(1) \leq c_1, \\
T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1.
\]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \) for all \( n \geq 2 \).

**Base case \( n = 2 \):** Requires \( C \geq c_1 + c_2 \).

**Inductive step:** Suppose that for all \( 2 \leq n' < n \), \( T(n') \leq Cn' \log n' \). Then we must prove \( T(n) \leq Cn \log n \).

By the induction hypothesis,

\[
T(n) \leq 2T(n/2) + c_2 n
\]
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2n \text{ for all } n > 1. \]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \) for all \( n \geq 2 \).

**Base case \( n = 2 \):** Requires \( C \geq c_1 + c_2 \).

\[ \text{Inductive step: Suppose that for all } 2 \leq n' < n, T(n') \leq Cn' \log n'. \]

Then we must prove \( T(n) \leq Cn \log n \).

By the induction hypothesis,

\[ T(n) \leq 2T(n/2) + c_2n \leq 2C \cdot \frac{n}{2} \log(n/2) + c_2n \]
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1. \]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \) for all \( n \geq 2 \).

**Base case** \( n = 2 \): Requires \( C \geq c_1 + c_2 \). \( \checkmark \)

**Inductive step:** Suppose that for all \( 2 \leq n' < n \), \( T(n') \leq Cn' \log n' \).
Then we must prove \( T(n) \leq Cn \log n \).

By the induction hypothesis,

\[ T(n) \leq 2T(n/2) + c_2 n \leq 2C \cdot \frac{n}{2} \log(n/2) + c_2 n \]
\[ = Cn(\log(n) - 1) + c_2 n \]
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1. \]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \) for all \( n \geq 2 \).

**Base case** \( n = 2 \): Requires \( C \geq c_1 + c_2 \).

**Inductive step:** Suppose that for all \( 2 \leq n' < n \), \( T(n') \leq Cn' \log n' \).
Then we must prove \( T(n) \leq Cn \log n \).

By the induction hypothesis,

\[ T(n) \leq 2T(n/2) + c_2 n \leq 2C \cdot \frac{n}{2} \log(n/2) + c_2 n \]
\[ = Cn(\log(n) - 1) + c_2 n = Cn \log(n) + (c_2 - C)n. \]
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2n \text{ for all } n > 1. \]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \text{ for all } n \geq 2. \)

**Base case \( n = 2: \)** Requires \( C \geq c_1 + c_2. \)

**Inductive step:** Suppose that for all \( 2 \leq n' < n, \) \( T(n') \leq Cn' \log n'. \)
Then we must prove \( T(n) \leq Cn \log n. \)

By the induction hypothesis,

\[
T(n) \leq 2T(n/2) + c_2n \leq 2C \cdot \frac{n}{2} \log(n/2) + c_2n \\
= Cn(\log(n) - 1) + c_2n = Cn \log(n) + (c_2 - C)n.
\]

This is at most \( Cn \log n \) as long as we choose \( C \geq c_2. \)
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1. \]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \text{ for all } n \geq 2. \)

**Base case** \( n = 2: \) Requires \( C \geq c_1 + c_2. \) ✓

**Inductive step:** Requires \( C \geq c_2. \) ✓
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2n \quad \text{for all } n > 1. \]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \) **for all** \( n \geq 2 \).

**Base case** \( n = 2 \): Requires \( C \geq c_1 + c_2 \). \( \checkmark \)

**Inductive step:** Requires \( C \geq c_2 \). \( \checkmark \)

So we have proved \( T(n) \leq (c_1 + c_2) \log n \) for all \( n \geq 2 \).

This implies \( T(n) = O(n \log n) \), as we were hoping.
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2n \text{ for all } n > 1. \]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \text{ for all } n \geq 2. \)

**Base case** \( n = 2: \) Requires \( C \geq c_1 + c_2. \)

**Inductive step:** Requires \( C \geq c_2. \)

So we have proved \( T(n) \leq (c_1 + c_2) \log n \text{ for all } n \geq 2. \)

This implies \( T(n) = O(n \log n) \), as we were hoping.

But what if \( n \) isn’t a power of 2?
The substitution method

\[ T(1) \leq c_1, \]
\[ T(n) \leq 2T(n/2) + c_2 n \text{ for all } n > 1. \]

**Step 3:** Prove by induction that \( T(n) \leq Cn \log n \text{ for all } n \geq 2. \)

**Base case** \( n = 2: \) Requires \( C \geq c_1 + c_2. \) ✓

**Inductive step:** Requires \( C \geq c_2. \) ✓

So we have proved \( T(n) \leq (c_1 + c_2) \log n \text{ for all } n \geq 2. \)

This implies \( T(n) = O(n \log n), \) as we were hoping.

But what if \( n \) isn’t a power of 2?

For a back-of-the-envelope calculation, we’d just say \( T(n) \leq T(N) \) where \( N \) is the nearest power of two. But sometimes this might be false...
Dealing with floors and ceilings

The “real” recurrence for mergesort is

\[ T(1) \leq c_1, \]
\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \]
for all \( n \geq 2. \)

To deal with floors and ceilings, our guess needs an additive term. Let’s try to find \( C \) and \( a \) such that \( T(n) \leq Cn \log(n) + a \) for all \( n \geq 2. \)
Dealing with floors and ceilings

The “real” recurrence for mergesort is

\[ T(1) \leq c_1, \]
\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \quad \text{for all } n \geq 2. \]

To deal with floors and ceilings, our guess needs an additive term.
Let’s try to find \( C \) and \( a \) such that \( T(n) \leq Cn \log(n) + a \) for all \( n \geq 2. \)

**Base case \( n = 2: \)**

As before, \( T(2) \leq 2T(1) + 2c_2 \leq 2(c_1 + c_2). \)
Dealing with floors and ceilings

The “real” recurrence for mergesort is

\[
T(1) \leq c_1, \\
T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2.
\]

To deal with floors and ceilings, our guess needs an additive term.
Let’s try to find \( C \) and \( a \) such that \( T(n) \leq C n \log(n) + a \) for all \( n \geq 2 \).

**Base case \( n = 2 \):**

As before, \( T(2) \leq 2 T(1) + 2c_2 \leq 2(c_1 + c_2) \).
Also, we have \( C \cdot 2 \log(2) + a = 2C + a \).
The “real” recurrence for mergesort is

\[
T(1) \leq c_1, \\
T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2.
\]

To deal with floors and ceilings, our guess needs an additive term. Let’s try to find \( C \) and \( a \) such that \( T(n) \leq Cn \log(n) + a \) for all \( n \geq 2 \).

**Base case \( n = 2 \):**

As before, \( T(2) \leq 2T(1) + 2c_2 \leq 2(c_1 + c_2) \).

Also, we have \( C \cdot 2 \log(2) + a = 2C + a \).

So the base case works whenever \( 2C + a \geq 2(c_1 + c_2) \). \( \checkmark \)
Dealing with floors and ceilings

\[ T(1) \leq c_1, \]

\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \] 

for all \( n \geq 2 \).

**Goal:** Prove by induction that for all \( n \geq 2 \), \( T(n) \leq Cn \log(n) + a \).

**Base case** \( n = 2 \): Requires \( 2C + a \geq 2(c_1 + c_2) \). √
Dealing with floors and ceilings

\[ T(1) \leq c_1, \]
\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2. \]

**Goal:** Prove by induction that for all \( n \geq 2, \) \( T(n) \leq Cn \log(n) + a. \)

**Base case \( n = 2: \)** Requires \( 2C + a \geq 2(c_1 + c_2). \)

**Inductive step:** Suppose that for all \( 2 \leq n' < n, \) \( T(n') \leq Cn' \log n' + a. \)

Then we must prove \( T(n) \leq Cn \log n + a. \)
Dealing with floors and ceilings

\begin{align*}
T(1) & \leq c_1, \\
T(n) & \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2.
\end{align*}

**Goal:** Prove by induction that for all $n \geq 2$, $T(n) \leq Cn \log(n) + a$.

**Base case $n = 2$:** Requires $2C + a \geq 2(c_1 + c_2)$. \checkmark

**Inductive step:** Suppose that for all $2 \leq n' < n$, $T(n') \leq Cn' \log n' + a$. Then we must prove $T(n) \leq Cn \log n + a$. We have

\begin{align*}
T(n) & \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \\
& \leq C \left( \left\lfloor \frac{n}{2} \right\rfloor \log(\lfloor n/2 \rfloor) + \left\lceil \frac{n}{2} \right\rceil \log(\lceil n/2 \rceil) \right) + 2a + c_2 n.
\end{align*}
Dealing with floors and ceilings

\[ T(1) \leq c_1, \]
\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2. \]

**Goal:** Prove by induction that for all \( n \geq 2, T(n) \leq Cn \log(n) + a. \)

**Base case** \( n = 2: \) Requires \( 2C + a \geq 2(c_1 + c_2). \) \( \checkmark \)

**Inductive step:** Suppose that for all \( 2 \leq n' < n, T(n') \leq Cn' \log n' + a. \) Then we must prove \( T(n) \leq Cn \log n + a. \) We have

\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \]
\[ \leq C \left( \left\lfloor \frac{n}{2} \right\rfloor \log(\lfloor n/2 \rfloor) + \left\lceil \frac{n}{2} \right\rceil \log(\lceil n/2 \rceil) \right) + 2a + c_2 n. \]

To deal with floors and ceilings, we normally use these bounds:

\( \lfloor x \rfloor \leq x \text{ for all } x \in \mathbb{R}, \quad \lceil x \rceil \leq x + 1 \text{ for all } x \in \mathbb{R}, \quad \lfloor x \rfloor \leq 2x \text{ for all } x \geq 1. \)
Dealing with floors and ceilings

\[ T(1) \leq c_1, \]
\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2. \]

**Goal:** Prove by induction that for all \( n \geq 2, \) \( T(n) \leq Cn \log(n) + a. \)

**Base case \( n = 2: \)** Requires \( 2C + a \geq 2(c_1 + c_2). \) \( \checkmark \)

**Inductive step:** Suppose that for all \( 2 \leq n' < n, \) \( T(n') \leq Cn' \log n' + a. \) Then we must prove \( T(n) \leq Cn \log n + a. \) We have

\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \]
\[ \leq C\left( \lfloor \frac{n}{2} \rfloor \log(\lfloor n/2 \rfloor) + \lceil \frac{n}{2} \rceil \log(\lceil n/2 \rceil) \right) + 2a + c_2 n. \]

To deal with floors and ceilings, we normally use these bounds:

\[ \lfloor x \rfloor \leq x \text{ for all } x \in \mathbb{R}, \quad \lceil x \rceil \leq x + 1 \text{ for all } x \in \mathbb{R}, \quad \lceil x \rceil \leq 2x \text{ for all } x \geq 1. \]

Using the “right” bounds in the “right” expressions:

\[ T(n) \leq C\left( \frac{n}{2} \log(n/2) + \left( \frac{n}{2} + 1 \right) \log(n) \right) + 2a + c_2 n. \]
Dealing with floors and ceilings

\[
T(1) \leq c_1,
\]
\[
T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2.
\]

**Goal:** Prove by induction that for all \( n \geq 2 \), \( T(n) \leq Cn \log(n) + a \).

**Base case** \( n = 2 \): Requires \( 2C + a \geq 2(c_1 + c_2) \). \( \checkmark \)

**Inductive step:** Suppose that for all \( 2 \leq n' < n \), \( T(n') \leq Cn' \log n' + a \). Then we must prove \( T(n) \leq Cn \log n + a \). We showed

\[
T(n) \leq C \left( \frac{n}{2} \log(n/2) + \left( \frac{n}{2} + 1 \right) \log(n) \right) + 2a + c_2 n.
\]
Dealing with floors and ceilings

\[ T(1) \leq c_1, \]
\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2. \]

**Goal:** Prove by induction that for all \( n \geq 2 \), \( T(n) \leq Cn \log(n) + a \).

**Base case** \( n = 2 \): Requires \( 2C + a \geq 2(c_1 + c_2) \). √

**Inductive step:** Suppose that for all \( 2 \leq n' < n \), \( T(n') \leq Cn' \log n' + a \).

Then we must prove \( T(n) \leq Cn \log n + a \). We showed

\[ T(n) \leq C \left( \frac{n}{2} \log(n/2) + \left( \frac{n}{2} + 1 \right) \log(n) \right) + 2a + c_2 n. \]

We also bound \( \log(n/2) \leq \log(n) \) to make the algebra a bit easier.
Dealing with floors and ceilings

\[ T(1) \leq c_1, \]
\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2. \]

**Goal:** Prove by induction that for all \( n \geq 2 \), \( T(n) \leq C n \log(n) + a \).

**Base case** \( n = 2 \): Requires \( 2C + a \geq 2(c_1 + c_2) \). \( \checkmark \)

**Inductive step:** Suppose that for all \( 2 \leq n' < n \), \( T(n') \leq C n' \log n' + a \).

Then we must prove \( T(n) \leq C n \log n + a \). We showed

\[ T(n) \leq C \left( \frac{n}{2} \log(n) + \left( \frac{n}{2} + 1 \right) \log(n) \right) + 2a + c_2 n. \]

We also bound \( \log(n/2) \leq \log(n) \) to make the algebra a bit easier.
Dealing with floors and ceilings

\[ T(1) \leq c_1, \]
\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \]  for all \( n \geq 2. \)

**Goal:** Prove by induction that for all \( n \geq 2, \) \( T(n) \leq Cn \log(n) + a. \)

**Base case \( n = 2: \)** Requires \( 2C + a \geq 2(c_1 + c_2). \)  

**Inductive step:** Suppose that for all \( 2 \leq n' < n, \) \( T(n') \leq Cn' \log n' + a. \)  

Then we must prove \( T(n) \leq Cn \log n + a. \) We showed

\[ T(n) \leq C \left( \frac{n}{2} \log(n) + \left( \frac{n}{2} + 1 \right) \log(n) \right) + 2a + c_2 n. \]

We also bound \( \log(n/2) \leq \log(n) \) to make the algebra a bit easier.  

Then rearranging gives:

\[ T(n) \leq Cn \log(n) + \log(n) + 2a + c_2 n \]

This is at most \( Cn \log(n) \) as long as we take \( a \leq -(\log(n) + c_2 n)/2. \)
Dealing with floors and ceilings

\[ T(1) \leq c_1, \]
\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \quad \text{for all } n \geq 2. \]

**Goal:** Prove by induction that for all \( n \geq 2, T(n) \leq Cn \log(n) + a. \)

**Base case** \( n = 2: \) Requires \( 2C + a \geq 2(c_1 + c_2). \) ✓

**Inductive step:** Requires \( a \leq -(\log(n) + c_2 n)/2. \) ✓

So all that’s left is to pick \( C \) and \( a \) that work.
Dealing with floors and ceilings

\[
T(1) \leq c_1, \\
T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2.
\]

**Goal:** Prove by induction that for all \( n \geq 2, \ T(n) \leq Cn \log(n) + a. \)

**Base case** \( n = 2: \) Requires \( 2C + a \geq 2(c_1 + c_2). \) ✓

**Inductive step:** Requires \( a \leq -(\log(n) + c_2 n)/2. \) ✓

So all that’s left is to pick \( C \) and \( a \) that work.

If we take \( a(n) = -(\log(n) + c_2 n)/2, \) then the inductive step works and \( a(2) = -\frac{1}{2} - c_2. \)
Dealing with floors and ceilings

\[ T(1) \leq c_1, \]
\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2. \]

**Goal:** Prove by induction that for all \( n \geq 2, \ T(n) \leq C n \log(n) + a. \)

**Base case** \( n = 2: \) Requires \( 2C + a \geq 2(c_1 + c_2). \) ✓

**Inductive step:** Requires \( a \leq -(\log(n) + c_2 n)/2. \) ✓

So all that’s left is to pick \( C \) and \( a \) that work.

If we take \( a(n) = -(\log(n) + c_2 n)/2, \) then the inductive step works and \( a(2) = -\frac{1}{2} - c_2. \)

So to make the base case work, we take

\[ C = c_1 + c_2 - \frac{a}{2} = c_1 + \frac{3}{2} c_2 + \frac{1}{4} > 0. \]

(Note we do need \( C > 0 \) here!)
Dealing with floors and ceilings

\[ T(1) \leq c_1, \]
\[ T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2. \]

**We proved:** Let \( C = c_1 + \frac{3}{2} c_2 + \frac{1}{4} \), and let \( a(n) = -\frac{1}{2} (c_2 n + \log(n)) \). Then \( T(n) \leq C n \log(n) + a(n) \text{ for all } n \geq 2. \) □
Dealing with floors and ceilings

\[ T(1) \leq c_1, \]
\[ T(n) \leq T([n/2]) + T(\lceil n/2 \rceil) + c_2 n \text{ for all } n \geq 2. \]

**We proved:** Let \( C = c_1 + \frac{3}{2} c_2 + \frac{1}{4}, \) and let \( a(n) = -\frac{1}{2}(c_2 n + \log(n)). \)

Then \( T(n) \leq C n \log(n) + a(n) \text{ for all } n \geq 2. \)

In particular, this implies \( T(n) = O(n \log n) \) as before. Phew!

Note we proved something **stronger** than \( T(n) \leq C n \log(n) \) for all \( n \geq 2. \)

And yet, if we’d tried the proof with \( a(n) = 0, \) it wouldn’t have worked!

It’s counterintuitive, but if you’re having trouble with an induction, strengthening your inductive hypothesis can be very helpful.
Next time: More examples!
(Lecture to be given online...)