Algorithms & Data Structures

Lecture 04
Recurrences & The Master Method

Giovanni Bacci
giovbacci@cs.aau.dk

Outline

- The substitution method
- The recursion-tree method
- The master method

Intended Learning Goals

KNOWLEDGE

- Mathematical reasoning on concepts such as recursion, induction, concrete and abstract computational complexity
- Data structures, algorithm principles e.g., search trees, hash tables, dynamic programming, divide-and-conquer
- Graphs and graph algorithms e.g., graph exploration, shortest path, strongly connected components.

SKILLS

- Determine abstract complexity for specific algorithms
- Perform complexity and correctness analysis for simple algorithms
- Select and apply appropriate algorithms for standard tasks

COMPETENCES

- Ability to face a non-standard programming assignment
- Develop algorithms and data structures for solving specific tasks
- Analyse developed algorithms

Recall: Divide & Conquer

- **Divide** the problem into a number of subproblems that are smaller instances of the same problem
- **Conquer** the subproblem by solving them recursively. If the subproblem is small (and easy) enough solve it trivially
- **Combine:** the solution of the subproblems into the solution for the original problem

Recurrences go hand in hand with divide and conquer algorithms

Size of trivial subproblem

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \,, \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \,. \end{cases}$$

Number of **recursive** calls to the subproblems

Division of the problem $b > 1$

Running time for the **divide** step

Running time for the **combine** step

Example: Merge Sort

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

- **Divide:** computing the middle of the subarray takes $\Theta(1)$
- **Conquer:** solving recursively two subproblems each of size $n/2$, contributes $2T(n/2)$ to the running time
- **Combine:** the merge takes $\Theta(n)$ on an n-elements subarray.

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n = 1 \\ 2T(n/2) + \Theta(n) & \textbf{if } n > 1 \end{cases}$$

…to be precise $T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$

Example: Insertion Sort

```
INSERTIONSORT(A, p)
if p > 1
    INSERTIONSORT(A, p − 1)
    // Insert A[p] into the sorted sequence A[1 . . p − 1]
    key = A[p]
    i = p − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n = 1 \\ T(n-1) + \Theta(n) & \textbf{if } n > 1 \end{cases}$$

…and many other forms

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n \leq 2 \\ \boxed{T(n-1) + T(n-2)} + \Theta(1) & \textbf{if } n > 2 \end{cases}$$

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n = 1 \\ \boxed{T(2n/3) + T(n/3)} + \Theta(n) & \textbf{if } n > 1 \end{cases}$$

or it may divide the problem into unequal sizes

How to solve recurrences

There is no straightforward solution for recurrences in general, but there are 3 methods which can help

- In the **substitution method**, we guess a bound and use mathematical induction to prove our guess correct

- The **recursion-tree method** converts the recurrence into a tree whose nodes represent the costs at various levels of the recursion. Usually, one uses techniques for bounding summations to solve the recurrence

- The **master method** provides bounds for recurrences of the form $T(n) = aT(n/b) + f(n)$ where $a \geq 1, b > 1$.

Substitution Method

The method comprises two steps:

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

- **Powerful method** which leads to an elegant analysis
- …but **we must have a good guess** for the form of the answer

Example

We want to establish an **upper bound** on the recurrence

$$T(n) = \begin{cases} 1 & \textbf{if } n \leq 1 \\ 2T(\lfloor n/2 \rfloor) + n & \textbf{if } n > 1 \end{cases}$$

- Let us guess that the solution is $T(n) = O(n \lg n)$
- The substitution method requires us to find appropriate constants $c > 0$ and $n_0 > 0$ such that

$$T(n) \leq cn \lg n \qquad \text{for all } n \geq n_0$$

Example

We want to establish an **upper bound** on the recurrence

$$T(n) = \begin{cases} 1 & \textbf{if } n \leq 1 \\ 2T(\lfloor n/2 \rfloor) + n & \textbf{if } n > 1 \end{cases}$$

- Let us guess that the solution is $T(n) = O(n \lg n)$
- The substitution method requires us to constants $c > 0$ and $n_0 > 0$ such th

This gives us a format for an inductive hypothesis!

$$T(n) \leq cn \lg n \qquad \text{for all } n \geq n_0$$

Example: Inductive Step

- We start by assuming that $T(m) \leq cm \lg m$ holds for all $n_0 \leq m < n$ (i.e., inductive hypothesis)
- Substituting into the recurrence yields

$$
\begin{aligned}
T(n) &= 2T(\lfloor n/2 \rfloor) + n && \text{(def } T \text{)} \\
&\leq 2c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor) + n && \text{(} \lfloor n/2 \rfloor < n \text{ and Ind.Hp.)} \\
&\leq cn \lg(n/2) + n && \text{(} c > 0 \text{ and } \lfloor n/2 \rfloor \leq n/2 \text{)} \\
&= cn \lg n - cn \lg 2 + n \\
&= cn \lg n - cn + n \\
&\leq cn \lg n && \text{(assuming } c \geq 1 \text{)}
\end{aligned}
$$

Example: Base Case

- Typically we need to show that the hypothesis holds for the base case of the recurrence, i.e., $T(n) \leq cn \lg n$ for $n \leq 1$.
- Here there is a problem: $T(1) = 1$ but $c \lg 1 = 0$

Example: Base Case

- Typically we need to show that the hypothesis holds for the base case of the recurrence, i.e., $T(n) \leq cn \lg n$ for $n \leq 1$.
- Here there is a problem: $T(1) = 1$ but $c \lg 1 = 0$

How do we proceed now?

Example: Base Case

- Recall that we are performing an asymptotic analysis!
- the property must hold for $n \geq n_0$ and we can choose n_0
- Observe that for $n > 3$ the recurrence $T(n)$ does not depend directly on $T(1)$.
- Thus we can fix $n_0 = 2$ and use as bases for our induction $T(2) = 4$ and $T(3) = 5$ (*)

We complete the proof by choosing some $c \geq 1$ such that

$$T(2) \leq c2\lg 2 \qquad\qquad T(3) \leq c3\lg 3$$

Now one can verify that this holds for any $c \geq 2$

Why $T(2)$ and $T(3)$?

- They can be both derived by $T(1) = 1$:
  - $T(2) = 2T(1) + 2 = 4$
  - $T(3) = 2T(1) + 3 = 5$

- For all $n > 3$, the unfolding of $T(n)$ can be stopped when encountering $T(2)$ or $T(3)$ avoiding to use $T(1)$.

- For this reason they are alternative base cases for proving

$$T(n) \leq cn \lg n \qquad \text{for all } n \geq 2$$

Subtleties with Asymptotic Notation

- Consider $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- If we guess that $T(n) = O(n)$ and try to show that $T(n) \leq cn$ for some $c > 0$
- Substituting our guess in the recurrence yields

$$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \\ &\leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\ &= cn + 1 \end{aligned}$$

Subtleties with Asymptotic Notation

- Consider $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- If we guess that $T(n) = O(n)$ and try to show that $T(n) \leq cn$ for some $c > 0$
- Substituting our guess in the recurrence yields

$$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \\ &\leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\ &= cn + 1 \end{aligned}$$

Does not imply that $T(n) \leq cn$ for any choice of c

Subtleties with Asymptotic Notation

- Consider $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- If we guess that $T(n) = O(n)$ and try to show that $T(n) \leq cn$ for some $c > 0$
- Substituting our guess in the recurrence yields

$$
\begin{aligned}
T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \\
&\leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\
&= cn + 1 \\
&= O(n)
\end{aligned}
$$

Subtleties with Asymptotic Notation

- Consider $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- If we guess that $T(n) = O(n)$ and try to show that $T(n) \leq cn$ for some $c > 0$
- Substituting our guess in the recurrence yields

$$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \\ &\leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\ &= cn + 1 \\ &= O(n) \end{aligned}$$

WRONG!!

We have to prove the **exact form** of the inductive hypothesis.

(Asymptotic notation hides the accumulation of lower order terms)

Subtleties with Asymptotic Notation

- Consider $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- `The guess $T(n) = O(n)$ is correct!
- The inductive hypothesis $T(n) \leq cn$ is not strong enough
- **We can also try to prove** $T(n) \leq cn - d$ **for some** $c, d > 0$
- Substituting our new guess in the recurrence yields

$$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \\ &\leq (c\lfloor n/2 \rfloor - d) + (c\lceil n/2 \rceil - d) + 1 \\ &= cn - 2d + 1 \\ &\leq cn - d \end{aligned}$$

(assuming $d \geq 1$)

Change of Variable

Sometimes a little algebraic manipulation can make an unknown recurrence similar to one you have seen before

Example:

Consider the recurrence $T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n$

- Define $m = \lg n$ (let's not worry about rounding off values)
- Changing variable yields to $T(2^m) = 2T(2^{m/2}) + m$
- We can further rename $S(m) = T(2^m)$
- Producing $S(m) = 2S(m/2) + m$ and we know $S(m) = O(m \lg m)$
- Substituting back we obtain $T(n) = O(\lg n \lg \lg n)$

Making a good guess

- **No general way** to guess the correct solution to recurrences
  - We'll see that **recursion-trees** can help
  - It takes **experience** and, occasionally creativity
  - If a recursion is similar to one seen before, then guessing a similar solution is reasonable

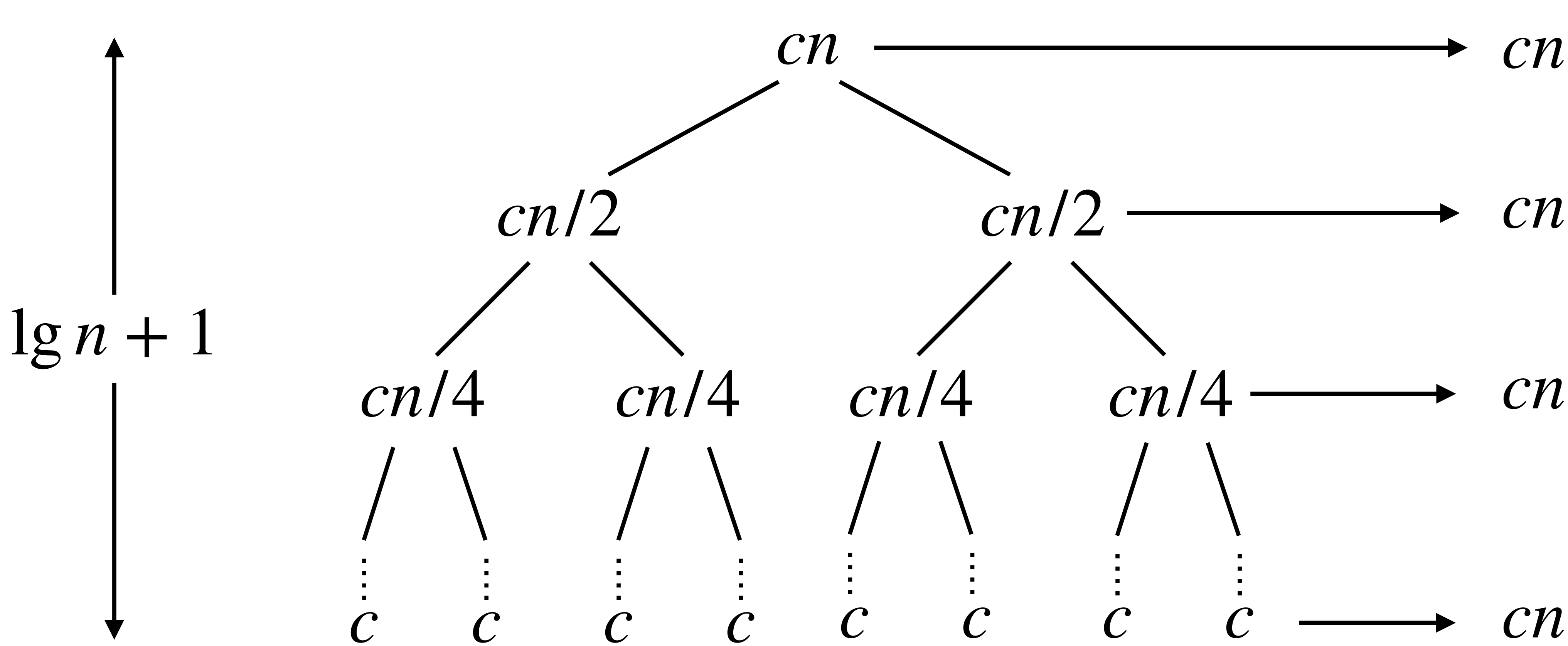
Example:

- Consider $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$
- When n is large, the difference between $\lfloor n/2 \rfloor + 17$ and $\lfloor n/2 \rfloor$ is not that large
- One can prove that the guess $T(n) = O(n \lg n)$ works

Recursion-tree Method

How to construct the tree:

1. Each node represents the cost of a single subproblem in the unravelling of recursive function invocations
2. We sum the costs within each level obtaining peer-level costs
3. We sum all the peer-level costs obtaining the total cost



Recursion-tree Method

- Best **used to generate good guesses**
- we can **tolerate some amount of “sloppiness”** in the development of the guess (e.g., assuming that n is a power of 2)
- The guess will be later (rigorously) verified using the substitution method

Example

We want to provide a good guess for the recurrence

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n = 1 \\ 3T(\lfloor n/4 \rfloor) + \Theta(n^2) & \textbf{if } n > 1 \end{cases}$$

- We focus at finding an upper bound for the solution
- We assume that n is a power of 4
- We create a recursion-tree for $T(n) = 3T(n/4) + cn^2$ having in mind that $c > 0$

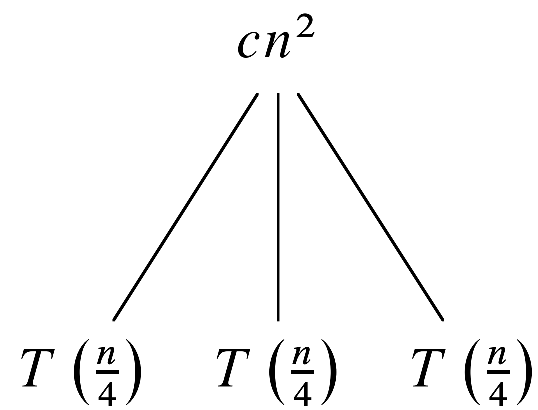
Example

$$T(n) = 3T(n/4) + cn^2$$

$$T(n)$$

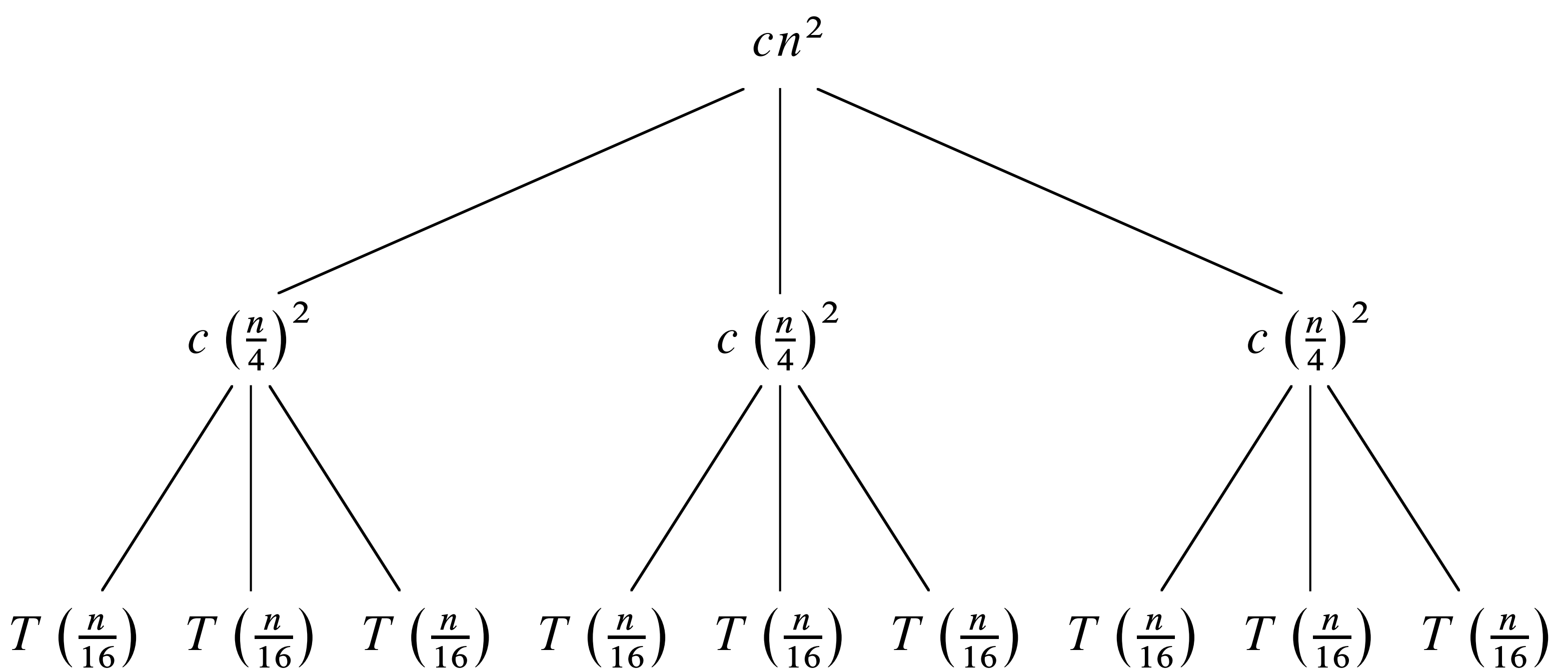
Example

$$T(n) = 3T(n/4) + cn^2$$



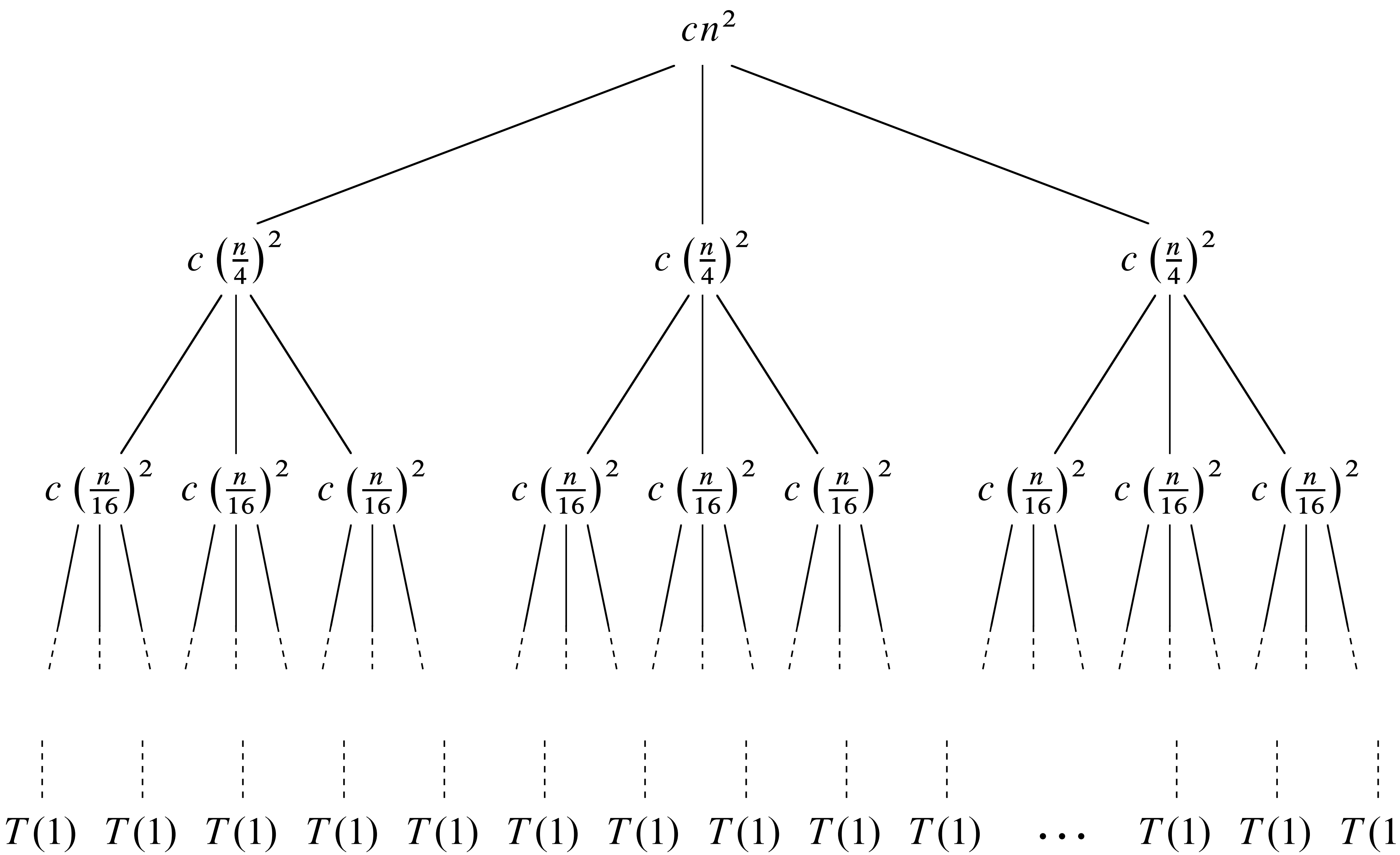
Example

$$T(n) = 3T(n/4) + cn^2$$



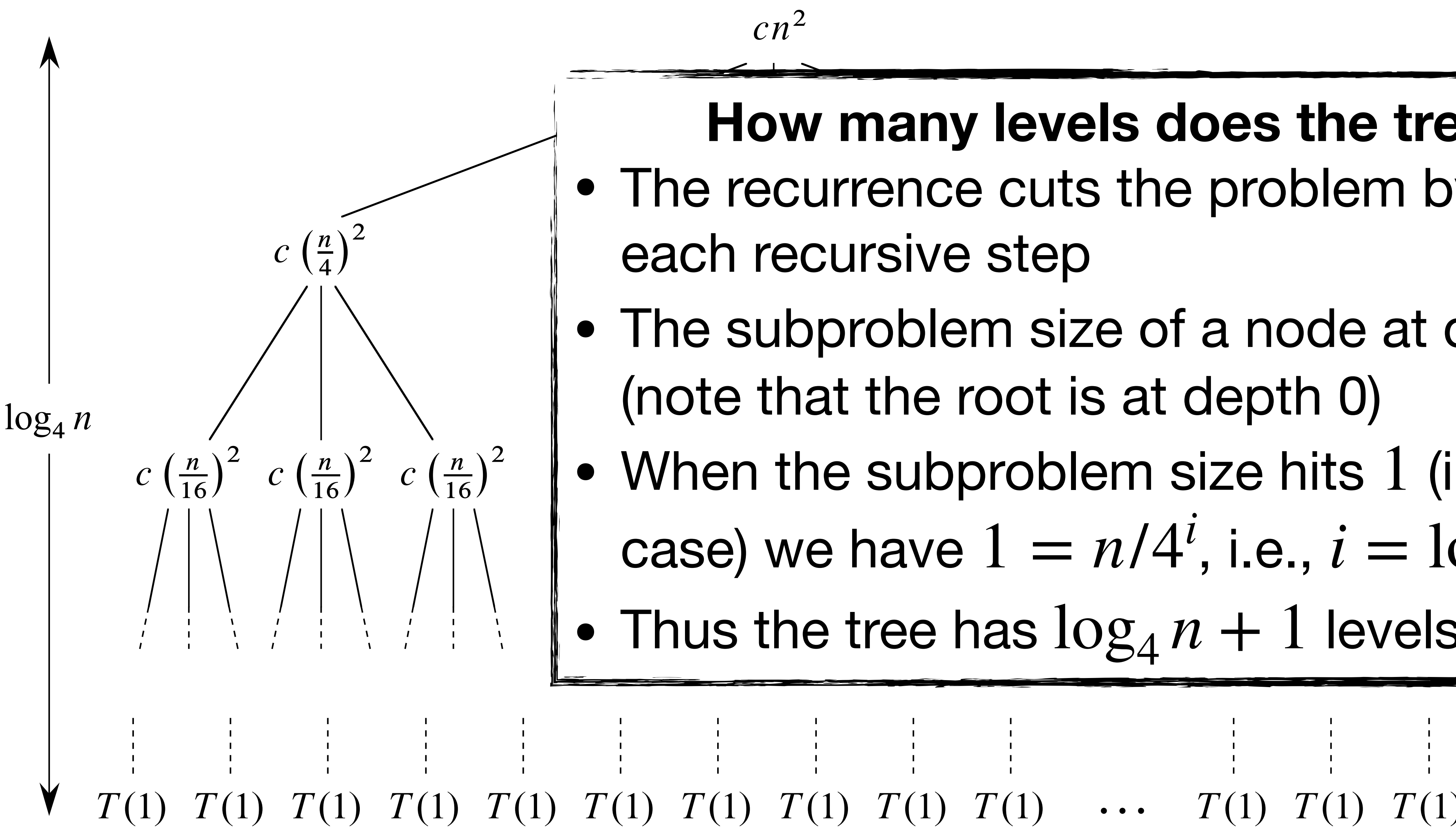
Example

$$T(n) = 3T(n/4) + cn^2$$



Example

$$T(n) = 3T(n/4) + cn^2$$

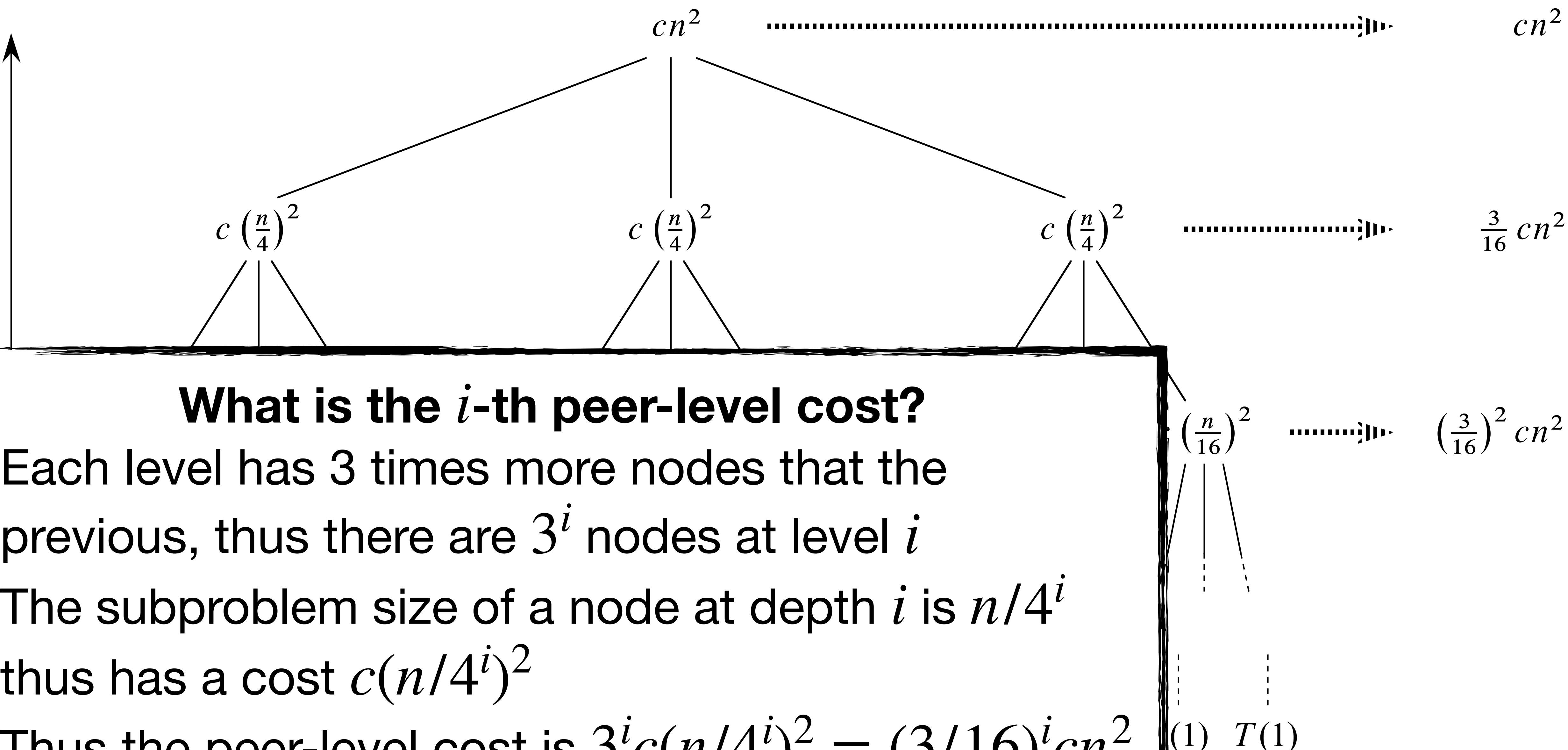


How many levels does the tree have?

- The recurrence cuts the problem by a factor of 4 each recursive step
- The subproblem size of a node at depth i is $n/4^i$ (note that the root is at depth 0)
- When the subproblem size hits 1 (i.e., the base case) we have $1 = n/4^i$, i.e., $i = \log_4 n$
- Thus the tree has $\log_4 n + 1$ levels

Example

$$T(n) = 3T(n/4) + cn^2$$

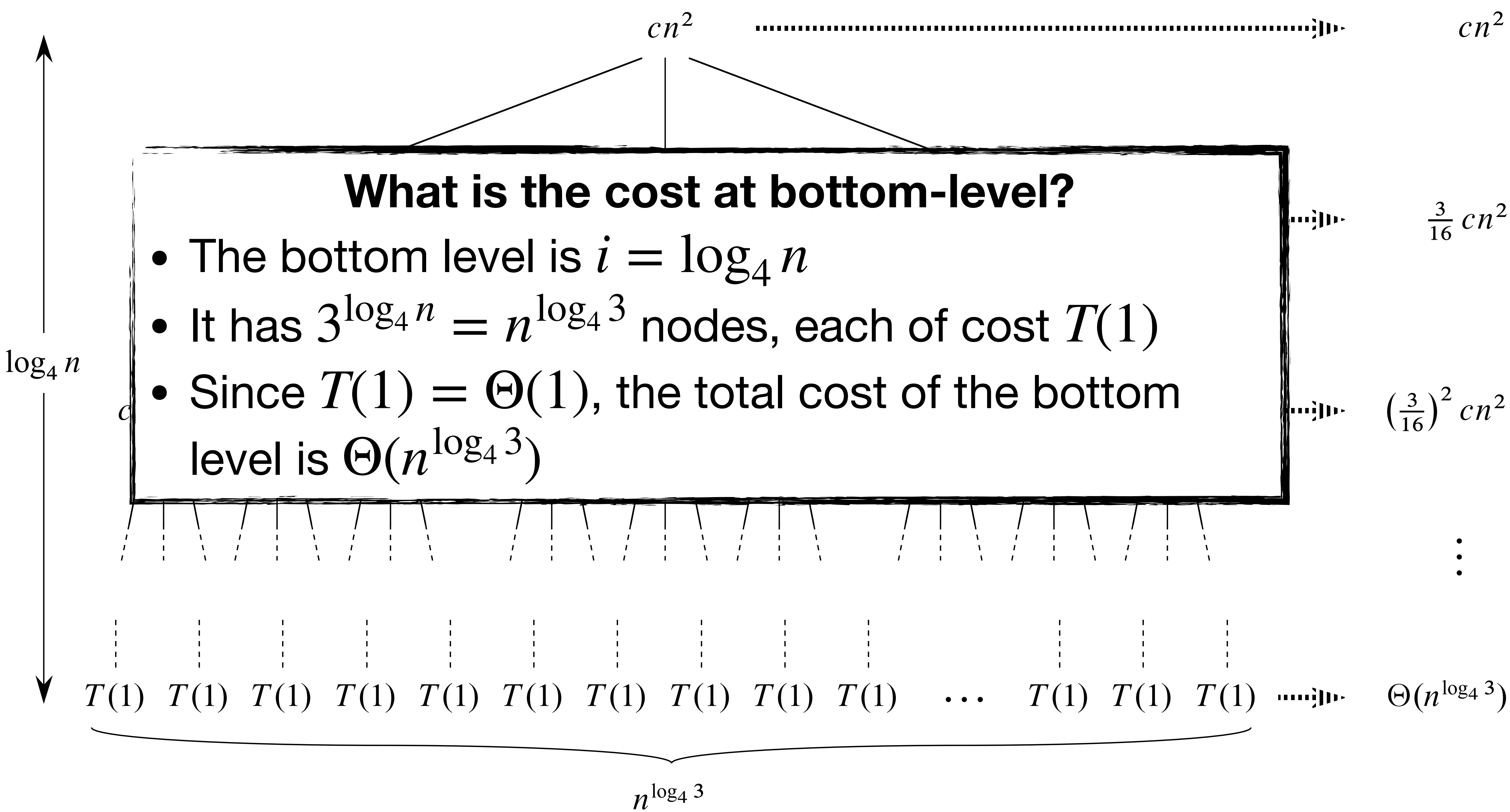


What is the i-th peer-level cost?

- Each level has 3 times more nodes that the previous, thus there are 3^i nodes at level i
- The subproblem size of a node at depth i is $n/4^i$ thus has a cost $c(n/4^i)^2$
- Thus the peer-level cost is $3^i c(n/4^i)^2 = (3/16)^i cn^2$

Example

$T(n) = 3T(n/4) + cn^2$



What is the cost at bottom-level?

- The bottom level is $i = \log_4 n$
- It has $3^{\log_4 n} = n^{\log_4 3}$ nodes, each of cost $T(1)$
- Since $T(1) = \Theta(1)$, the total cost of the bottom level is $\Theta(n^{\log_4 3})$

Example

Finally, we can sum up the costs of each peer-level ($i = 0,1,2,\ldots,\log_4 n - 1$) leading to the following upper bound

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3}) \qquad \text{(recall geometric series (A.6) CLRS)} \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2)
\end{aligned}
$$

Example

Finally, we can sum up the costs of each peer-level ($i = 0,1,2,\ldots,\log_4 n - 1$) leading to the following upper bound

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2)
\end{aligned}
$$

(recall geometric series (A.6) CLRS)

Now we have our guess!

Quiz

Let us verify if our guess is correct.

Use the **substitution method** to prove that $T(n) = O(n^2)$ for the recurrence

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n = 1 \\ 3T(\lfloor n/4 \rfloor) + \Theta(n^2) & \textbf{if } n > 1 \end{cases}$$

Master Method

It provides a “cookbook” method for solving recurrences of the form*

$$T(n) = aT(n/b) + f(n)$$

Classic format of divide & conquer

where $a \geq 1, b > 1$ and $f(n)$ is an asymptotically positive function.

- Simpler than substitution method, because does not require a good guess;
- Does not cover some classic recurrences

(*) Replacing $T(n/b)$ with either $T(\lceil n/b \rceil)$ or $T(\lfloor n/b \rfloor)$ will not affect the asymptotic behaviour of the recurrence (optional see CLRS 4.6.2)

The Master Theorem

Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

A closer look

- The theorem comprises 3 cases:
  1. $f(n)$ is **asymptotically smaller** than $n^{\log_b a}$ by a factor of n^ϵ for some constant $\epsilon > 0$.
  2. $f(n)$ is **asymptotically equal** to $n^{\log_b a}$
  3. $f(n)$ is **asymptotically larger** than $n^{\log_b a}$ by a factor of n^ϵ for some constant $\epsilon > 0$. Moreover, $f(n)$ satisfies the "regularity" condition $af(n/b) \leq cf(n)$ (most of the polynomially bounded functions that we'll encounter do)
- These cases do not cover all the possibilities for $f(n)$. There are gaps between cases 1 and 2 and cases 2 and 3.

HOW To Use the Master Theorem

You can follow these steps:

1. Identify if the recurrence is in the proper format $T(n) = aT(n/b) + f(n)$ (where $a \geq 1$ and $b > 1$)
2. Simplify $n^{\log_b a}$ by plugging in the values of a and b
3. Check if one of the 3 cases holds:
    1. $f(n) = O(n^{\log_b a - \epsilon}) = O(n^{\log_b a}/n^{\epsilon})$ for some $\epsilon > 0$
    2. $f(n) = \Theta(n^{\log_b a})$
    3. $f(n) = \Omega(n^{\log_b a + \epsilon}) = \Omega(n^{\log_b a} \cdot n^{\epsilon})$ for some $\epsilon > 0$. Moreover, check if there exist $n_0 \in \mathbb{N}$ and $0 < c < 1$ such that $af(n/b) \leq cf(n)$ for all $n \geq n_0$.
4. Simplify the corresponding conclusion of the Theorem by plugging in the values of a, b, and $f(n)$

Example 1

Let's try to solve $T(n) = 9T(n/3) + n$

1. $T(n) = aT(n/b) + f(n)$ where $a = 9$, $b = 3$, and $f(n) = n$.
2. Simplify $n^{\log_b a} = n^{\log_3 9} = n^{\log_3 3^2} = n^2$
3. Case 1 applies: $f(n) = n = O(n^{\log_3 9 - \epsilon}) = O(n^{2-\epsilon})$ for $\epsilon = 1$
4. By Theorem 4.1-Case 1 we conclude that $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$

Example 2

Let's try to solve $T(n) = T(2n/3) + 1$

1. $T(n) = aT(n/b) + f(n)$ where $a = 1$, $b = 3/2$, and $f(n) = 1$.

2. Simplify $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

3. Case 2 applies: $f(n) = 1 = \Theta(n^{\log_b a}) = \Theta(1)$

4. By Theorem 4.1-Case 2 we conclude that
   $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n)$

Example 3

Let's try to solve $T(n) = 3T(n/4) + n \lg n$

1. $T(n) = aT(n/b) + f(n)$ where $a = 3$, $b = 4$, and $f(n) = n \lg n$.
2. Simplify $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
3. Case 3 applies:
   - $f(n) = n \lg n = \Omega(n^{\log_4 3 + \epsilon})$ where $\epsilon \cong 0.2$
   - For $n \geq n_0$, we have

$$
\begin{aligned}
af(n/b) &= a(n/b)\lg(n/b) && (f(n) = n \lg n) \\
&= (3/4)n(\lg n - \lg 4) && (a = 3 \text{ and } b = 4) \\
&\leq (3/4)n \lg n && (\text{choose } n_0 \geq 4) \\
&= c f(n) && (c = 3/4)
\end{aligned}
$$

8. By Theorem 4.1-Case 3 we conclude that $T(n) = \Theta(f(n)) = \Theta(n \lg n)$

Example 4

- Let's try to solve $T(n) = 2T(n/2) + n \lg n$
- $T(n) = aT(n/b) + f(n)$ where $a = 2$, $b = 2$, and $f(n) = n \lg n$
- Simplify $n^{\log_b a} = n^{\log_2 2} = n$
- Case 1 and Case 2 clearly don't work. Maybe Case 3…
  - Can we find $\epsilon > 0$ such that $f(n) = n \lg n = \Omega(n^{\log_b a + \epsilon}) = \Omega(n^{1+\epsilon})$?
  - Note that $f(n)/n^{\log_b a} = (n \lg n)/n = \lg n$ is asymptotically smaller than n^ϵ for any $\epsilon > 0$
- In this case, the Master Theorem doesn't help.

Example 4

- Let's try to solve $T(n) = 2T(n/2) + n\lg n$
- $T(n) = aT(n/b) + f(n)$ where $a = 2$, $b = 2$, and $f(n) = n\lg n$
- Simplify $n^{\log_b a} = n^{\log_2 2} = n$
- Case 1 and Case 2 clearly don't work. Maybe Case 3…
  - Can we find $\epsilon > 0$ such that $f(n) = n\lg n = \Omega(n^{\log_b a + \epsilon}) = \Omega(n^{1+\epsilon})$?
  - Note that $f(n)/n^{\log_b a} = (n\lg n)/n = \lg n$ is asymptotically smaller than n^{ϵ} for any $\epsilon > 0$
- In this case, the Master Theorem doesn't help.

Try to solve it with another method

Learned Today

- Analysis Techniques for solving recurrences:
  - The substitution method
  - The recursion-tree method
  - The master method