Algorithms and Data Structures

A11. Runtime Analysis: Solving Recurrences

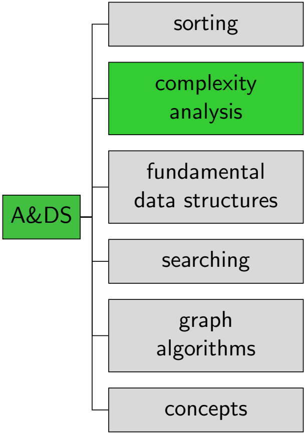
Gabriele Röger

University of Basel

March 14, 2024

Solving Recurrences

Content of the Course



Introduction

In Ch. A10, we derived (algorithmic) recurrences from divide-and-conquer algorithms:

- $T(m) = T(m/2) + \Theta(m)$
  for merge sort.
- $T(n) = 8T(n/2) + \Theta(1)$
  for simple recursive matrix multiplication.
- $T(n) = 7T(n/2) + \Theta(n^2)$
  for Strassen's algorithm for matrix multiplication.

For the asymptotic running time, we want an expression that only depends on the input size (and not recursively on T)!

How can we solve such recurrences?

Approaches

- substitution method
- recursion-tree method
- master theorem
- Akra-Bazzi method
  - Generalization of the master theorem.
  - Not covered in this course.

Substitution Method

Substitution Method

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

Example: Top-down Merge Sort I

Try to Guess the Form of the Solution

Consider $T(m) = 2T(m/2) + \Theta(m)$

Consider $m = 2^k$ with $k \in \mathbb{N}_{>0}$

$$T(m) = 2T(m/2) + c'm$$

Example: Top-down Merge Sort I

Try to Guess the Form of the Solution

Consider $T(m) = 2T(m/2) + \Theta(m)$

Consider $m = 2^k$ with $k \in \mathbb{N}_{>0}$

$$\begin{aligned} T(m) &= 2T(m/2) + c'm \\ &= 2(2T(m/4) + c'(m/2)) + c'm \end{aligned}$$

Example: Top-down Merge Sort I

Try to Guess the Form of the Solution

Consider $T(m) = 2T(m/2) + \Theta(m)$

Consider $m = 2^k$ with $k \in \mathbb{N}_{>0}$

$$\begin{aligned} T(m) &= 2T(m/2) + c'm \\ &= 2(2T(m/4) + c'(m/2)) + c'm \\ &= 2c'm + 4T(m/4) \end{aligned}$$

Example: Top-down Merge Sort I

Try to Guess the Form of the Solution

Consider $T(m) = 2T(m/2) + \Theta(m)$

Consider $m = 2^k$ with $k \in \mathbb{N}_{>0}$

$$\begin{aligned} T(m) &= 2T(m/2) + c'm \\ &= 2(2T(m/4) + c'(m/2)) + c'm \\ &= 2c'm + 4T(m/4) \\ &= 2c'm + 4(2T(m/8) + c'(m/4)) \end{aligned}$$

Example: Top-down Merge Sort I

Try to Guess the Form of the Solution

Consider $T(m) = 2T(m/2) + \Theta(m)$

Consider $m = 2^k$ with $k \in \mathbb{N}_{>0}$

$$
\begin{aligned}
T(m) &= 2T(m/2) + c'm \\
&= 2(2T(m/4) + c'(m/2)) + c'm \\
&= 2c'm + 4T(m/4) \\
&= 2c'm + 4(2T(m/8) + c'(m/4)) \\
&= 3c'm + 8T(m/8)
\end{aligned}
$$

Example: Top-down Merge Sort I

Try to Guess the Form of the Solution

Consider $T(m) = 2T(m/2) + \Theta(m)$

Consider $m = 2^k$ with $k \in \mathbb{N}_{>0}$

$$\begin{aligned}
T(m) &= 2T(m/2) + c'm \\
&= 2(2T(m/4) + c'(m/2)) + c'm \\
&= 2c'm + 4T(m/4) \\
&= 2c'm + 4(2T(m/8) + c'(m/4)) \\
&= 3c'm + 8T(m/8) \\
&= \dots \\
&= kc'm + 2^k c_0 \qquad (\text{use } c_0 \text{ for } T(1)) \\
&= c'm \log_2 m + mc_0 \qquad (k = \log_2 m, 2^k = m)
\end{aligned}$$

Example: Top-down Merge Sort I

Try to Guess the Form of the Solution

Consider $T(m) = 2T(m/2) + \Theta(m)$

Consider $m = 2^k$ with $k \in \mathbb{N}_{>0}$

$$
\begin{aligned}
T(m) &= 2T(m/2) + c'm \\
&= 2(2T(m/4) + c'(m/2)) + c'm \\
&= 2c'm + 4T(m/4) \\
&= 2c'm + 4(2T(m/8) + c'(m/4)) \\
&= 3c'm + 8T(m/8) \\
&= \dots \\
&= kc'm + 2^k c_0 \qquad (\text{use } c_0 \text{ for } T(1)) \\
&= c'm\log_2 m + mc_0 \qquad (k = \log_2 m, 2^k = m) \\
&\leq (c_0 + c')m\log_2 m \qquad (\log_2 m = k \geq 1)
\end{aligned}
$$

Example: Top-down Merge Sort II

Guess Solution and Formulate Hypothesis

Guess: $T(m) \in O(m \log_2 m)$

Example: Top-down Merge Sort II

Guess Solution and Formulate Hypothesis

Guess: $T(m) \in O(m \log_2 m)$

Hypothesis: $T(m) \leq cm \log_2 m$ for all $m \geq m_0$
for some constants $c, m_0 > 0$ (taken care of later).

Example: Top-down Merge Sort II

Guess Solution and Formulate Hypothesis

Guess: $T(m) \in O(m \log_2 m)$

Hypothesis: $T(m) \leq cm \log_2 m$ for all $m \geq m_0$
for some constants $c, m_0 > 0$ (taken care of later).

Let's try the inductive step with this hypothesis.

Example: Top-down Merge Sort II

Verify Guessed Solution with Induction (Inductive Case)

Assume by induction that
$T(m') \leq cm' \log_2(m')$ for all m' with $m_0 \leq m' < m$.

Example: Top-down Merge Sort II

Verify Guessed Solution with Induction (Inductive Case)

Assume by induction that
$T(m') \leq cm' \log_2(m')$ for all m' with $m_0 \leq m' < m$.

Inductive step: $m - 1 \to m$, where $m \geq 2m_0$:

$$\begin{aligned}
T(m) &= 2T(m/2) + c'm \\
&\leq 2c(m/2)\log_2(m/2) + c'm \qquad \text{(induction hypothesis)} \\
&= cm\log_2(m) - cm\log_2 2 + c'm \\
&= cm\log_2(m) - cm + c'm \\
&\leq cm\log_2(m) \qquad \text{if } c > c'
\end{aligned}$$

Example: Top-down Merge Sort II

Verify Guessed Solution with Induction (Inductive Case)

Assume by induction that
$T(m') \leq cm' \log_2(m')$ for all m' with $m_0 \leq m' < m$.

Inductive step: $m - 1 \to m$, where $m \geq 2m_0$:

$$
\begin{aligned}
T(m) &= 2T(m/2) + c'm \\
&\leq 2c(m/2)\log_2(m/2) + c'm \qquad \text{(induction hypothesis)} \\
&= cm\log_2(m) - cm\log_2 2 + c'm \\
&= cm\log_2(m) - cm + c'm \\
&\leq cm\log_2(m) \qquad \text{if } c > c'
\end{aligned}
$$

Inductive steps works if we constrain c to be sufficiently large such that $cm \geq c'm$ for all $m \geq 2m_0$ (with c' hidden constant from $O(m)$).

Example: Top-down Merge Sort II

Verify Guessed Solution with Induction (Base Case)

Show that $T(m) \leq cm\log_2(m)$ for all m with $m_0 \leq m < 2m_0$.

Example: Top-down Merge Sort II

Verify Guessed Solution with Induction (Base Case)

Show that $T(m) \leq cm\log_2(m)$ for all m with $m_0 \leq m < 2m_0$.
Consider $m_0 = 2$.

Example: Top-down Merge Sort II

Verify Guessed Solution with Induction (Base Case)

Show that $T(m) \leq cm\log_2(m)$ for all m with $m_0 \leq m < 2m_0$.
Consider $m_0 = 2$.

- Let $d = \max\{T(2), T(3)\}$
  - $T(2) \leq d \leq d2\log_2 2$
  - $T(3) \leq d \leq d3\log_2 3$

Example: Top-down Merge Sort II

Verify Guessed Solution with Induction (Base Case)

Show that $T(m) \leq cm \log_2(m)$ for all m with $m_0 \leq m < 2m_0$.
Consider $m_0 = 2$.

- Let $d = \max\{T(2), T(3)\}$
  - $T(2) \leq d \leq d2 \log_2 2$
  - $T(3) \leq d \leq d3 \log_2 3$

With $c = \max\{c', d\}$ it holds for all $m \geq 2$ that

$$T(m) \leq cm \log_2(m).$$

We have shown that $T(m) \in O(m \log_2 m)$.

Recursion-tree Method

Recursion-tree Method

In a recursion tree, each node represents the cost of a single subproblem somewhere in the set of recursive function invocations.

Recursion-tree Method

In a recursion tree, each node represents the cost of a single subproblem somewhere in the set of recursive function invocations.

Analyse the cost on each level of the tree and the depth of the tree to get an idea of the overall running time.

Recursion-tree Method

In a recursion tree, each node represents the cost of a single subproblem somewhere in the set of recursive function invocations.

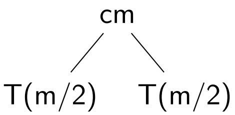
Analyse the cost on each level of the tree and the depth of the tree to get an idea of the overall running time.

Suitable for making a good guess (to be verified by induction).

Example: Top-down Merge Sort I

Consider again $m = 2^k$ with $k \in \mathbb{N}_{>0}$

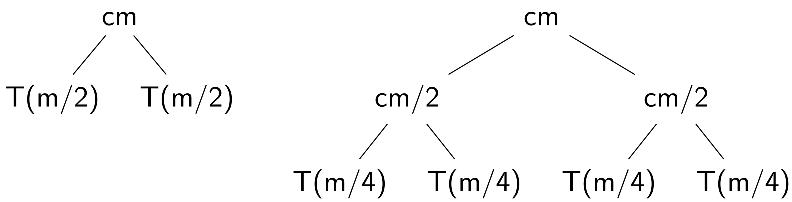
$T(m) = 2T(m/2) + \Theta(m)$



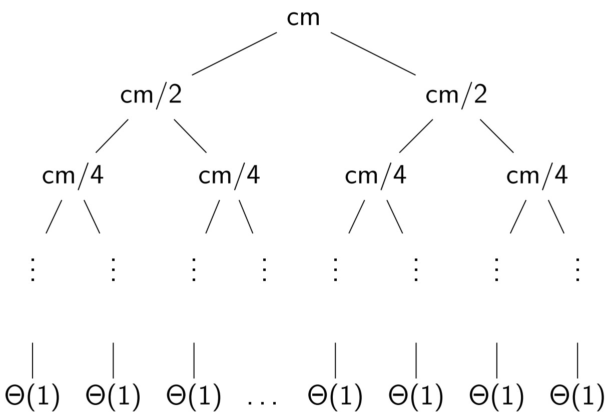
Example: Top-down Merge Sort I

Consider again $m = 2^k$ with $k \in \mathbb{N}_{>0}$

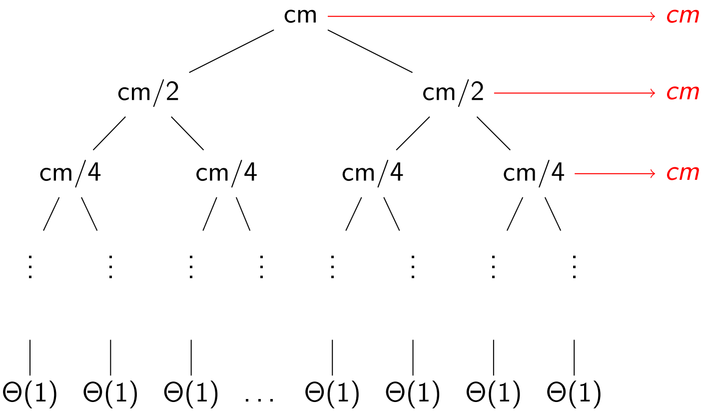
$T(m) = 2T(m/2) + \Theta(m)$



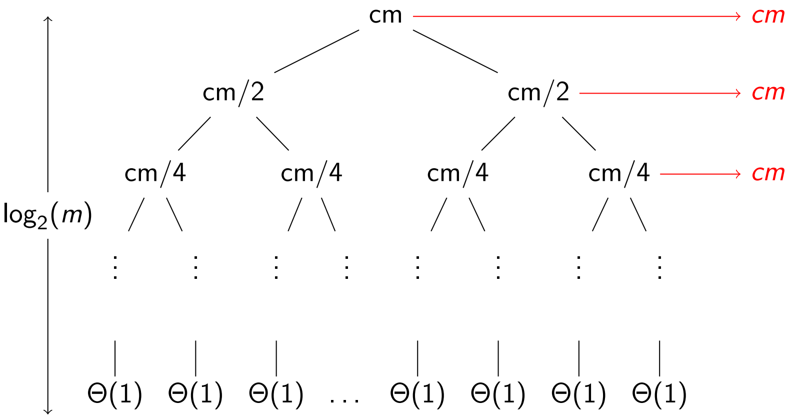
Example: Top-down Merge Sort II



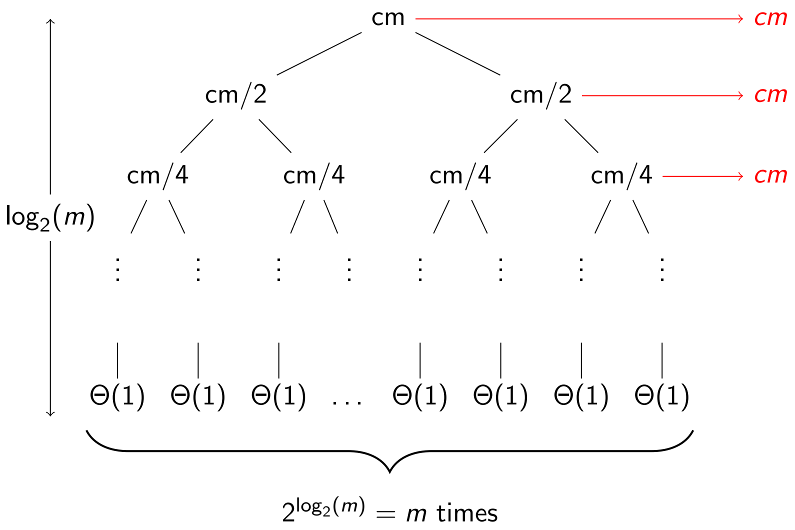
Example: Top-down Merge Sort II



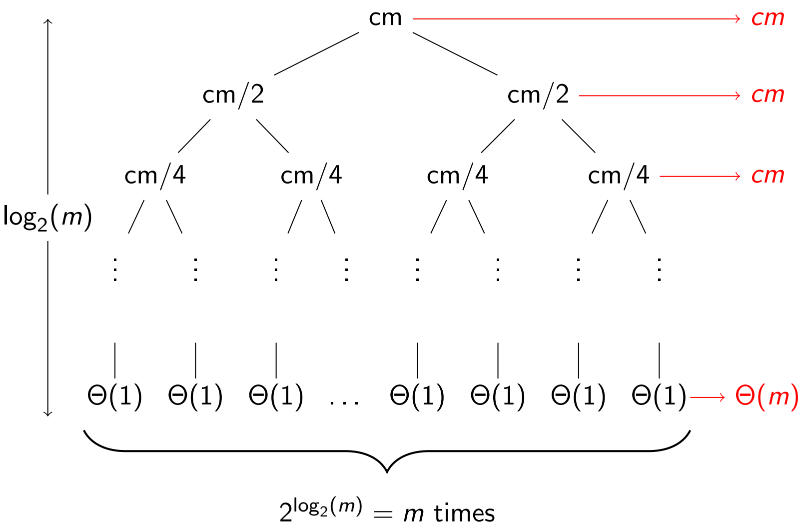
Example: Top-down Merge Sort II



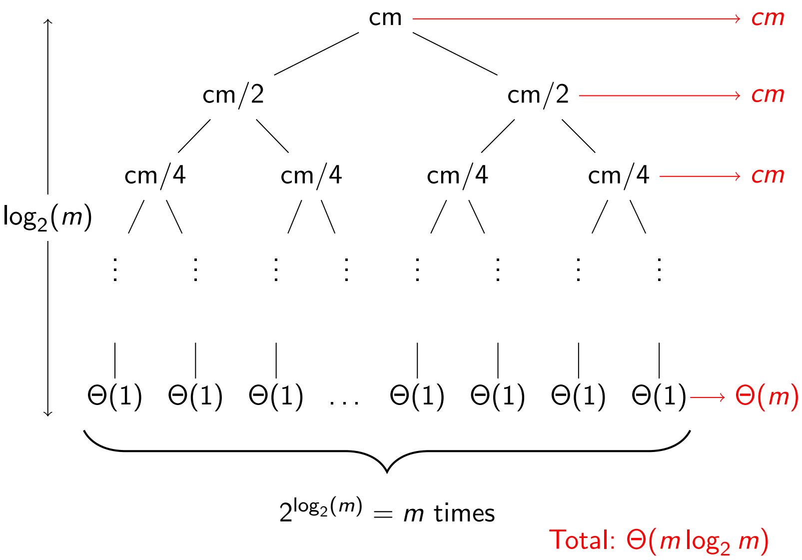
Example: Top-down Merge Sort II



Example: Top-down Merge Sort II

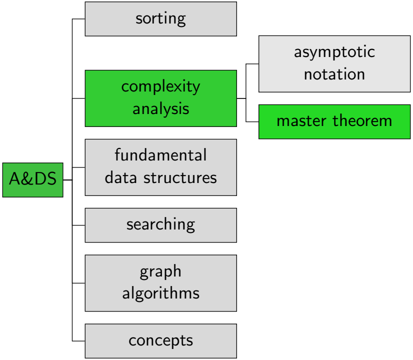


Example: Top-down Merge Sort II



Master Theorem

Content of the Course



Master Recurrences

A common instantiation of the divide-and-conquer algorithm scheme works as follows:

- For inputs of small size $n < C$, solve the problem directly.
- Otherwise:
  1. Construct A smaller inputs of size n/B.
  2. Recursively solve these inputs using the same algorithm.
  3. Compute the result from the recursively computed results.

Master Recurrences

A common instantiation of the divide-and-conquer algorithm scheme works as follows:

- For inputs of small size $n < C$, solve the problem directly.
- Otherwise:
  1. Construct A smaller inputs of size n/B.
  2. Recursively solve these inputs using the same algorithm.
  3. Compute the result from the recursively computed results.

If 1.+3. take time $f(n)$, the overall run-time for $n > C$ can be expressed as $T(n) = A \cdot T(n/B) + f(n)$.

- We call this a master recurrence.
- $f(n)$ is called the driving function.
- We do not care about run-time for $n < C$ because it does not affect asymptotic analysis.

Master Recurrences – Examples

Reminder:

1. Construct A smaller inputs of size n/B.
2. Recursively solve these inputs using the same algorithm.
3. Compute the result from the recursively computed results.

master recurrence: $T(n) = A \cdot T(n/B) + f(n)$

Examples:

- Merge sort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$
- Strassen's algorithm: $A = 7,\ B = 2,\ f(n) = \Theta(n^2)$

Master Theorem

The theorem compares the asymptotic growth of the driving function to the one of the watershed function $n^{\log_B A}$:

Master Theorem

The theorem compares the asymptotic growth of the driving function to the one of the watershed function $n^{\log_B A}$:

Theorem

Let $A \geq 1, B > 1$ be constants and $f(n)$ be a driving function that is defined and nonnegative on all sufficiently large reals. Let T satisfy the master recurrence $T(n) = A \cdot T(n/B) + f(n)$. Then:

- *If $f(n) = O(n^{\log_B A - \varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_B A})$.*
- *If $f(n) = \Theta(n^{\log_B A} \log_2^k n)$ for some $k \geq 0$, then $T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$.*
- *If $f(n) = \Omega(n^{\log_B A + \varepsilon})$ for some $\varepsilon > 0$ and if $Af(n/B) \leq cf(n)$ for some $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.*

Master Theorem: Intuition (Case 1)

$f(n) = O(n^{\log_B A - \varepsilon})$

- Watershed function grows polynomially faster than the driving function.
- Cost per level in recursion tree grows at least geometrically from root to leaves.
- Cost of leaves dominates total cost of inner nodes.

Master Theorem: Intuition (Case 2)

$f(n) = \Theta(n^{\log_B A} \log_2^k n)$

- $\log_2^k n = (\log_2 n)^k$
- Both functions grow at nearly the same asymptotic rates.
- Precisely: driving function only grows faster than the watershed function by a factor of $\log_2^k n$.
- Each level of the tree costs approximately the same.
- With $k = 0$, the second case covers case $f(n) = \Theta(n^{\log_B A})$.

Master Theorem: Intuition (Case 3)

$f(n) = \Omega(n^{\log_B A+\varepsilon})$

- Driving function grows polynomially faster than the watershed function.
- Regularity condition $Af(n/B) \leq cf(n)$ is typically satisfied (no big growth differences of driving function in different areas of the recursion tree).
- Cost per level in recursion tree drops at least geometrically from root to leaves.
- Cost of root dominates cost of other nodes in the recursion tree.

Application: Merge Sort

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A} \log_2^k n) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Merge sort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$

Application: Merge Sort

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A} \log_2^k n) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Merge sort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$
$\rightsquigarrow \log_B A = \log_2 2 = 1$

Application: Merge Sort

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A} \log_2^k n) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Merge sort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$
$\rightsquigarrow \log_B A = \log_2 2 = 1$

- Case 1 $f(n) = O(n^{1-\varepsilon})$ for some $\varepsilon > 0$?

Application: Merge Sort

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A} \log_2^k n) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Merge sort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$
$\rightsquigarrow \log_B A = \log_2 2 = 1$

- Case 1 $f(n) = O(n^{1-\varepsilon})$ for some $\varepsilon > 0$? $\rightsquigarrow$ No.

Application: Merge Sort

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A} \log_2^k n) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Merge sort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$
$\rightsquigarrow \log_B A = \log_2 2 = 1$

- Case 1 $f(n) = O(n^{1-\varepsilon})$ for some $\varepsilon > 0$? $\rightsquigarrow$ No.
- Case 2 $f(n) = \Theta(n^1 \log_2^k n)$ for some k?

Application: Merge Sort

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A} \log_2^k n) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Merge sort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$
$\rightsquigarrow \log_B A = \log_2 2 = 1$

- Case 1 $f(n) = O(n^{1-\varepsilon})$ for some $\varepsilon > 0$? $\rightsquigarrow$ No.
- Case 2 $f(n) = \Theta(n^1 \log_2^k n)$ for some k? $\rightsquigarrow$ Yes with $k = 0$!

Application: Merge Sort

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A} \log_2^k n) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Merge sort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$
$\rightsquigarrow \log_B A = \log_2 2 = 1$

- Case 1 $f(n) = O(n^{1-\varepsilon})$ for some $\varepsilon > 0$? $\rightsquigarrow$ No.
- Case 2 $f(n) = \Theta(n^1 \log_2^k n)$ for some k? $\rightsquigarrow$ Yes with $k = 0$!
  $\rightsquigarrow$ $T(n) = \Theta(n^1 \log_2 n)$

Application: Strassen's Algorithm

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A} \log_2^k n) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Strassen's algorithm: $A = 7,\ B = 2,\ f(n) = \Theta(n^2)$

Application: Strassen's Algorithm

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A} \log_2^k n) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Strassen's algorithm: $A = 7,\ B = 2,\ f(n) = \Theta(n^2)$
$\rightsquigarrow \log_B A = \log_2 7 \approx 2.807\ldots$

Application: Strassen's Algorithm

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A-\varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A} \log_2^k n) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$
- $f(n) = \Omega(n^{\log_B A+\varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Strassen's algorithm: $A = 7,\ B = 2,\ f(n) = \Theta(n^2)$
$\rightsquigarrow \log_B A = \log_2 7 \approx 2.807\ldots$

- Case 1 $f(n) = O(n^{\log_2 7-\varepsilon})$ for some $\varepsilon > 0$?

Application: Strassen's Algorithm

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A-\varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A} \log_2^k n) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$
- $f(n) = \Omega(n^{\log_B A+\varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Strassen's algorithm: $A = 7,\ B = 2,\ f(n) = \Theta(n^2)$
$\rightsquigarrow \log_B A = \log_2 7 \approx 2.807\ldots$

- Case 1 $f(n) = O(n^{\log_2 7-\varepsilon})$ for some $\varepsilon > 0$?
  $\rightsquigarrow$ Yes, for instance with $\varepsilon = 0.8$!

Application: Strassen's Algorithm

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A} \log_2^k n) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2^{k+1} n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Strassen's algorithm: $A = 7,\ B = 2,\ f(n) = \Theta(n^2)$
$\rightsquigarrow \log_B A = \log_2 7 \approx 2.807\ldots$

- Case 1 $f(n) = O(n^{\log_2 7 - \varepsilon})$ for some $\varepsilon > 0$?
  $\rightsquigarrow$ Yes, for instance with $\varepsilon = 0.8$! $\rightsquigarrow$ $T(n) = \Theta(n^{\log_2 7})$

Floors and Ceilings?

Merge Sort Revisited

We used $T(m) = 2T(m/2) + \Theta(m)$ as recurrence for merge sort.

Merge Sort Revisited

We used $T(m) = 2T(m/2) + \Theta(m)$ as recurrence for merge sort.

If $m = 5$, we have one recursive call for 2 and one for 3 elements.

Merge Sort Revisited

We used $T(m) = 2T(m/2) + \Theta(m)$ as recurrence for merge sort.

If $m = 5$, we have one recursive call for 2 and one for 3 elements.

The precise recurrence for the running time is

$$T(m) = T(\lfloor m/2 \rfloor)) + T(\lceil m/2 \rceil)) + \Theta(m).$$

Merge Sort Revisited

We used $T(m) = 2T(m/2) + \Theta(m)$ as recurrence for merge sort.

If $m = 5$, we have one recursive call for 2 and one for 3 elements.

The precise recurrence for the running time is

$$T(m) = T(\lfloor m/2 \rfloor)) + T(\lceil m/2 \rceil)) + \Theta(m).$$

Does this make a difference for the asymptotic growth?

Good News

Ignoring floors and ceilings does not generally affect the order of growth of the solution of a divide-and-conquer recurrence.

Good News

Ignoring floors and ceilings does not generally affect the order of growth of the solution of a divide-and-conquer recurrence.

The master theorem also holds for recurrences

$$T(n) = A' T(\lfloor n/B \rfloor)) + A'' T(\lceil n/B \rceil)) + f(n)$$

for some constant $A', A'' \geq 0$ (set $A := A' + A''$).

Example: Merge Sort (Optional Material)

Inductive Case Revisited

Assume by induction that
$T(m') \leq cm' \log_2(m')$ for all m' with $m_0 \leq m' < m$.

Example: Merge Sort (Optional Material)

Inductive Case Revisited

Assume by induction that
$T(m') \leq cm' \log_2(m')$ for all m' with $m_0 \leq m' < m$.

Inductive step: $m - 1 \to m$, where $m \geq 2m_0$:

$$
\begin{aligned}
T(m) &= T(\lfloor m/2 \rfloor)) + T(\lceil m/2 \rceil)) + c'm \\
&\leq c\lfloor m/2 \rfloor \log_2(\lfloor m/2 \rfloor) + c\lceil m/2 \rceil \log_2(\lceil m/2 \rceil) + c'm \\
&\leq c\lfloor m/2 \rfloor \log_2(\lceil m/2 \rceil) + c\lceil m/2 \rceil \log_2(\lceil m/2 \rceil) + c'm \\
&\leq cm \log_2(\lceil m/2 \rceil) + c'm \qquad (\lfloor m/2 \rfloor + \lceil m/2 \rceil = m) \\
&\leq cm \log_2((m+1)/2) + c'm \qquad (\lceil m/2 \rceil \leq (m+1)/2) \\
&= cm \log_2(m+1) - cm + c'm \\
&\leq cm \log_2 m + cm/m - cm + c'm \quad (\log_2(m+1) \leq \log_2(m) + \frac{1}{m}) \\
&= cm \log_2 m + c - cm + c'm \\
&\leq cm \log_2 m \qquad \text{if } c > 2c' \text{ and } m \geq 2
\end{aligned}
$$

Summary

Summary

- The substitution method is the most general one:
  - Guess the running time (typically by substituting the recursive term a few times).
  - Prove by mathematical induction that the guess is correct.
- The recursion-tree method is good for quickly getting an impression of a running time.
- The master theorem is not always applicable. If it is, it is the quickest way to determine the running time.

- Top-down merge sort has linearithmic running time $\Theta(m \log_2 m)$.