Lectures 13/14: Solving Recurrences
COMS10007 - Algorithms

Dr. Christian Konrad

16.03.2020 and 18.03.2020
Algorithmic Design Principle: Divide-and-conquer
Algorithmic Design Principle: Divide-and-conquer

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
Algorithmic Design Principle: Divide-and-conquer

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem

2. **Conquer** the subproblems by solving them recursively (if subproblems have constant size, solve them *directly*)
Algorithmic Design Principle: Divide-and-conquer

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem

2. **Conquer** the subproblems by solving them recursively (if subproblems have constant size, solve them *directly*)

3. **Combine** the solutions to the subproblems into the solution for the original problem

Examples: Quicksort, mergesort, maximum subarray algorithm, binary search, Fast-Peak-Finding, ...
Algorithmic Design Principle: Divide-and-conquer

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem

2. **Conquer** the subproblems by solving them recursively (if subproblems have constant size, solve them *directly*)

3. **Combine** the solutions to the subproblems into the solution for the original problem

Examples
Algorithmic Design Principle: Divide-and-conquer

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem

2. **Conquer** the subproblems by solving them recursively (if subproblems have constant size, solve them *directly*)

3. **Combine** the solutions to the subproblems into the solution for the original problem

**Examples**

Quicksort,
Algorithmic Design Principle: Divide-and-conquer

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem

2. **Conquer** the subproblems by solving them recursively (if subproblems have constant size, solve them *directly*)

3. **Combine** the solutions to the subproblems into the solution for the original problem

**Examples**
Quicksort, mergesort,
Algorithmic Design Principle: Divide-and-conquer

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem
2. **Conquer** the subproblems by solving them recursively (if subproblems have constant size, solve them *directly*)
3. **Combine** the solutions to the subproblems into the solution for the original problem

**Examples**
Quicksort, mergesort, maximum subarray algorithm,
**Algorithmic Design Principle: Divide-and-conquer**

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
2. **Conquer** the subproblems by solving them recursively (if subproblems have constant size, solve them *directly*).
3. **Combine** the solutions to the subproblems into the solution for the original problem.

**Examples**

Quicksort, mergesort, maximum subarray algorithm, binary search,
Algorithmic Design Principle: Divide-and-conquer

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem

2. **Conquer** the subproblems by solving them recursively (if subproblems have constant size, solve them *directly*)

3. **Combine** the solutions to the subproblems into the solution for the original problem

Examples
Quicksort, mergesort, maximum subarray algorithm, binary search, Fast-Peak-Finding, ...
Example: Merge sort

Recall: Merge Sort

Runtime:

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

```
12  9  7  2  3  8  15  7
  2  7  9  12
  3  7  8  15
```
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 $A$

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

$T(1) = O(1)$

$T(n) = 2T(n/2) + O(n)$
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 $A$

![Diagram showing the process of merge sort]

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

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

\[
T(1) = O(1)
\]
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 showing the divide and conquer process]

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

- $T(1) = O(1)$
- $T(n) = 2T(n/2) + O(n)$
How to solve Recurrences?

Recurrences

Divide-and-Conquer algorithms naturally lead to recurrences. How can we solve them? Often only interested in asymptotic upper bounds.

Methods for solving recurrences:
- **Substitution method**: guess solution, verify, induction.
- **Recursion-tree method** (previously seen for merge sort and maximum subarray problem) may have plenty of awkward details, provides good guess that can be verified with substitution method.
- **Master theorem**: very powerful, cannot always be applied.

Dr. Christian Konrad
Lectures 13/14: Solving Recurrences
Recurrences

- Divide-and-Conquer algorithms naturally lead to recurrences
Recurrences

- Divide-and-Conquer algorithms naturally lead to recurrences
- How can we *solve* them?
How to solve Recurrences?

**Recurrences**
- Divide-and-Conquer algorithms naturally lead to recurrences
- How can we *solve* them? Often only interested in asymptotic upper bounds
How to solve Recurrences?

**Recurrences**
- Divide-and-Conquer algorithms naturally lead to recurrences
- How can we *solve* them? Often only interested in asymptotic upper bounds

**Methods for solving recurrences**
How to solve Recurrences?

Recurrences

- Divide-and-Conquer algorithms naturally lead to recurrences
- How can we solve them? Often only interested in asymptotic upper bounds

Methods for solving recurrences

- Substitution method
How to solve Recurrences?

**Recurrences**
- Divide-and-Conquer algorithms naturally lead to recurrences
- How can we *solve* them? Often only interested in asymptotic upper bounds

**Methods for solving recurrences**
- Substitution method
  - guess solution, verify, induction
How to solve Recurrences?

Recurrences
- Divide-and-Conquer algorithms naturally lead to recurrences
- How can we *solve* them? Often only interested in asymptotic upper bounds

Methods for solving recurrences
- Substitution method
  - guess solution, verify, induction
- Recursion-tree method (previously seen for merge sort and maximum subarray problem)
Recurrences

- Divide-and-Conquer algorithms naturally lead to recurrences
- How can we solve them? Often only interested in asymptotic upper bounds

Methods for solving recurrences

- Substitution method
  - guess solution, verify, induction
- Recursion-tree method (previously seen for merge sort and maximum subarray problem)
  - may have plenty of awkward details, provides good guess that can be verified with substitution method
How to solve Recurrences?

Recurrences

- Divide-and-Conquer algorithms naturally lead to recurrences
- How can we solve them? Often only interested in asymptotic upper bounds

Methods for solving recurrences

- Substitution method
guess solution, verify, induction
- Recursion-tree method (previously seen for merge sort and maximum subarray problem)
may have plenty of awkward details, provides good guess that can be verified with substitution method
- Master theorem
How to solve Recurrences?

Recurrences
- Divide-and-Conquer algorithms naturally lead to recurrences
- How can we solve them? Often only interested in asymptotic upper bounds

Methods for solving recurrences
- Substitution method
guess solution, verify, induction
- Recursion-tree method (previously seen for merge sort and maximum subarray problem)
  may have plenty of awkward details, provides good guess that can be verified with substitution method
- Master theorem
  very powerful, cannot always be applied
The Substitution Method

1. Guess the form of the solution

2. Use mathematical induction to find the constants and show that the solution works

3. Method provides an upper bound on the recurrence

Example (suppose $n$ is always a power of two)

Step 1. Guess good upper bound

$T(n) \leq Cn \log n$
The Substitution Method

1. Guess the form of the solution
The Substitution Method

1. Guess the form of the solution
2. Use mathematical induction to find the constants and show that the solution works

Example (suppose \( n \) is always a power of two)

Step 1. Guess good upper bound

\[ T(n) \leq Cn \log n \]
The Substitution Method

1. Guess the form of the solution
2. Use mathematical induction to find the constants and show that the solution works
3. Method provides an upper bound on the recurrence
The Substitution Method

1. Guess the form of the solution
2. Use mathematical induction to find the constants and show that the solution works
3. Method provides an upper bound on the recurrence

Example (suppose $n$ is always a power of two)
The Substitution Method

Guess the form of the solution

Use mathematical induction to find the constants and show that the solution works

Method provides an upper bound on the recurrence

Example (suppose \( n \) is always a power of two)

\[
\begin{align*}
T(1) &= O(1) \\
T(n) &= 2T(n/2) + O(n)
\end{align*}
\]
The Substitution Method

1. Guess the form of the solution
2. Use mathematical induction to find the constants and show that the solution works
3. Method provides an upper bound on the recurrence

Example (suppose $n$ is always a power of two)

$$T(1) = c_1$$
$$T(n) = 2T(n/2) + c_2n$$

Eliminate $O$-notation in recurrence
The Substitution Method

1. Guess the form of the solution
2. Use mathematical induction to find the constants and show that the solution works
3. Method provides an upper bound on the recurrence

Example (suppose $n$ is always a power of two)

\[
T(1) = c_1 \\
T(n) = 2T(n/2) + c_2 n
\]

Eliminate $O$-notation in recurrence

Step 1. Guess good upper bound
The Substitution Method

1. Guess the form of the solution
2. Use mathematical induction to find the constants and show that the solution works
3. Method provides an upper bound on the recurrence

Example (suppose $n$ is always a power of two)

$$T(1) = c_1$$
$$T(n) = 2T(n/2) + c_2n$$

Eliminate $O$-notation in recurrence

Step 1. Guess good upper bound

$$T(n) \leq Cn \log n$$
Step 2. Substitute into the Recurrence

Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$.

Corresponds to induction step of a proof by induction.

$T(n) = 2T(n/2) + c_2n \leq 2Cn \log(n/2) + c_2n = Cn(\log(n) - \log(2)) + c_2n = Cn \log n - Cn + c_2n \leq Cn \log n$, if we chose $C \geq c_2$.

✓ Verify the Base Case

$T(1) \leq C \cdot 1 \log(1) = 0 \nRightarrow c_2$
Step 2. Substitute into the Recurrence

Assume that our guess \( T(n) \leq Cn \log n \) is correct for every \( n' < n \).
Step 2. Substitution Method (2)

Step 2. Substitute into the Recurrence

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$
- Corresponds to induction step of a proof by induction
Step 2. Substitute into the Recurrence

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$
- Corresponds to induction step of a proof by induction

$$T(n)$$
The Substitution Method (2)

Step 2. Substitute into the Recurrence

1. Assume that our guess \( T(n) \leq Cn \log n \) is correct for every \( n' < n \)

2. Corresponds to induction step of a proof by induction

\[
T(n) = 2T(n/2) + c_2 n
\]
Step 2. Substitute into the Recurrence

- Assume that our guess \( T(n) \leq Cn \log n \) is correct for every \( n' < n \).
- Corresponds to induction step of a proof by induction.

\[
T(n) = 2T(n/2) + c_2n \leq 2C \frac{n}{2} \log \left( \frac{n}{2} \right) + c_2n
\]
Step 2. Substitute into the Recurrence

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$
- Corresponds to induction step of a proof by induction

\[
T(n) = 2T\left(\frac{n}{2}\right) + c_2 n \leq 2C \frac{n}{2} \log\left(\frac{n}{2}\right) + c_2 n
\]
\[
= Cn \left(\log(n) - \log(2)\right) + c_2 n
\]
Step 2. Substitute into the Recurrence

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$

- Corresponds to induction step of a proof by induction

$$T(n) = 2T(n/2) + c_2 n \leq 2C \frac{n}{2} \log \left(\frac{n}{2}\right) + c_2 n$$

$$= Cn \left( \log(n) - \log(2) \right) + c_2 n$$

$$= Cn \log n - Cn + c_2 n$$
Step 2. Substitute into the Recurrence

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$

- Corresponds to induction step of a proof by induction

\[
T(n) = 2T(n/2) + c_2 n \leq 2C \frac{n}{2} \log \left( \frac{n}{2} \right) + c_2 n \\
= Cn (\log(n) - \log(2)) + c_2 n \leq Cn \log n - Cn + c_2 n \leq Cn \log n ,
\]

if we chose $C \geq c_2$. 

✓ Verify the Base Case

$T(1) \leq C \cdot 1 \log(1) = 0 \not\geq c_2$

The base case is a problem...
Step 2. Substitute into the Recurrence

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$
- Corresponds to induction step of a proof by induction

\[
T(n) = 2T(n/2) + c_2 n \leq 2C \frac{n}{2} \log \left(\frac{n}{2}\right) + c_2 n
\]
\[
= Cn (\log(n) - \log(2)) + c_2 n
\]
\[
= Cn \log n - Cn + c_2 n \leq Cn \log n
\]

if we chose $C \geq c_2$. √
Step 2. **Substitute into the Recurrence**

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$
- Corresponds to induction step of a proof by induction

\[
T(n) = 2T(n/2) + c_2 n \
\leq 2C \frac{n}{2} \log \left(\frac{n}{2}\right) + c_2 n \
= Cn (\log(n) - \log(2)) + c_2 n \
= Cn \log n - Cn + c_2 n \leq Cn \log n ,
\]

if we chose $C \geq c_2$. ✓

**Verify the Base Case**
The Substitution Method (2)

Step 2. Substitute into the Recurrence

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$
- Corresponds to induction step of a proof by induction

\[
T(n) = 2T(n/2) + c_2 n \leq 2C \frac{n}{2} \log\left(\frac{n}{2}\right) + c_2 n \\
= Cn \left(\log(n) - \log(2)\right) + c_2 n \\
= Cn \log n - Cn + c_2 n \leq Cn \log n,
\]

if we chose $C \geq c_2$. ✓

Verify the Base Case

\[
T(1)
\]
The Substitution Method (2)

Step 2. Substitute into the Recurrence

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$
- Corresponds to induction step of a proof by induction

\[
T(n) = 2T(n/2) + c_2 n \leq 2C \frac{n}{2} \log\left(\frac{n}{2}\right) + c_2 n
\]
\[
= Cn (\log(n) - \log(2)) + c_2 n
\]
\[
= Cn \log n - Cn + c_2 n \leq Cn \log n,
\]
if we chose $C \geq c_2$. ✓

Verify the Base Case

\[
T(1) \leq C \cdot 1 \log(1)
\]
Step 2. Substitute into the Recurrence

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$
- Corresponds to induction step of a proof by induction

\[
T(n) = 2T(n/2) + c_2n \leq 2C \frac{n}{2} \log\left(\frac{n}{2}\right) + c_2n
\]
\[
= Cn (\log(n) - \log(2)) + c_2n
\]
\[
= Cn \log n - Cn + c_2n \leq Cn \log n, \quad \text{if we chose } C \geq c_2. \checkmark
\]

Verify the Base Case

\[
T(1) \leq C \cdot 1 \log(1) = 0
\]
Step 2. Substitute into the Recurrence

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$
- Corresponds to induction step of a proof by induction

\[
T(n) = 2T(n/2) + c_2 n \leq 2C \frac{n}{2} \log\left(\frac{n}{2}\right) + c_2 n \\
= Cn (\log(n) - \log(2)) + c_2 n \\
= Cn \log n - Cn + c_2 n \leq Cn \log n ,
\]

if we chose $C \geq c_2$. ✓

Verify the Base Case

\[
T(1) \leq C \cdot 1 \log(1) = 0 \neq c_2
\]
Step 2. **Substitute into the Recurrence**

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$

- Corresponds to induction step of a proof by induction

\[
T(n) = 2T(n/2) + c_2 n \leq 2C \frac{n}{2} \log \left( \frac{n}{2} \right) + c_2 n \\
= Cn \left( \log(n) - \log(2) \right) + c_2 n \\
= Cn \log n - Cn + c_2 n \leq Cn \log n ,
\]

if we chose $C \geq c_2$. ✓

**Verify the Base Case**

\[
T(1) \leq C \cdot 1 \log(1) = 0 \not\geq c_2 \quad \times
\]
Step 2. Substitute into the Recurrence

- Assume that our guess $T(n) \leq Cn \log n$ is correct for every $n' < n$
- Corresponds to induction step of a proof by induction

$$T(n) = 2T(n/2) + c_2 n \leq 2C \frac{n}{2} \log\left(\frac{n}{2}\right) + c_2 n$$
$$= Cn (\log(n) - \log(2)) + c_2 n$$
$$= Cn \log n - Cn + c_2 n \leq Cn \log n,$$

if we chose $C \geq c_2$. ✓

Verify the Base Case

$$T(1) \leq C \cdot 1 \log(1) = 0 \not\geq c_2 \quad \times$$

The base case is a problem...
The Substitution Method (3)

Recall: \( T(1) = c_1 \) and \( T(n) = 2T(n/2) + c_2 n \)
Our guess: \( T(n) \leq Cn \log n \) (induction step holds for \( C \geq c_2 \))
The Substitution Method (3)

Recall: \( T(1) = c_1 \) and \( T(n) = 2T(n/2) + c_2 n \)
Our guess: \( T(n) \leq Cn \log n \) (induction step holds for \( C \geq c_2 \))

Solution:

Choose a different base case!

\[ T(2) = 2T(1) + 2c_2 = 2c_1 + 2c_2 = 2(c_2 + c_1) \]

\[ C_2 \log 2 \]

Hence, for every \( C \geq c_2 + c_1 \), our guess holds for \( n = 2 \):

\[ T(2) \leq C_2 \log 2. \]

Result:
We proved \( T(n) \leq Cn \log n \), for every \( n \geq 2 \), when choosing \( C \geq c_2 + c_1 \).

Observe:
This implies \( T(n) \in O(n \log n) \) (important).

Asymptotic notation allows us to choose arbitrary base case.
The Substitution Method (3)

Recall: $T(1) = c_1$ and $T(n) = 2T(n/2) + c_2 n$

Our guess: $T(n) \leq C n \log n$ (induction step holds for $C \geq c_2$)

Solution: Choose a different base case! $n = 2$
The Substitution Method (3)

Recall: $T(1) = c_1$ and $T(n) = 2T(n/2) + c_2n$

Our guess: $T(n) \leq Cn \log n$ (induction step holds for $C \geq c_2$)

Solution: Choose a different base case! $n = 2$

$$T(2)$$
The Substitution Method (3)

Recall: $T(1) = c_1$ and $T(n) = 2T(n/2) + c_2n$

Our guess: $T(n) \leq C n \log n$ (induction step holds for $C \geq c_2$)

Solution: Choose a different base case! $n = 2$

\[
T(2) = 2T(1) + 2c_2
\]
Recall: \( T(1) = c_1 \) and \( T(n) = 2T(n/2) + c_2 n \)

Our guess: \( T(n) \leq Cn \log n \) (induction step holds for \( C \geq c_2 \))

**Solution:** Choose a different base case! \( n = 2 \)

\[
T(2) = 2T(1) + 2c_2 = 2c_1 + 2c_2
\]
Recall: $T(1) = c_1$ and $T(n) = 2T(n/2) + c_2 n$

Our guess: $T(n) \leq Cn \log n$ (induction step holds for $C \geq c_2$)

Solution: Choose a different base case! $n = 2$

$$T(2) = 2T(1) + 2c_2 = 2c_1 + 2c_2 = 2(c_2 + c_1)$$
The Substitution Method (3)

Recall: \( T(1) = c_1 \) and \( T(n) = 2T(n/2) + c_2 n \)

Our guess: \( T(n) \leq C n \log n \) (induction step holds for \( C \geq c_2 \))

Solution: Choose a different base case! \( n = 2 \)

\[
T(2) = 2 T(1) + 2 c_2 = 2 c_1 + 2 c_2 = 2 (c_2 + c_1) C 2 \log 2
\]
Recall: \( T(1) = c_1 \) and \( T(n) = 2T(n/2) + c_2 n \)

Our guess: \( T(n) \leq Cn \log n \) (induction step holds for \( C \geq c_2 \))

**Solution:** Choose a different base case! \( n = 2 \)

\[
T(2) = 2T(1) + 2c_2 = 2c_1 + 2c_2 = 2(c_2 + c_1)
\]

\( C \log 2 = 2C \)
Recall: \( T(1) = c_1 \) and \( T(n) = 2T(n/2) + c_2n \)

Our guess: \( T(n) \leq Cn \log n \) (induction step holds for \( C \geq c_2 \))

Solution: Choose a different base case! \( n = 2 \)

\[
T(2) = 2T(1) + 2c_2 = 2c_1 + 2c_2 = 2(c_2 + c_1)
\]

\[
C2 \log 2 = 2C
\]

Hence, for every \( C \geq c_2 + c_1 \), our guess holds for \( n = 2 \):

\[
T(2) \leq C2 \log 2.
\]
The Substitution Method (3)

Recall: \( T(1) = c_1 \) and \( T(n) = 2T(n/2) + c_2 n \)

Our guess: \( T(n) \leq Cn \log n \) (induction step holds for \( C \geq c_2 \))

Solution: Choose a different base case! \( n = 2 \)

\[
T(2) = 2T(1) + 2c_2 = 2c_1 + 2c_2 = 2(c_2 + c_1)
\]

\[
C2 \log 2 = 2C
\]

Hence, for every \( C \geq c_2 + c_1 \), our guess holds for \( n = 2 \):

\[ T(2) \leq C2 \log 2. \]

Result
Recall: $T(1) = c_1$ and $T(n) = 2T(n/2) + c_2n$

Our guess: $T(n) \leq Cn \log n$ (induction step holds for $C \geq c_2$)

Solution: Choose a different base case! $n = 2$

\[
\begin{align*}
T(2) &= 2T(1) + 2c_2 = 2c_1 + 2c_2 = 2(c_2 + c_1) \\
C2 \log 2 &= 2C
\end{align*}
\]

Hence, for every $C \geq c_2 + c_1$, our guess holds for $n = 2$:

\[
T(2) \leq C2 \log 2 .
\]

Result

- We proved $T(n) \leq Cn \log n$, for every $n \geq 2$, when choosing $C \geq c_1 + c_2$
The Substitution Method (3)

Recall: \( T(1) = c_1 \) and \( T(n) = 2T(n/2) + c_2 n \)

Our guess: \( T(n) \leq Cn \log n \) (induction step holds for \( C \geq c_2 \))

Solution: Choose a different base case! \( n = 2 \)

\[
T(2) = 2T(1) + 2c_2 = 2c_1 + 2c_2 = 2(c_2 + c_1)
\]

\[
C2 \log 2 = 2C
\]

Hence, for every \( C \geq c_2 + c_1 \), our guess holds for \( n = 2 \):

\[
T(2) \leq C2 \log 2
\]

Result

- We proved \( T(n) \leq Cn \log n \), for every \( n \geq 2 \), when choosing \( C \geq c_1 + c_2 \)
- Observe: This implies \( T(n) \in O(n \log n) \) (important)
The Substitution Method (3)

Recall: \( T(1) = c_1 \) and \( T(n) = 2T(n/2) + c_2 n \)

Our guess: \( T(n) \leq Cn \log n \) (induction step holds for \( C \geq c_2 \))

Solution: Choose a different base case! \( n = 2 \)

\[
T(2) = 2T(1) + 2c_2 = 2c_1 + 2c_2 = 2(c_2 + c_1)
\]
\[
C2 \log 2 = 2C
\]

Hence, for every \( C \geq c_2 + c_1 \), our guess holds for \( n = 2 \):

\[
T(2) \leq C2 \log 2.
\]

Result

- We proved \( T(n) \leq Cn \log n \), for every \( n \geq 2 \), when choosing \( C \geq c_1 + c_2 \)
- Observe: This implies \( T(n) \in O(n \log n) \) (important)

Asymptotic notation allows us to choose arbitrary base case
A Strange Problem

Example:

Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

Step 1: Guess correct solution

\[T(n) \leq f(n) := Cn\]

Step 2: Verify the solution

\[T(n) \leq C \lceil n/2 \rceil + C \lfloor n/2 \rfloor + 1 = Cn + 1 \not\geq f(n)\]

We need a different guess

Let's try:

\[f_1(n) := Cn + 1 \quad \text{and} \quad f_2(n) := Cn - 1\]

\[f_1:\ T(n) \leq C \lceil n/2 \rceil + 1 + C \lfloor n/2 \rfloor + 1 + 1 = Cn + 3 \not\geq f_1(n)\]

\[f_2:\ T(n) \leq C \lceil n/2 \rceil - 1 + C \lfloor n/2 \rfloor - 1 + 1 = Cn - 1 = f_2(n)\]

✓

(holds for every positive \(C\))
Example: Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

Step 1: Guess correct solution

\[T(n) \leq f(n) := Cn\]

Step 2: Verify the solution

\[T(n) \leq C\lceil n/2 \rceil + 1 + C\lfloor n/2 \rfloor + 1 = Cn + 1 \not\geq f(n)\]

We need a different guess

Let's try:

\[f_1(n) := Cn + 1 \quad \text{and} \quad f_2(n) := Cn - 1\]

\[f_1: \quad T(n) \leq C\lceil n/2 \rceil + 1 + C\lfloor n/2 \rfloor + 1 + 1 = Cn + 3 \not\geq f_1(n)\]

\[f_2: \quad T(n) \leq C\lceil n/2 \rceil - 1 + C\lfloor n/2 \rfloor - 1 + 1 = Cn - 1 = f_2(n)\]

(holds for every positive \(C\))
Example: Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]
Example: Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

Step 1: Guess correct solution
A Strange Problem

**Example:** Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

**Step 1:** Guess correct solution \( T(n) \leq f(n) := Cn \)
Example: Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution
Example: Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution

\( T(n) \)
Example: Give an upper bound on the recurrence

\[
\begin{align*}
T(1) &= 1 \\
T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\end{align*}
\]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution

\[
T(n) \leq C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1
\]
Example: Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution

\[
T(n) \leq C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1 = Cn + 1
\]
Example: Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution

\[
T(n) \leq C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1 = Cn + 1 \n\]

\( X \)
A Strange Problem

Example: Give an upper bound on the recurrence

\[ T(1) = 1 \]
\[ T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution

\[ T(n) \leq C \lceil n/2 \rceil + C \lfloor n/2 \rfloor + 1 = Cn + 1 \not\leq f(n) \]

- We need a different guess
**Example:** Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

**Step 1:** Guess correct solution \( T(n) \leq f(n) := Cn \)

**Step 2:** Verify the solution

\[
T(n) \leq C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1 = Cn + 1 \not\leq f(n) \times
\]

- We need a different guess
- Let’s try: \( f_1(n) := Cn + 1 \) and \( f_2(n) := Cn - 1 \)
A Strange Problem

**Example:** Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

**Step 1:** Guess correct solution \( T(n) \leq f(n) := Cn \)

**Step 2:** Verify the solution

\[
T(n) \leq C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1 = Cn + 1 \not\leq f(n) \]

- We need a different guess
- Let’s try: \( f_1(n) := Cn + 1 \) and \( f_2(n) := Cn - 1 \)

\( f_1 : T(n) \)
Example: Give an upper bound on the recurrence

\[ T(1) = 1 \]
\[ T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution

\[ T(n) \leq C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1 = Cn + 1 \n\neq f(n) \]

- We need a different guess
- Let’s try: \( f_1(n) := Cn + 1 \) and \( f_2(n) := Cn - 1 \)

\[ f_1: T(n) \leq C\lceil n/2 \rceil + 1 + C\lfloor n/2 \rfloor + 1 + 1 \]
Example: Give an upper bound on the recurrence

\[ T(1) = 1 \]
\[ T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution

\[ T(n) \leq C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1 = Cn + 1 \not\leq f(n) \]

- We need a different guess
- Let’s try: \( f_1(n) := Cn + 1 \) and \( f_2(n) := Cn - 1 \)

\[ f_1 : T(n) \leq C\lceil n/2 \rceil + 1 + C\lfloor n/2 \rfloor + 1 + 1 = Cn + 3 \]
A Strange Problem

Example: Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution

\[
T(n) \leq C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1 = Cn + 1 \not\leq f(n) \times
\]

- We need a different guess
- Let’s try: \( f_1(n) := Cn + 1 \) and \( f_2(n) := Cn - 1 \)

\[
f_1 : T(n) \leq C\lceil n/2 \rceil + 1 + C\lfloor n/2 \rfloor + 1 + 1 = Cn + 3 \not\leq f_1(n)
\]
Example: Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution

\[
T(n) \leq C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1 = Cn + 1 \nleq f(n) \qquad \text{X}
\]

- We need a different guess
- Let’s try: \( f_1(n) := Cn + 1 \) and \( f_2(n) := Cn - 1 \)

\[
f_1 : T(n) \leq C\lceil n/2 \rceil + 1 + C\lfloor n/2 \rfloor + 1 + 1 = Cn + 3 \nleq f_1(n) \qquad \text{X}
\]
Example: Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution

\[
T(n) \leq C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1 = Cn + 1 \n\n\n\text{✓ (holds for every positive } C) \]

- We need a different guess
- Let’s try: \( f_1(n) := Cn + 1 \) and \( f_2(n) := Cn - 1 \)

\[
f_1 : T(n) \leq C\lceil n/2 \rceil + 1 + C\lfloor n/2 \rfloor + 1 + 1 = Cn + 3 \n\n\text{✓}
\]

\[
f_2 : T(n)
\]
A Strange Problem

Example: Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution

\[
T(n) \leq C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1 = Cn + 1 \not\leq f(n)
\]

- We need a different guess
- Let’s try: \( f_1(n) := Cn + 1 \) and \( f_2(n) := Cn - 1 \)

\[
f_1 : T(n) \leq C\lceil n/2 \rceil + 1 + C\lfloor n/2 \rfloor + 1 + 1 = Cn + 3 \not\leq f_1(n)
\]

\[
f_2 : T(n) \leq C\lceil n/2 \rceil - 1 + C\lfloor n/2 \rfloor - 1 + 1
\]
Example: Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1
\]

Step 1: Guess correct solution \( T(n) \leq f(n) := Cn \)

Step 2: Verify the solution

\[
T(n) \leq C\lfloor n/2 \rfloor + C\lceil n/2 \rceil + 1 = Cn + 1 \not\leq f(n) \times
\]

- We need a different guess
- Let’s try: \( f_1(n) := Cn + 1 \) and \( f_2(n) := Cn − 1 \)

\[
f_1 : T(n) \leq C\lfloor n/2 \rfloor + 1 + C\lceil n/2 \rceil + 1 + 1 = Cn + 3 \not\leq f_1(n) \times \\
f_2 : T(n) \leq C\lfloor n/2 \rfloor − 1 + C\lceil n/2 \rceil − 1 + 1 = Cn − 1
\]
A Strange Problem

**Example:** Give an upper bound on the recurrence

\[
T(1) = 1 \\
T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1
\]

**Step 1: Guess correct solution** \( T(n) \leq f(n) := Cn \)

**Step 2: Verify the solution**

\[
T(n) \leq C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1 = Cn + 1 \not\leq f(n) \times
\]

- We need a different guess
- Let’s try: \( f_1(n) := Cn + 1 \) and \( f_2(n) := Cn − 1 \)

\[
f_1 : T(n) \leq C\lceil n/2 \rceil + 1 + C\lfloor n/2 \rfloor + 1 + 1 = Cn + 3 \not\leq f_1(n) \times
\]
\[
f_2 : T(n) \leq C\lceil n/2 \rceil − 1 + C\lfloor n/2 \rfloor − 1 + 1 = Cn − 1 = f_2(n)
\]
A Strange Problem

**Example:** Give an upper bound on the recurrence

\[
\begin{align*}
T(1) &= 1 \\
T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1
\end{align*}
\]

**Step 1:** Guess correct solution \( T(n) \leq f(n) := Cn \)

**Step 2:** Verify the solution

\[
T(n) \leq C\lfloor n/2 \rfloor + C\lceil n/2 \rceil + 1 = Cn + 1 \not\leq f(n) \times
\]

- We need a different guess
- Let’s try: \( f_1(n) := Cn + 1 \) and \( f_2(n) := Cn - 1 \)

\[
\begin{align*}
f_1 : T(n) &\leq C\lfloor n/2 \rfloor + 1 + C\lceil n/2 \rceil + 1 + 1 = Cn + 3 \not\leq f_1(n) \times \\
f_2 : T(n) &\leq C\lfloor n/2 \rfloor - 1 + C\lceil n/2 \rceil - 1 + 1 = Cn - 1 = f_2(n) \checkmark
\end{align*}
\]

(holds for every positive \( C \))
Verify Base Case for \( f_2 \)
Verify Base Case for $f_2$

- We have: $T(1) = 1$ and $f_2(1) = C - 1 \geq T(1)$ for $C \geq 2$
Verify Base Case for \( f_2 \)

- We have: \( T(1) = 1 \) and \( f_2(1) = C - 1 \geq T(1) \) for \( C \geq 2 \)
- We thus set the constant \( C \) in \( f_2 \) to \( C = 2 \)
A Strange Problem (2)

Verify Base Case for $f_2$

- We have: $T(1) = 1$ and $f_2(1) = C - 1 \geq T(1)$ for $C \geq 2$
- We thus set the constant $C$ in $f_2$ to $C = 2$
- Then $f_2(n) = 2n - 1 \geq T(n)$ for every $n \geq 1$
A Strange Problem (2)

Verify Base Case for $f_2$

- We have: $T(1) = 1$ and $f_2(1) = C - 1 \geq T(1)$ for $C \geq 2$
- We thus set the constant $C$ in $f_2$ to $C = 2$
- Then $f_2(n) = 2n - 1 \geq T(n)$ for every $n \geq 1$
- This implies that $T(n) \in O(n)$
Verify Base Case for $f_2$

- We have: $T(1) = 1$ and $f_2(1) = C - 1 \geq T(1)$ for $C \geq 2$
- We thus set the constant $C$ in $f_2$ to $C = 2$
- Then $f_2(n) = 2n - 1 \geq T(n)$ for every $n \geq 1$
- This implies that $T(n) \in O(n)$

Comments

Dr. Christian Konrad  Lectures 13/14: Solving Recurrences 9 / 15
A Strange Problem (2)

Verify Base Case for $f_2$

- We have: $T(1) = 1$ and $f_2(1) = C - 1 \geq T(1)$ for $C \geq 2$
- We thus set the constant $C$ in $f_2$ to $C = 2$
- Then $f_2(n) = 2n - 1 \geq T(n)$ for every $n \geq 1$
- This implies that $T(n) \in O(n)$

Comments

- Guessing right can be difficult and requires experience
A Strange Problem (2)

Verify Base Case for $f_2$

- We have: $T(1) = 1$ and $f_2(1) = C - 1 \geq T(1)$ for $C \geq 2$
- We thus set the constant $C$ in $f_2$ to $C = 2$
- Then $f_2(n) = 2n - 1 \geq T(n)$ for every $n \geq 1$
- This implies that $T(n) \in O(n)$

Comments

- Guessing right can be difficult and requires experience
- However, recursion tree method can provide a good guess!
Recursion Tree Method

Recursion Tree:

$T(1) = 1$, $T(n) = 2T(\lfloor n/4 \rfloor) + n/2$

$T(64) = 2T(16) + 32 = 2(2T(4) + 8) + 32 = 2(2(2T(1) + 2) + 8) + 32 = 64$
Recursion Tree Method

Recursion Tree:
- Each node represents cost of single subproblem

Example:
\[ T(1) = 1, \quad T(n) = 2T(\left\lfloor \frac{n}{4} \right\rfloor) + \frac{n}{2} \]

\[ T(64) = 2T(16) + 32 = 2(2T(4) + 8) + 32 \]
\[ = 2(2(2T(1) + 2) + 8) + 32 \]
\[ = 64 \]
Recursion Tree Method

Recursion Tree:
- Each node represents cost of single subproblem
- Recursive invocations become children of a node

Example

\[
T(1) = 1, \\
T(n) = 2T(\lfloor n/4 \rfloor) + n/2
\]

\[
T(64) = 2T(16) + 32 = 2(2T(4) + 8) + 32 = 2(2\cdot1 + 2) + 8 + 32 = 64
\]
Recursion Tree Method

Recursion Tree:
- Each node represents cost of single subproblem
- Recursive invocations become children of a node

Example

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

\[ T(64) = 2T(16) + 32 = 2(2T(4) + 8) + 32 = 2(2(2T(1) + 2) + 8) + 32 = 64 \]
Recursion Tree Method

Recursion Tree:
- Each node represents cost of single subproblem
- Recursive invocations become children of a node

Example

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]
Recursion Tree Method

Recursion Tree:
- Each node represents cost of single subproblem
- Recursive invocations become children of a node

Example

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

\[ T(64) \]
Recursion Tree Method

Recursion Tree:
- Each node represents cost of single subproblem
- Recursive invocations become children of a node

Example

\[ T(1) = 1, \quad T(n) = 2T\left(\lfloor n/4 \rfloor\right) + n/2 \]

\[ T(64) = 2T(16) + 32 \]
Recursion Tree Method

Recursion Tree:
- Each node represents cost of single subproblem
- Recursive invocations become children of a node

Example

\[
T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2
\]

\[
T(64) = 2T(16) + 32 = 2(2T(4) + 8) + 32
\]
Recursion Tree Method

Recursion Tree:
- Each node represents cost of single subproblem
- Recursive invocations become children of a node

Example

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

\[
T(64) = 2T(16) + 32 = 2(2T(4) + 8) + 32 \\
= 2(2(2T(1) + 2) + 8) + 32
\]
Recursion Tree Method

Recursion Tree:
- Each node represents cost of single subproblem
- Recursive invocations become children of a node

Example

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

\[
T(64) = 2T(16) + 32 = 2(2T(4) + 8) + 32
\]
\[
= 2(2(2T(1) + 2) + 8) + 32
\]
\[
= 2(2(2 \cdot 1 + 2) + 8) + 32
\]
Recursion Tree Method

Recursion Tree:
- Each node represents cost of single subproblem
- Recursive invocations become children of a node

Example

\[ T(1) = 1, \quad T(n) = 2T\left(\lfloor n/4 \rfloor \right) + n/2 \]

\[
T(64) = 2T(16) + 32 = 2(2T(4) + 8) + 32 \\
= 2(2(2T(1) + 2) + 8) + 32 \\
= 2(2(2 \cdot 1 + 2) + 8) + 32 = 64
\]
Example

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

Recursion Tree for \( n = 64 \):

Sum of values assigned to nodes equals \( T(64) \).

Dr. Christian Konrad
Lectures 13/14: Solving Recurrences
Example

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + \frac{n}{2} \]

cost of subproblem
Example

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + \frac{n}{2} \]

Recursion Tree for \( n = 64 \):
$T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + \frac{n}{2}$

**Recursion Tree for $n = 64$:**

![Recursion Tree](attachment:image.png)

The cost of subproblem is indicated at each node.

**Sum of values assigned to nodes equals $T(64)$:**

The sum of the values assigned to the nodes of the recursion tree for $n = 64$ equals the total cost of the subproblems, which is $T(64)$. This represents the total cost of solving the recurrence relation for $n = 64$.
Example

\[
T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + \frac{n}{2}
\]

Recursion Tree for \( n = 64 \):

Sum of values assigned to nodes equals \( T(64) \)
Obtaining a Good Guess for Solution

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]
Obtaining a Good Guess for Solution

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

Draw Recursion Tree for general \( n \) (Observe: we ignore \( \lfloor . \rfloor \))
Obtaining a Good Guess for Solution

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

**Draw Recursion Tree for general \( n \) (Observe: we ignore \( \lfloor . \rfloor \))**
Obtaining a Good Guess for Solution

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

Draw Recursion Tree for general \( n \) (Observe: we ignore \( \lfloor . \rfloor \))
$T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2$

**Draw Recursion Tree for general $n$ (Observe: we ignore $\lfloor . \rfloor$)**

![Recursion Tree Image]

**Sum of Nodes in Level $i$:** $\frac{n}{2^i}$ (except the last level)
Obtaining a Good Guess for Solution (2)

Number of Levels: $\ell$

We have

$$n^{\ell - 1} \approx \ell = \log_4(n) + 1$$

Cost on last Level:

$$= \text{number of nodes on last level} \approx 2 \log_4(n)$$

$$= 2 \log(n) \log(4)$$

$$= 2 \log(n) / 2$$

$$= n^{1/2}.$$  

Our Guess:

$$\left( \log_4(n) \sum_{i=1}^{n/2^i} \right) + \sqrt{n} = n \cdot O(1) + \sqrt{n} = O(n).$$

Use substitution method to prove that guess is correct!
Number of Levels: $\ell$

- We have $\frac{n}{4^{\ell-1}} \approx 1$
Number of Levels: $\ell$

- We have $\frac{n}{4^{\ell-1}} \approx 1$
- $\ell = \log_4(n) + 1$
Number of Levels: $\ell$

- We have $\frac{n}{4^{\ell-1}} \approx 1$
- $\ell = \log_4(n) + 1$

Cost on last Level: = number of nodes on last level
Number of Levels: $\ell$

- We have $\frac{n}{4^{\ell-1}} \approx 1$
- $\ell = \log_4(n) + 1$

Cost on last Level: $= \text{number of nodes on last level}$

$\approx 2^{\log_4(n)}$
Obtaining a Good Guess for Solution (2)

**Number of Levels: \( \ell \)**
- We have \( \frac{n}{4^{\ell-1}} \approx 1 \)
- \( \ell = \log_4(n) + 1 \)

**Cost on last Level:** = number of nodes on last level

\[
\approx 2^{\log_4(n)} = 2^{\frac{\log n}{\log 4}}
\]
Number of Levels: $\ell$

- We have $\frac{n}{4^{\ell-1}} \approx 1$
- $\ell = \log_4(n) + 1$

Cost on last Level: number of nodes on last level

$$\approx 2^{\log_4(n)} = 2^{\frac{\log n}{\log 4}} = 2^{\log(n)/2}$$
Obtaining a Good Guess for Solution (2)

**Number of Levels:** \( \ell \)
- We have \( \frac{n}{4^{\ell-1}} \approx 1 \)
- \( \ell = \log_4(n) + 1 \)

**Cost on last Level:** = number of nodes on last level

\[
\approx 2^{\log_4(n)} = 2^{\frac{\log n}{\log 4}} = 2^{\log(n)/2} = n^{\frac{1}{2}}
\]
Obtaining a Good Guess for Solution (2)

**Number of Levels:** \( \ell \)
- We have \( \frac{n}{4^{\ell-1}} \approx 1 \)
- \( \ell = \log_4(n) + 1 \)

**Cost on last Level:** = number of nodes on last level
\[ \approx 2^{\log_4(n)} = 2^{\frac{\log n}{\log 4}} = 2^{\log(n)/2} = n^{\frac{1}{2}} = \sqrt{n}. \]

**Our Guess:**
Number of Levels: \( \ell \)
- We have \( \frac{n}{4^{\ell-1}} \approx 1 \)
- \( \ell = \log_4(n) + 1 \)

Cost on last Level: = number of nodes on last level
\[
\approx 2^{\log_4(n)} = 2^{\frac{\log n}{\log 4}} = 2^{\log(n)/2} = n^{\frac{1}{2}} = \sqrt{n}.
\]

Our Guess:
\[
\left( \sum_{i=1}^{\log_4(n)} \frac{n}{2^i} \right) + \sqrt{n}
\]
Obtaining a Good Guess for Solution (2)

**Number of Levels: \( \ell \)**
- We have \( \frac{n}{4^{\ell-1}} \approx 1 \)
- \( \ell = \log_4(n) + 1 \)

**Cost on last Level:** = number of nodes on last level
\[
\approx 2^{\log_4(n)} = 2^{\frac{\log n}{\log 4}} = 2^{\log(n)/2} = \frac{n^{\frac{1}{2}}}{2} = \sqrt{n}.
\]

**Our Guess:**
\[
\left( \sum_{i=1}^{\log_4(n)} \frac{n}{2^i} \right) + \sqrt{n} = \left( n \cdot \sum_{i=1}^{\log_4(n)} \frac{1}{2^i} \right) + \sqrt{n}
\]

Use substitution method to prove that guess is correct!
Obtaining a Good Guess for Solution (2)

Number of Levels: $\ell$

- We have $\frac{n}{4^{\ell-1}} \approx 1$
- $\ell = \log_4(n) + 1$

Cost on last Level: = number of nodes on last level

$\approx 2^{\log_4(n)} = 2^{\frac{\log n}{\log 4}} = 2^{\log(n)/2} = n^{\frac{1}{2}} = \sqrt{n}$.

Our Guess:

$$\left(\sum_{i=1}^{\log_4(n)} \frac{n}{2^i}\right) + \sqrt{n} = \left(n \cdot \sum_{i=1}^{\log_4(n)} \frac{1}{2^i}\right) + \sqrt{n} = n \cdot O(1) + \sqrt{n}$$

Use substitution method to prove that guess is correct!
Obtaining a Good Guess for Solution (2)

Number of Levels: $\ell$

- We have $\frac{n}{4^{\ell-1}} \approx 1$
- $\ell = \log_4(n) + 1$

Cost on last Level: = number of nodes on last level

$\approx 2^{\log_4(n)} = 2^{\frac{\log n}{\log 4}} = 2^{\log(n)/2} = n^{\frac{1}{2}} = \sqrt{n}$.

Our Guess:

$$\left(\sum_{i=1}^{\log_4(n)} \frac{n}{2^i}\right) + \sqrt{n} = \left(n \cdot \sum_{i=1}^{\log_4(n)} \frac{1}{2^i}\right) + \sqrt{n} = n \cdot O(1) + \sqrt{n} = O(n).$$
Obtaining a Good Guess for Solution (2)

**Number of Levels:** \( \ell \)
- We have \( \frac{n}{4^{\ell-1}} \approx 1 \)
- \( \ell = \log_4(n) + 1 \)

**Cost on last Level:** = number of nodes on last level
\[
\approx 2^{\log_4(n)} = 2^{\frac{\log n}{\log 4}} = 2^{\log(n)/2} = n^{\frac{1}{2}} = \sqrt{n}.
\]

**Our Guess:**
\[
\left( \sum_{i=1}^{\log_4(n)} \frac{n}{2^i} \right) + \sqrt{n} = \left( n \cdot \sum_{i=1}^{\log_4(n)} \frac{1}{2^i} \right) + \sqrt{n} = n \cdot O(1) + \sqrt{n} = O(n).
\]

Use substitution method to prove that guess is correct!
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

**Our Guess:** \( T(n) \leq c \cdot n \)
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

Our Guess: \[ T(n) \leq c \cdot n \]

Substitute into the Recurrence:
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

**Our Guess:** \( T(n) \leq c \cdot n \)

**Substitute into the Recurrence:**

\[ T(n) \]
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

**Our Guess:** \( T(n) \leq c \cdot n \)

**Substitute into the Recurrence:**

\[ T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

**Our Guess:** \( T(n) \leq c \cdot n \)

**Substitute into the Recurrence:**

\[ T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \leq 2c\lfloor \frac{n}{4} \rfloor + \frac{n}{2} \]
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

Our Guess: \( T(n) \leq c \cdot n \)

Substitute into the Recurrence:

\[
T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \leq 2c\lfloor \frac{n}{4} \rfloor + \frac{n}{2} \leq n\frac{c + 1}{2} \leq c \cdot n ,
\]
\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

**Our Guess:** \( T(n) \leq c \cdot n \)

**Substitute into the Recurrence:**

\[ T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \leq 2c \lfloor n/4 \rfloor + \frac{n}{2} \leq n \frac{c + 1}{2} \leq c \cdot n , \]

for every \( c \geq 1 \).
\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

**Our Guess:** \( T(n) \leq c \cdot n \)

**Substitute into the Recurrence:**

\[ T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \leq 2c\lfloor n/4 \rfloor + \frac{n}{2} \leq n\frac{c+1}{2} \leq c \cdot n, \]

for every \( c \geq 1. \)

**Verify the Base Case:**
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

Our Guess: \( T(n) \leq c \cdot n \)

Substitute into the Recurrence:

\[ T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \leq 2c\lfloor n/4 \rfloor + \frac{n}{2} \leq n \frac{c + 1}{2} \leq c \cdot n , \]

for every \( c \geq 1 \).

Verify the Base Case: \( T(1) \)
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

**Our Guess:** \( T(n) \leq c \cdot n \)

**Substitute into the Recurrence:**

\[ T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \leq 2c\lfloor \frac{n}{4} \rfloor + \frac{n}{2} \leq n\frac{c + 1}{2} \leq c \cdot n, \]

for every \( c \geq 1. \)

**Verify the Base Case:** \( T(1) = 1 \leq c \cdot 1 \)
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T\left(\lceil n/4 \rceil \right) + n/2 \]

**Our Guess:** \( T(n) \leq c \cdot n \)

**Substitute into the Recurrence:**

\[ T(n) = 2T\left(\lceil n/4 \rceil \right) + n/2 \leq 2c\left\lfloor \frac{n}{4} \right\rfloor + \frac{n}{2} \leq n\frac{c + 1}{2} \leq c \cdot n, \]

for every \( c \geq 1 \).

**Verify the Base Case:** \( T(1) = 1 \leq c \cdot 1 = c \) for every \( c \geq 1 \).

Summary: We proved \( T(n) \leq n \), for every \( n \geq 1 \)

Hence \( T(n) \in O(n) \).
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

Our Guess: \( T(n) \leq c \cdot n \)

Substitute into the Recurrence:

\[ T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \leq 2c\lfloor \frac{n}{4} \rfloor + \frac{n}{2} \leq n\frac{c+1}{2} \leq c \cdot n, \]

for every \( c \geq 1 \).

Verify the Base Case: \( T(1) = 1 \leq c \cdot 1 = c \) for every \( c \geq 1 \).

Summary:
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

Our Guess: \( T(n) \leq c \cdot n \)

Substitute into the Recurrence:

\[
T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \leq 2c\lfloor \frac{n}{4} \rfloor + \frac{n}{2} \leq n\frac{c+1}{2} \leq c \cdot n ,
\]

for every \( c \geq 1 \).

Verify the Base Case: \( T(1) = 1 \leq c \cdot 1 = c \) for every \( c \geq 1 \).

Summary:
- We proved \( T(n) \leq n \), for every \( n \geq 1 \)
Verification via Substitution Method

\[ T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \]

**Our Guess:** \( T(n) \leq c \cdot n \)

**Substitute into the Recurrence:**

\[ T(n) = 2T(\lfloor n/4 \rfloor) + n/2 \leq 2c\lfloor \frac{n}{4} \rfloor + \frac{n}{2} \leq n\frac{c+1}{2} \leq c \cdot n, \]

for every \( c \geq 1 \).

**Verify the Base Case:** \( T(1) = 1 \leq c \cdot 1 = c \) for every \( c \geq 1 \).

**Summary:**

- We proved \( T(n) \leq n \), for every \( n \geq 1 \)
- Hence \( T(n) \in O(n) \)
Recursion Tree Method
Summary

Recursion Tree Method

- Assign contribution of subproblem to each node
Recursion Tree Method

- Assign contribution of subproblem to each node
- Sum up contributions using tree structure
Recursion Tree Method

- Assign contribution of subproblem to each node
- Sum up contributions using tree structure
- Allows us to be sloppy, since we only aim for a good guess
Summary

**Recursion Tree Method**
- Assign contribution of subproblem to each node
- Sum up contributions using tree structure
- Allows us to be sloppy, since we only aim for a good guess
- Verify guess with substitution method
Summary

Recursion Tree Method

- Assign contribution of subproblem to each node
- Sum up contributions using tree structure
- Allows us to be sloppy, since we only aim for a good guess
- Verify guess with substitution method

Substitution Method
Summary

Recursion Tree Method
- Assign contribution of subproblem to each node
- Sum up contributions using tree structure
- Allows us to be sloppy, since we only aim for a good guess
- Verify guess with subsitution method

Substitution Method
- Guess correct solution
Recursion Tree Method
- Assign contribution of subproblem to each node
- Sum up contributions using tree structure
- Allows us to be sloppy, since we only aim for a good guess
- Verify guess with substitution method

Substitution Method
- Guess correct solution
- Verify guess using mathematical induction
Recursion Tree Method
- Assign contribution of subproblem to each node
- Sum up contributions using tree structure
- Allows us to be sloppy, since we only aim for a good guess
- Verify guess with subsitution method

Substitution Method
- Guess correct solution
- Verify guess using mathematical induction
- Guessing can be difficult