MATHEMATICAL ANALYSIS OF RECURSIVE FUNCTION

Chapter3

Prepared by: Enas Abu Samra

KEY POINTS OF CHAPTER3

- Recursive function definition.
- Idea of recursive function.
- Examples of recursive functions.
- Common recurrence relations.
- Methods for solving recurrences:
  - ✓ Master Method
  - ✓ Recursion Tree Method
  - ✓ Iteration Method
  - ✓ Substitution Method

What is the Recursive Function?

RECURSIVE FUNCTION DEFINITION

- **Recursion** is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

- **Algorithmically:** a way to design solutions to problems by **divide-and-conquer**.
- **Semantically:** a programming technique where a function **calls itself**.

DIVIDE AND CONQUER TECHNIQUE

- Divide and conquer is an **algorithm design paradigm** based on multi-branched recursion. It works by recursively **breaking down** (reducing) a problem into (two or more) sub-problems of the **same (or related type),** until these become simple enough to be solved directly. The solutions to the sub-problems are then **combined** to give a solution to the original problem.

IDEA OF RECURSIVE FUNCTION

- **We can distill the idea of recursion into two simple rules:**
  - Each recursive call should be on a smaller instance of the same problem, that is, a smaller subproblem.
  - The recursive calls must eventually reach a **base case**, which is solved without further recursion.

Note:

If you forget to include a base case, or your recursive cases fail to eventually reach a base case, then infinite recursion happens. **Infinite recursion** is a special case of an infinite loop when a recursive function fails to stop recursing.

EXAMPLE (1) OF RECURSIVE FUNCTION

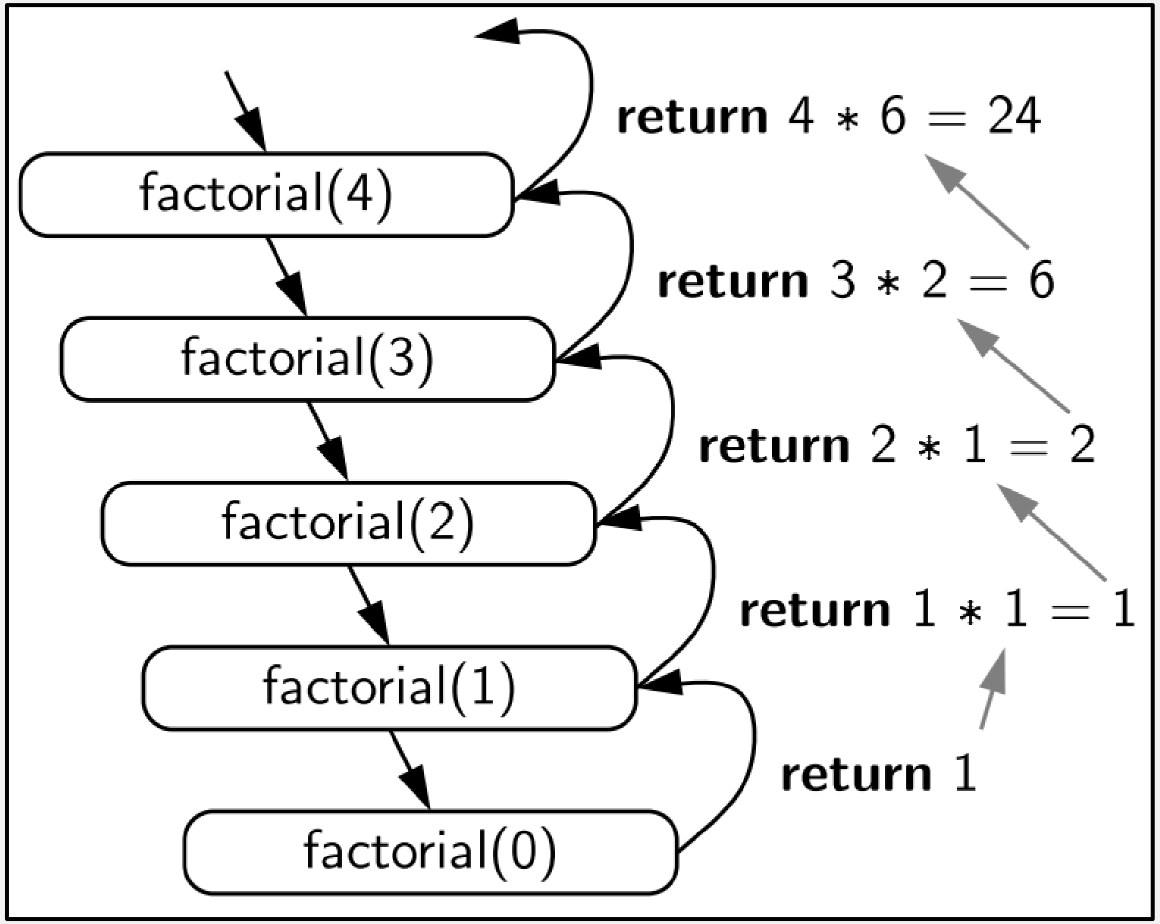
```
def factorial(n):
    if n == 0:   # base case
        return 1
    else:        # recursive case
        return n * factorial(n - 1)

print(factorial (4))
```

TRACING OF RECURSIVE FUNCTION

- The execution of a recursive function is usually illustrated using a **recursion trace**.
- **Each new recursive function call** is indicated by a **downward arrow** to a new invocation.
- **When the function returns,** an arrow showing this return is drawn and the return value may be indicated alongside this arrow.

RECURSION TRACE FOR EXAMPLE(1)



COMPLEXITY OF EXAMPLE(1)

```
def factorial(n):
    if n == 0:                          →1
        return 1                        →1
    else:
        return n * factorial(n - 1)     → T(n-1)+2
```

$$T(n) = \begin{cases} 1 & , n = 0 \\ T(n-1) + 3 & , n > 0 \end{cases}$$

EXAMPLE (2) OF RECURSIVE FUNCTION

Mathematical Analysis of Recursive Algorithms

$$T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+1 & n>0 \end{cases}$$

```
void Test(int n)        ------------T(n)
  {
     If (n>0)
      {
         Print (n)             ---------- 1
         Test(n-1)             -----------  T(n-1)

      }
   }
```

$$T(n)= T(n-1) +1$$

What is the Recurrence Relations?

COMMON RECURRENCE RELATIONS

- **A recurrence relation** is an equation that defines a sequence based on a rule that gives the next term as a function of the previous term(s).

$T(n)=T(n-1)+1 \rightarrow O(n)$

$T(n)=T(n-1)+n \rightarrow O(n^2)$

$T(n)=T(n-1)+\log n \rightarrow O(n\log n)$

$T(n)=T(n-1)+n^2 \rightarrow O(n^3)$

$T(n)=T(n-2)+1 \rightarrow n/2 \rightarrow O(n)$

$T(n)=T(n-100)+n \rightarrow n^2/100 \rightarrow O(n^2)$

$T(n)=2T(n-1) + 1 \rightarrow O(2^n)$

COMMON RECURRENCE RELATIONS

$T(n)=3T(n-1)+1 \rightarrow O(3^n)$

$T(n)= 2T(n-1)+n \rightarrow O(n2^n)$

$T(n) = 3T(n-1)+\log n \rightarrow O(3^n \log n)$

$T(n) = T(n/2) + 1 \rightarrow O(\log n)$

$T(n)=T(n/2) + n \rightarrow O(n)$

$T(n)=2T(n/2)+n \rightarrow O(n\log n)$

METHODS FOR SOLVING RECURRENCES

- ✓ Master Method
- ✓ Recursion Tree Method
- ✓ Iteration Method
- ✓ Substitution Method

The Master Method

MASTER METHOD

$T(n) = aT(\frac{n}{b}) + f(n)$, **where $a >= 1$, $b > 1$, $f(n) > 0$, $d = \log_b a$**

$$T(n) = \begin{cases} \Theta(n^d) & , \text{ if } f(n) < n^d \quad \textbf{case1} \\ \Theta(n^d \log n) & , \text{ if } f(n) = n^d \quad \textbf{case2} \\ \Theta(f(n)) & , \text{ if } f(n) > n^d \quad \textbf{case3} \end{cases}$$

In case3, this is another condition: $af(\frac{n}{b}) <= cf(n)$ where $c < 1$

MASTER EXAMPLES

Ex1. $T(n) = 4T(\frac{n}{2}) + n$

Sol ➔ f(n) n^d

n $n^{\log_2 4}$

$n < n^2$

$T(n) = \Theta(n^2)$

MASTER EXAMPLES

Ex2. $T(n) = 2T(\frac{n}{2}) + n$

Sol ➔ f(n) n^d

n $n^{\log_2 2}$

$n = n$

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

MASTER EXAMPLES

Ex3. $T(n) = 2T(\frac{n}{2}) + n^2$

Sol ➔ $f(n)$ n^d

n^2 $n^{\log_2 2}$

$n^2 > n$

$af(n/b)$ $cf(n)$

$2(\frac{n2}{4})$ cn^2

$(\frac{n2}{2}) <= cn^2$ ➔ $\frac{1}{2}(n^2) <= cn^2$ ➔ **$c >= 0.5$**

$T(n) = \Theta(n^2)$

The Recursion Tree Method

RECURSION TREE METHOD

$T(n) = aT(\frac{n}{b}) + f(n)$, $T(1) = C$

number of nodes ➔ a

size of node ➔ (n/b)

root ➔ f(n)

size of node in leaves ➔ $T(1) = C$

Note: It is not necessary that the T(n) be always in this form.

STEPS OF RECURSION TREE METHOD

1.Draw Tree

First→ draw the root (look at **f(n):** size of root)

Second→ draw nodes (look at **a:** # of nodes and $\frac{n}{b}$ **:** size of node)

Third → draw the leaves (look at **T(1):** size of the leaves)

2.Draw Table

Level	# of Nodes	Level Sum
0	1	Weight of node * number of nodes
1	?	Weight of node * number of nodes
2	?	Weight of node * number of nodes
. . .	. . .	. . .
i	? = n	T(1) * n

3.Find (i) and Pattern

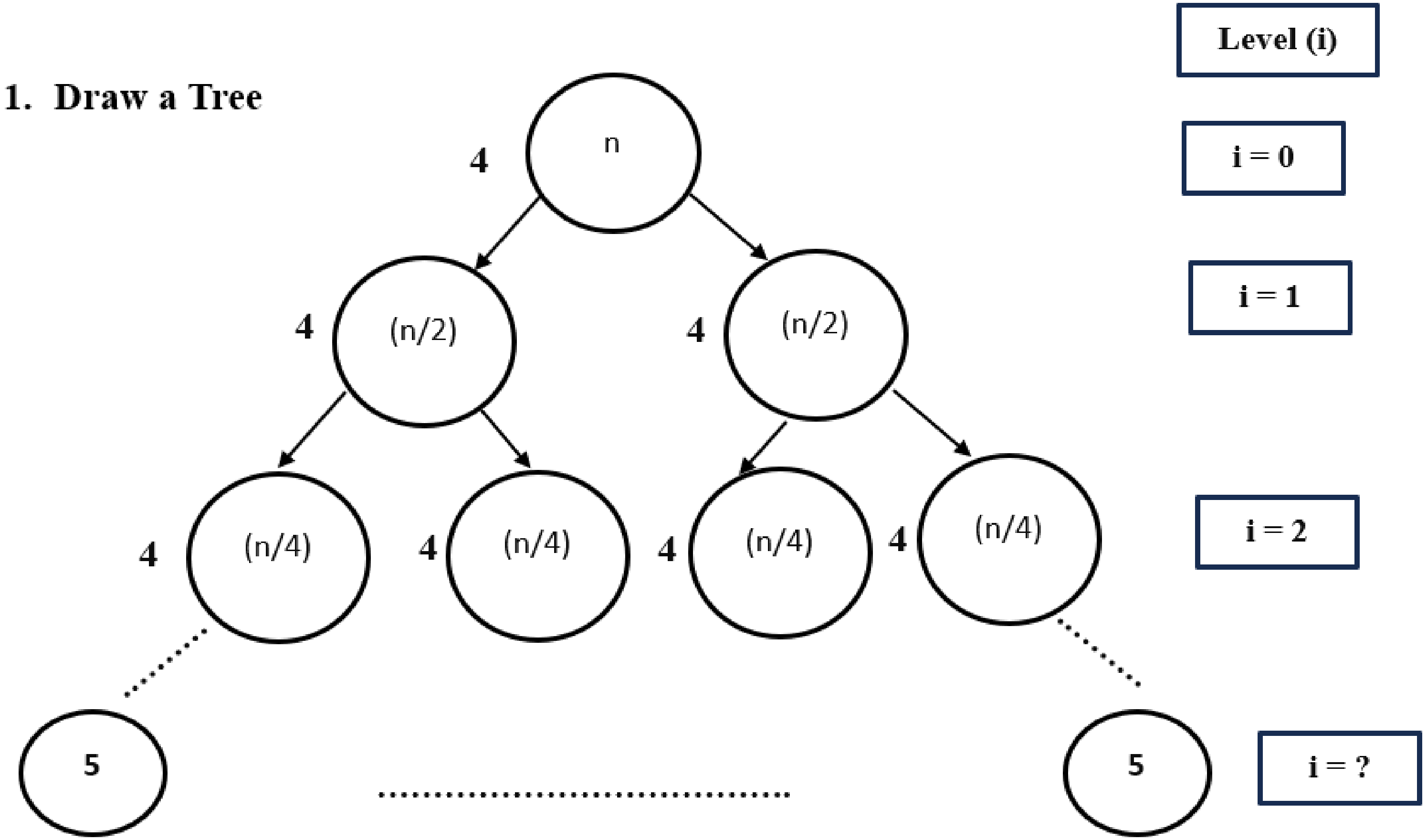
Find i→ from this equation: ? = n

Find Pattern→ make series from the **Level Sum Column** then use summation (if required) and finally find the complexity.

RECURSION TREE EXAMPLES

Ex1. $T(n) = 2T(\frac{n}{2}) + 4n$, $T(1) = 5$

1. **Draw a Tree**



RECURSION TREE EXAMPLES

2. Draw a Table

Level	# of Nodes	Level Sum
0	1	4n
1	2	4n
2	4	4n
. . .	. . .	. . .
i	2^i	5n

3. Find(i) and Pattern

$2^i = n$ ➔ **i = logn**

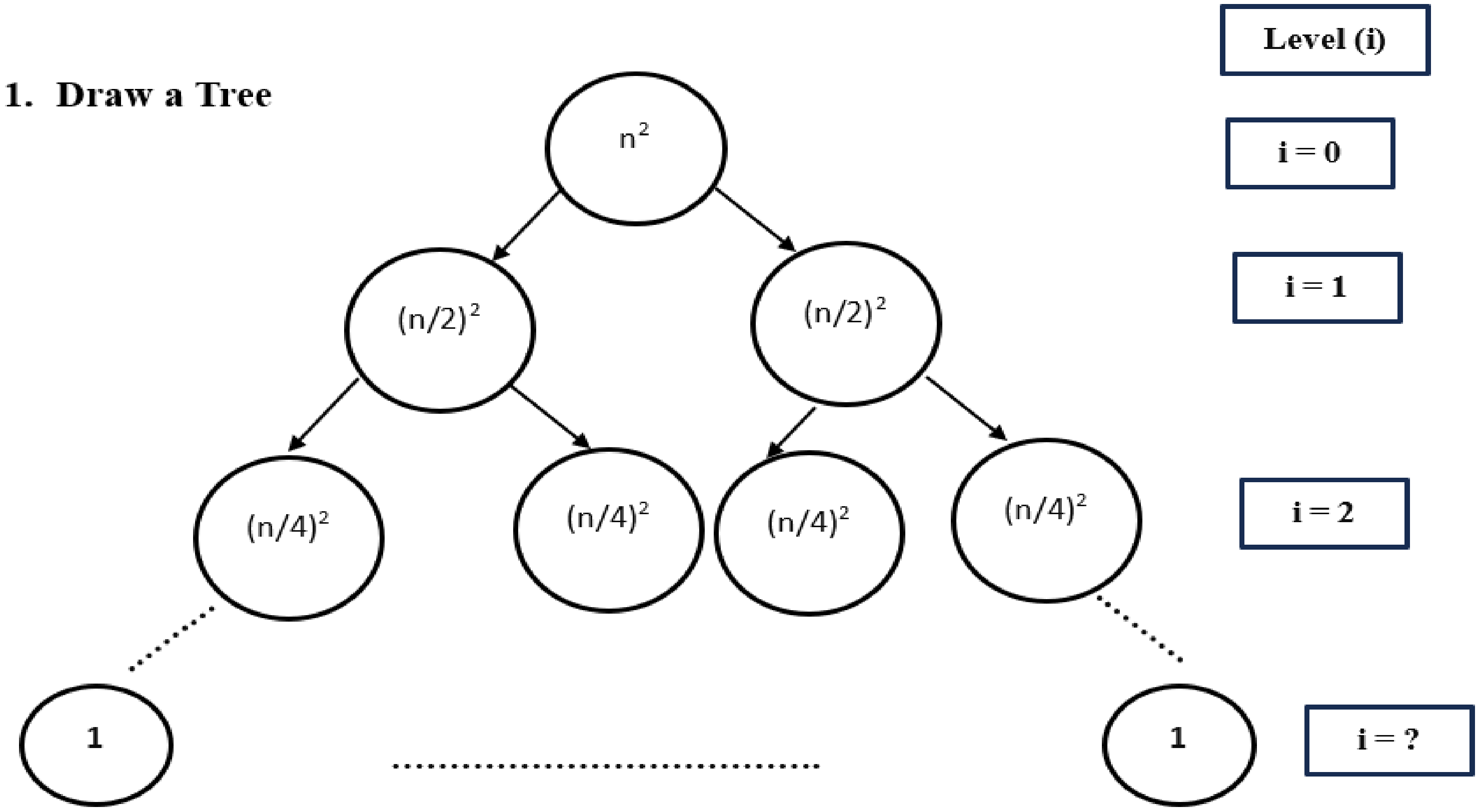
Pattern ➔ $\sum_{i=0}^{i} 4n + n = \sum_{i=0}^{logn} 4n + n = 4n \sum_{i=0}^{logn} 1 + n =$
$4n(logn - 0 + 1) + n = 4nlogn + 5n$

T(n) = Θ(nlogn)

RECURSION TREE EXAMPLES

Ex1. $T(n) = 2T(\frac{n}{2}) + n^2$, $T(1) = 1$

1. **Draw a Tree**



RECURSION TREE EXAMPLES

2. Draw a Table

Level	# of Nodes	Level Sum
0	1	n2
1	2	$n^2/2$
2	4	$n^2/4$
. . .	. . .	. . .
i	2^i	n

3. Find(i) and Pattern

$2^i = n$ ➔ **i = logn**

$$\boxed{n^2 + n^2/2 + n^2/4 + \ldots. + n^2/2^{i-1} + n}$$

Pattern ➔ $\sum_{i=0}^{i-1} \frac{n^2}{2^i} + n = n^2 \sum_{i=0}^{i-1} \frac{1}{2^i} + n = n^2 \sum_{i=0}^{logn-1} \frac{1}{2^i} + n = n^2 * c + n$

$T(n) = \Theta(n^2)$

The Iteration Method

ITERATION METHOD

Solution Steps:

1. Iteration Step
2. Find (i) and pattern

ITERATION EXAMPLES

Ex1. $T(n) = T(\frac{n}{2}) + C$, $T(1) = 1$

Sol:

1. **Iteration Step**

Level	**T(n)**
1	$T(n) = T(\frac{n}{2}) + C$
2	$T(n) = T(\frac{n/2}{2}) + C + C \rightarrow T(\frac{n}{4}) + C + C$
3	$T(n) = T(\frac{n/2}{4}) + C + C + C \rightarrow T(\frac{n}{8}) + C + C + C$
. . .	. . .
i	$T(n) = T(n/2^i) + \underbrace{(C + C + C + \ldots + C)}_{i}$

ITERATION EXAMPLES

2. Find (i) and Pattern

$T(1) = T(n/2^i)$ ➔ **$i = \log n$**

$$T(n) = T(n/2^i) + \sum_{i=0}^{i} C =$$

$$T(n/2^{\log n}) + C\sum_{i=0}^{\log n} 1 =$$

$$T(1) + C(\log n - 0 + 1) =$$

$$1 + C*\log n + C$$

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

ITERATION EXAMPLES

Ex2. $T(n) = T(n-1) + n$, $T(1) = 0$

Sol:

1. **Iteration Step**

Level	**T(n)**
1	T(n) = T(n-1) + n
2	T(n) = T(n-2) + (n-1) + n
3	$T(n) = T(n-3) + (n-2) + (n-1) + n$
. . .	. . .
i	T(n) = T(n -i) + (n -i -1)+ …. + (n-3) +(n-2) + (n-1) + n

ITERATION EXAMPLES

2. Find (i) and Pattern

T(1) = T(n -i) ➔ **i = n**

$$T(n) = T(n-i) + \sum_{i=0}^{i}(n-i) =$$

$$T(n-n) + \sum_{i=0}^{n} n - \sum_{i=0}^{n} i$$

$$T(1) + n(n - 0 + 1) - n(n+1)/2$$

$$0 + n^2 + n + n(n+1)/2$$

$$T(n) = \Theta(n^2)$$

The Substitution Method

SUBSTITUTION METHOD

1. Guess a solution.
2. Use induction to prove that the solution works. (Substitution in base and recurrence cases)

SUBSTITUTION EXAMPLES

Ex1.

$$T(n) = \begin{cases} 2 & , n = 1 \\ T(\frac{n}{2}) + \mathbf{2} & , n > 1 \end{cases}$$

Guessed solution ➔ $T(n) = 2\log n + 2$

SUBSTITUTION EXAMPLES

Solution:

Base Case ➔ n = 1

T(1) = log(1) + 2 = 2 (same as the function, correct)

Recurrence Case ➔ n > 1,

$T(\frac{n}{2}) = 2\log\frac{n}{2} + 2$ **+ 2** $=$ 2logn – 2log2 + 2 **+ 2** =

2logn – 2 + 2 + 2 =

2logn + 2 (same as the function, correct)

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

SUBSTITUTION EXAMPLES

Ex2.

$$T(n) = \begin{cases} 1 & , n = 1 \\ \color{red}{\mathbf{2}}T(\frac{n}{2}) \color{red}{\mathbf{+ n}} & , n > 1 \end{cases}$$

Guessed solution ➔ T(n) = nlogn + n

SUBSTITUTION EXAMPLES

Solution:

Base Case ➔ $n = 1$

$T(1) = 1\log 1 + 1 = 1$ (same as the function, correct)

Recurrence Case ➔ $n > 1$,

$T(\frac{n}{2}) = \mathbf{2}\ [\frac{n}{2}\log\frac{n}{2} + \frac{n}{2}]\ \mathbf{+\ n} =$

$n\log\frac{n}{2} + n \mathbf{+ n} = n(\log n - \log 2) + n + n$

$= n\log n - n + n + n$

$n\log n + n$ (same as the function, correct)

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

EXTRA EXAMPLE

- Solve the following example using master, recursion tree and iteration methods:

$$\mathbf{T(n) = T(\frac{\boldsymbol{n}}{3}) + n}$$

T(1) = 1

END OF CHAPTER3