1. Introduction
2. Methods for solving recurrence relations
   - Substitution
   - Recursion-tree
   - Master theorem
3. Appendix : Linear Recurrence Relations

Introduction : Recurrences

1. Recurrences go hand in hand with the divide-and-conquer (DC) paradigm because they give us a natural way to characterize the running time of DC algorithms.
2. A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.
3. Recurrences have 2 types of terms : The *recursive* term(s) and the *non-recursive* terms.
   - The *recursive* terms refer to the recurrence relation.
   - The *non-recursive* terms refer to the time in one execution of the recursive function.
4. Example : The worst-case running time of the Merge-Sort procedure is given by a recurrence form :

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

A few relations to remember

$\log_a(xy) = \log_a x + \log_a y$
$\log_a \frac{x}{y} = \log_a x - \log_a y$
$\log_a(x^r) = r \log_a x$
$\log_a a = 1; \log_a 1 = 0$
$\log_a \frac{1}{x} = -\log_a x$
$\log_a x = \frac{\log x}{\log a}$
$x^{\frac{a}{b}} = \sqrt[b]{x^a}$
$x^{1/b} = z$ means $z^b = x$; $x^{a/b} = z$ means $z^b = x^a$

Recurrence for Factorial

```
int factorial(int n)
if (n==1)
   return 1;
else
   return n*factorial(n-1);
```

Running-time of factorial, $T(n)$, is given by the following recurrence relation :

$$T(n) = \begin{cases} 1 & \text{when } n = 1 \\ T(n-1) + 1 & \text{otherwise} \end{cases}$$

Mapping recurrences to algorithms

How do these terms relate to algorithms?

- Recursive terms refer to recursive calls.
- Non-recursive terms computation in the subroutine, including any splitting or combining of data.

```
int factorial(int n)
    if (n==1)
       return 1;
    else
       return n*factorial(n-1);
```

$$T(n) = \begin{cases} 1 & \text{when } n = 1 \\ T(n-1) + 1 & \text{otherwise} \end{cases}$$

Recurrence for Binary Search

procedure $BinarySearch(L[i..j], x)$
  if $i = j$ **then return** i
  $k = \frac{(i+j)}{2}$;
  if $x \leq L[k]$ **then**
    return $BinarySearch(L[i..k], x)$
  else
    return $BinarySearch(L[k+1..j], x)$

- To solve an instance of binary search, we need to do a compare operation, followed by an instance of binary search of half the size.
- Thus, the recurrence for the (worst-case) run-time of this algorithm is

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + 1$$

Other examples of recurrences

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

Solution : $T(n) = n$

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n \geq 1 \end{cases}$$

Solution : $T(n) = n \lg n + n$

$$T(n) = \begin{cases} 0 & \text{if } n = 2 \\ T(\sqrt{n}) + n & \text{if } n > 2 \end{cases}$$

Solution : $T(n) = \lg \lg n$

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

Solution : $T(n) = \Theta(n \lg n)$

How to analyze recursive functions

Identify the number of elementary operations perform in each call of the recursive function. Usually elementary operations are :

1. operations executed in each instantiation of the r.f. to divide its input in several recursive calls and
2. operations executed in each instantiation of the r.f. to combine the solutions of the recursive calls.

Write the recurrence relation (rather than the summation) expressing the number of basic operations executed based on n, the input.

Find the closed form of the recurrence relation.

The substitution method

The substitution method consists of two steps :

1. Guess the form of the solution.
2. Use mathematical induction to show that the guess is correct.

It can be used to obtain either upper ($O()$) or lower bounds ($\Omega()$) on a recurrence.

A good guess is vital when applying this method. If the initial guess is wrong, the guess needs to be adjusted later.

Example 1

Solve the recurrence $T(n) = T(n-1) + 2n - 1$, $T(0) = 0$

Initial guess. Use the method of forward substitution :

n	0	1	2	3	4	5
$T(n)$	0	1	4	9	16	25

Guess is $T(n) = n^2$

Proof by induction :

Base case $n = 0$: $T(0) = 0 = 0^2$
Induction step : Assume the inductive hypothesis is true for $n = n - 1$.
We have

$$\begin{aligned} T(n) &= T(n-1) + 2n - 1 \\ &= (n-1)^2 + 2n - 1 \text{ (Induction hypothesis)} \\ &= n^2 - 2n + 1 + 2n - 1 \\ &= n^2 \end{aligned}$$

Example 2

Solve the recurrence $T(n) = T(\lfloor \frac{n}{2} \rfloor) + n,\ T(0) = 0$

Guess using the method of forward substitution :

n	0	1	2	3	4	5	8	16	32	64
$T(n)$	0	1	3	4	7	8	15	31	63	127

Guess : $T(n) \leq 2n$

Base case $n = 0$: $T(n) = 0 \leq 2 \times 0$
Induction step : Assume the inductive hypothesis is true for some $n = \lfloor \frac{n}{2} \rfloor$. We have
$T(n) = T(\lfloor \frac{n}{2} \rfloor) + n <= 2\lfloor \frac{n}{2} \rfloor + n <= 2(n/2) + n = 2n$, i.e.
$T(n) \in O(n)$

Example 3

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

Guess $T(n) \leq cn \log n$ for some constant c (that is, $T(n) = O(n \log n)$)

Proof :
Base case : $T(1) = 1 = nlogn + n$ (note : we need to show that our guess holds for some base case (not necessarily $n = 1$, some small n is ok).
Induction step : Assume the inductive hypothesis holds for $n/2$ (n is a power of 2) : $T(n/2) \leq c\frac{n}{2} \log \frac{n}{2}$.

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\leq 2(c\frac{n}{2} \log \frac{n}{2}) + n \\ &= cn \log \frac{n}{2} + n \\ &= cn \log n - cn \log 2 + n \\ &= cn \log n - cn + n \end{aligned}$$

Which is true if $c \geq 1$

Exercise

Assume that the running time $T(n)$ satisfies the recurrence relation

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n.$$

Show that $T(n) \in \Theta(n \lg n)$.

Remark : To have a good guess, we could use the recursion-tree or changing variables to reduce to the linear recurrence that could be solved (see Appendix).

The recursion tree

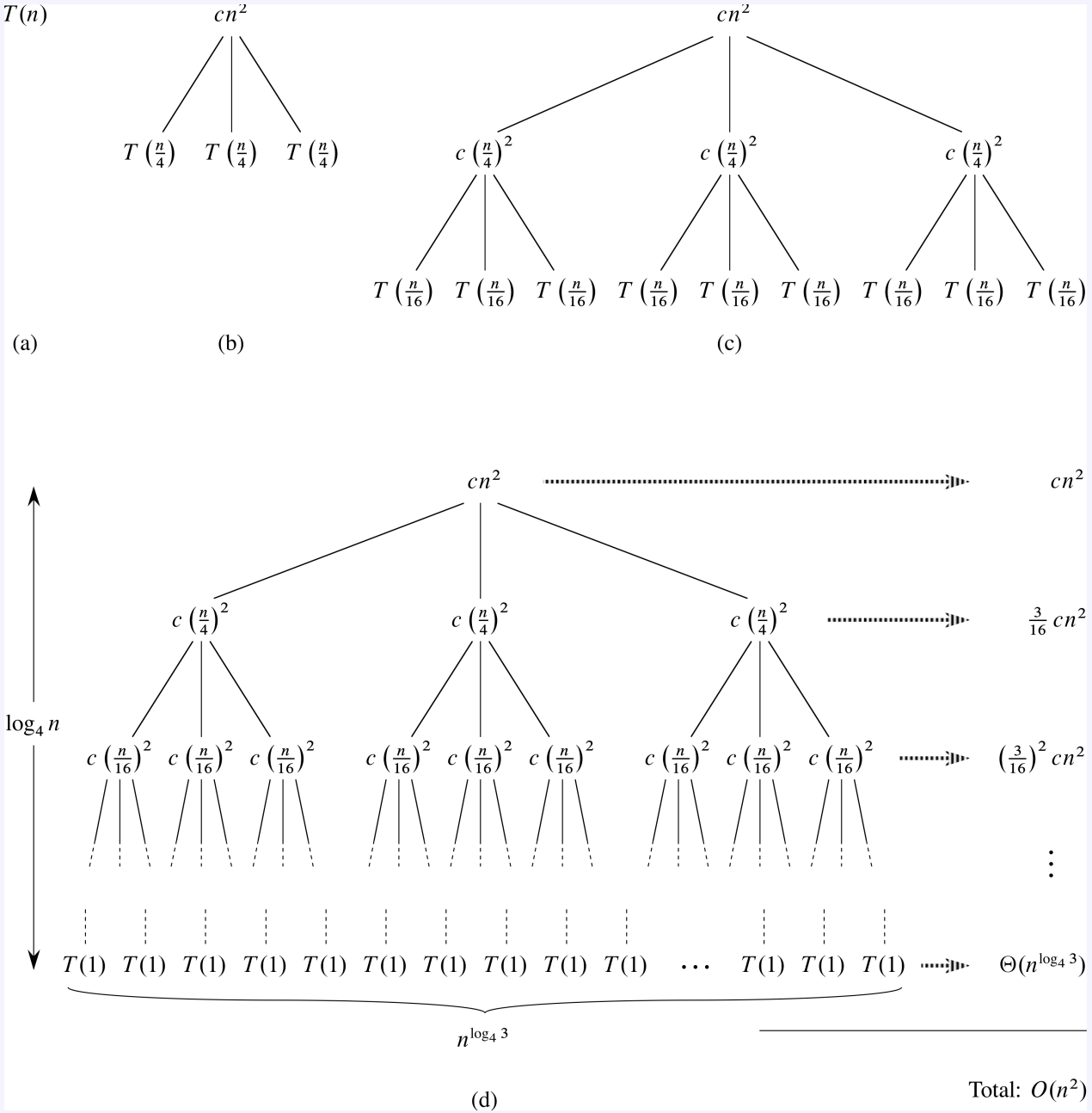
To analyze the recurrence by using the recursion tree, we will

- draw a recursion tree with cost of single call in each node.
- find the running time at each level of the tree by summing the running time of each call at that level.
- find the number of levels
- Last, the running time is sum of costs in all nodes.

Example of a recurrence relation :

$$T(n) = 3T(\lfloor n/4 \rfloor) + cn^2.$$

The recursion tree



The recursion tree

Remark : The number of levels depends by how much subproblem sizes decrease. In this example, subproblems decrease by a factor of 4 at each new level, the level where $n = 1$ is where $n/4^i = 1$, i.e. when $i = \log_4 n$. Therefore, the number of terms in the above summation = number of levels in the tree = $log_4 n$. More precisely,

$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \cdots + (\frac{n}{16})^{log_4(n-1)} cn^2 + cn^{\log_4 3} \\ &= cn^2 \left[1 + \frac{3}{16} + (\frac{3}{16})^2 + \cdots + (\frac{3}{16})^{log_4(n-1)}\right] + cn^{\log_4 3} \\ &= O(n^2) \quad \text{(as geometric series and maximum rule)} \end{aligned}$$