Algorithms

complexity of recursive algorithms

Jiří Vyskočil, Marko Genyk-Berezovskyj

2010-2014

Recurrences

- A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs. For example

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if } n > 1\,. \end{cases}$$

  Where $T(n)$ is the overall complexity of the algorithm.

  The complexities for different values of n are listed on the right hand side.

- Marginal cases (n < constant) can be neglected, because the complexity of an algorithm in such cases is also constant. Often it happens that rounding the values does not change the results too. The given recurrence can be simplified to:

$$T(n) = 2T(n/2) + \Theta(n)\,,$$

Simplifying recurrences

- We want to obtain a formula with no recurrence.
  - Example: $T(n) = \Theta(\log(n))$

- Methods:
  - **Substitution method**
    - First, guess the solution and then prove its correctness by induction.
  - **Recursion-tree method**
    - Evaluate characteristics of the recursion tree.
  - **Use the „cookbook“ - Master theorem**
    - Some common forms of recurrences are solved in general by the Master theorem, we only have to evaluate the conditions of the theorem.

Substitution method

- Two-steps solution:
  1. **Guess the exact form of the solution.**
     - Typically, one would precompute a set of numerical results for different input n's and derive a formula.
  2. **Prove the correctness of the guess.**
     - Mathematical induction is the tool of choice in most cases.

- The method is very effective provided we can complete the step 1 correctly. On the other hand, there are no general rules how to deal with it.

Substitution method - Example

- Example:

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

- Suppose, we come (somehow) to a guess:

  $T(n) = O(n \log(n))$

- Using the definition of upper bound "O" we want to prove:

  $T(n) \leq cn \log(n)$

  for some suitable $c > 0$.

- State an induction hypothesis (i.e. let the guess be true for $n/2$): $T(n/2) \leq c(n/2) \log(n/2)$

Substitution method - Example

- Substitute the hypothesis into the original recurrence $T(n) = 2T(n/2) + n$ and complete the proof in standard manner as you would do in any other induction proof.

$$
\begin{aligned}
T(n) &\leq 2(c(n/2)\log(n/2)) + n \\
&\leq cn\log(n/2) + n \qquad \text{// due to the hypothesis} \\
&= cn\log(n) - cn\log(2) + n \\
&= cn\log(n) - cn + n \\
&\leq cn\log(n)
\end{aligned}
$$

The last inequality holds for $c \geq 1$.

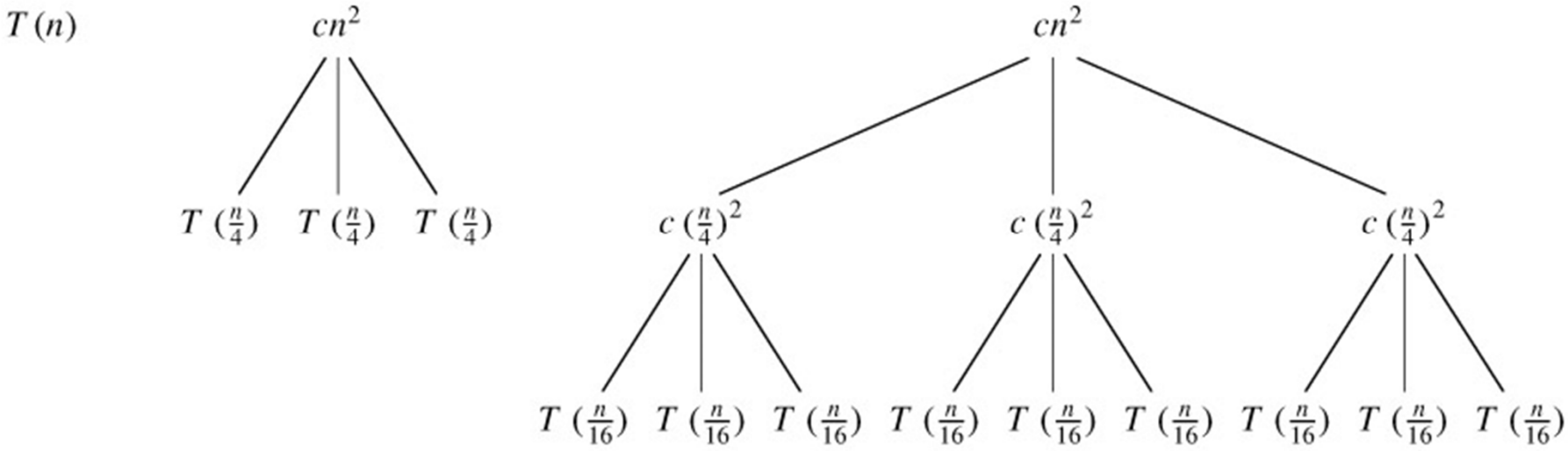
- The induction base case holds trivially: All what is needed is to show that for some values of n_0 and $c > 0$ the base case holds. Choose for example n_0=3 and $c \geq 2$.
- The proof is complete, the substitution method has yielded a result.

Recursion-tree method - Example

- Recurrence:

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

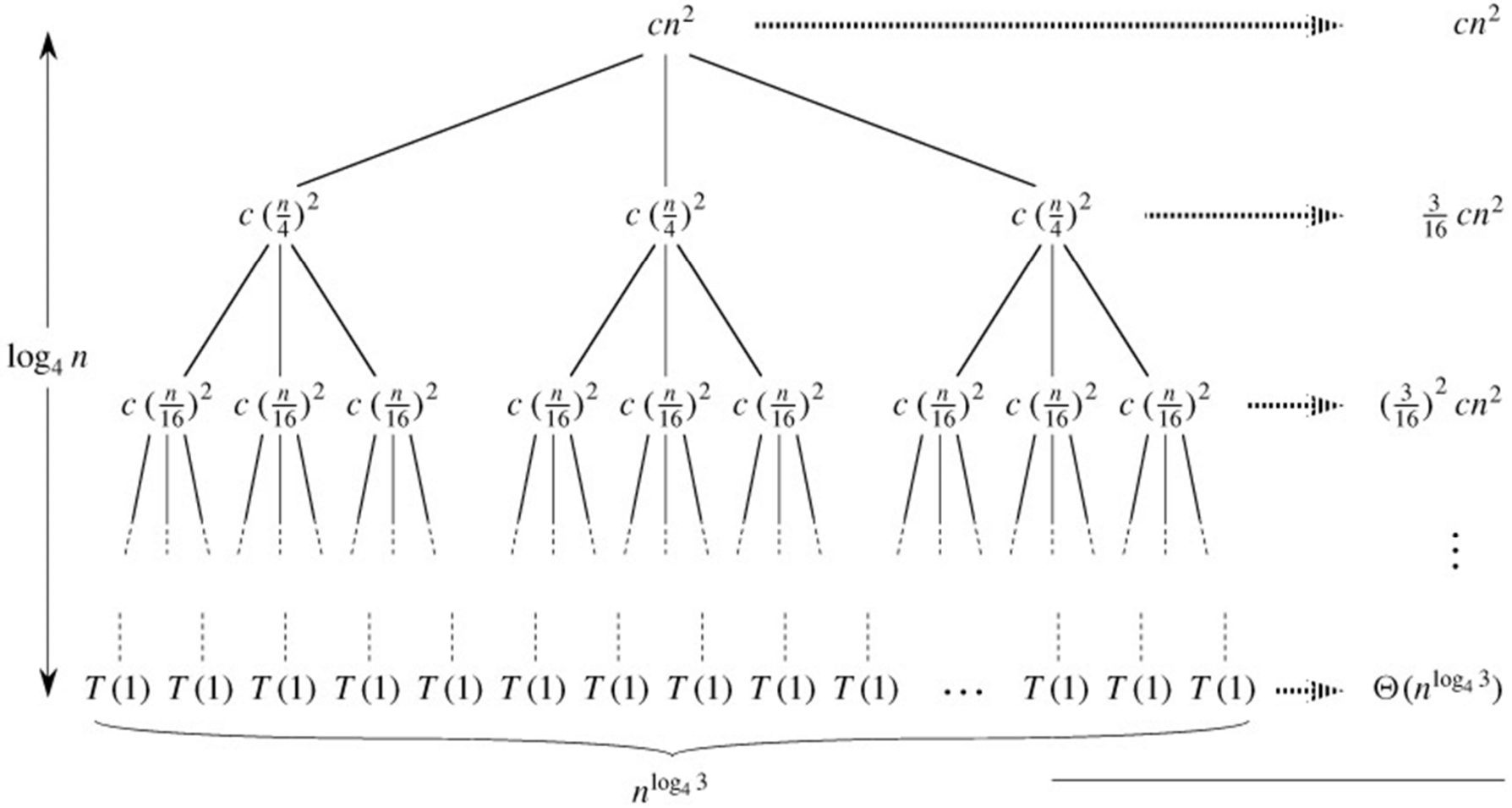
- Iteratively build more and more complete recursion trees:



- Recursion tree visualizes the recursive process, a node represents a subprocess and it is labeled by its complexity. The sum of all labels must be the equal to the complexity $T(n)$ specified in the given recurrence.

Recursion-tree method - Example

- The resulting tree looks like:



- Sums of the labels in particular tree depths are listed to the right.
  By adding sums in all levels we obtain the resulting complexity: $O\left(n^2\right)$

Recursion-tree method - Example

- Adding the sums in particular depths might be done as follows:

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

Using the "cookbook"

- Master theorem is applicable to the recurrences of the form:

$$T(n) = a\,T(n/b) + f(n)$$

Where $a \geq 1$ and $b > 1$ are constants

and $f(n)$ is asymptotically positive function.

- Rounding the term $T(n/b)$ to either $T(\lfloor n/b \rfloor)$ or $T(\lceil n/b \rceil$ does not affect the resulting complexity in this case.

Using the "cookbook"

- Master theorem
  - Let $a \geq 1,\ b > 1$ be constants, let $f(n)$ be a function and let $T(n)$ be defined for non-negative integers by the recurrence

$$T(n) = a\,T(n/b) + f(n)$$

where n/b means $\lceil n/b \rceil$ or $\lfloor n/b \rfloor$. Then the following holds.

1. If $f(n) \in O(n^{\log_b(a)-\varepsilon})$ for some constant $\varepsilon > 0$, then

$$\boldsymbol{T(n) \in \Theta(n^{\log_b(a)})}.$$

2. If $f(n) \in \Theta(n^{\log_b(a)})$, then

$$\boldsymbol{T(n) \in \Theta(n^{\log_b(a)} \log(n))}.$$

3. If $f(n) \in \Omega(n^{\log_b(a)+\varepsilon})$ for some constant $\varepsilon > 0$ and if $a \cdot f(n/b) \leq c \cdot f(n)$ for some constant $c < 1$ and all sufficiently big n, then

$$\boldsymbol{T(n) \in \Theta(f(n))}.$$

Using the "cookbook" Example 1

- Example 1:

  $T(n) = 9T(n/3) + n$

- The parameters are: $a = 9$, $b = 3$, $f(n) = n \in O(n^{\log_3(9)-1})$.

  It is the case 1 of Master theorem.

- The resulting complexity is thus:

  $T(n) \in \Theta(n^{\log_3(9)}) = \boldsymbol{\Theta(n^2)}$.

Using the "cookbook" Example 2

- Example 2:

  $T(n) = T(2n/3) + 1$

- The parameters are: $a = 1$, $b = 3/2$,
- $$f(n) = 1 = n^{\log_{3/2}(1)} \in \Theta\left(n^{\log_{3/2}(1)}\right) .$$

  It is the case 2 of Master theorem.

- The resulting complexity is thus:

  $$T(n) \in \Theta\left(n^{\log_{3/2}(1)} \log(n)\right) = \boldsymbol{\Theta(\log(n))}$$

Using the "cookbook" Example 3

- Example 3: $T(n) = 3T(n/4) + n\log(n)$
- The parameters are: $a = 3$, $b = 4$,

  $f(n) = n\log(n)$ and we know that $n^{\log_4(3)} = O(n^{0.793})$.

  Therefore, we can state: $f(n) \in \Omega(n^{\log_4(3)+0.2})$.

  To satisfy conditions of case 3 it must hold for all $c < 1$ and all sufficiently big n that $a\,f(n/b) \le c\,f(n)$. It really does: $a\,f(n/b) = 3(n/4)\log(n/4) \le (3/4)n\log(n) = c\,f(n)$, for $c = ¾$.
- The resulting complexity is thus:

  $T(n) \in \boldsymbol{\Theta(n\log(n))}$