

Indian Institute of Technology Madras
ZANZIBAR CAMPUS

Z5007 · PROGRAMMING AND DATA STRUCTURES

Analyzing Recursion: Recurrences, Recursion Trees and the Master Theorem

Lecture 18 · Friday 6 November 2026 · CLRS Ch 4 (4.4, 4.5)

Dr. Innocent Nyalala | MTech Semester 1 | IIT Madras Zanzibar

Today's Plan

IIT Madras Zanzibar

1. (0–8 min) From code to recurrence: three questions to ask of any recursive function
2. (8–15 min) Solving by unrolling: $T(n - 1) + n$ and $T(n/2) + 1$
3. (15–26 min) Recursion trees: work per level; three shapes; counting the leaves
4. (26–38 min) The Master Theorem: compare a with b^d; many worked examples
5. (38–42 min) When the theorem does not apply, and what to do instead
6. (42–50 min) Live demo, practice questions, your questions

Learning Outcomes

IIT Madras Zanzibar

By the end of this session, you will be able to:

Write	the recurrence of a recursive function from its code
Solve	simple recurrences by unrolling them
Draw	a recursion tree and add up the work level by level
Apply	the Master Theorem $T(n) = aT(n/b) + O(n^d)$ by comparing a with b^d
Recognize	recurrences the theorem cannot solve, and solve them another way

Key Terms for Today

IIT Madras Zanzibar

Term	Meaning	Example
Recurrence	cost of size n written with the cost of smaller sizes	$T(n) = 2T(n/2) + n$
a (branching)	how many recursive calls each call makes	merge sort: a = 2
b (shrink factor)	the input of each call is n / b	halving: b = 2
$f(n) = n^d$	work done in the call itself, outside the recursive calls	merge: n, so d = 1
Unrolling	substituting the recurrence into itself until a pattern shows	$T(n) = T(n - 2) + \ldots$
Recursion tree	every call drawn as a node, labelled with its own work	levels of n/2, n/4, …
Geometric series	a sum where each term is a fixed multiple of the last	$n + n/2 + n/4 + \ldots < 2n$

Recurrences and the Master Theorem

IIT Madras Zanzibar

→ **From Code to Recurrence**

Solving by Unrolling

Recursion Trees: Three Shapes

The Master Theorem

When It Does Not Apply

Demo and Practice

Three Questions Turn Code into a Recurrence

IIT Madras Zanzibar

```
def f(a, lo, hi):
    if hi - lo <= 1: return       # base case
    for i in range(lo, hi): ...   # own work: n
    mid = (lo + hi) // 2
    f(a, lo, mid)                 # call 1, size n/2
    f(a, mid, hi)                 # call 2, size n/2
```

1 **How many calls?**

a = 2

2 **On what size?**

n / 2, so b = 2

3 **Own work?**

a loop of n: f(n) = n, d = 1

Recurrence: $T(n) = 2\,T(n/2) + n$ $\quad (a = 2, b = 2, d = 1)$

? **Think:** three calls on n // 2 and no loop. What is the recurrence?

The Recurrences of This Week

IIT Madras Zanzibar

Algorithm	Calls (a)	Size	Own work	Recurrence
binary search	1	n/2	constant	T(n) = T(n/2) + 1
factorial, sum_list	1	n − 1	constant	T(n) = T(n − 1) + 1
merge sort	2	n/2	merge: n	T(n) = 2T(n/2) + n
quicksort, best	2	n/2	partition: n	T(n) = 2T(n/2) + n
quicksort, worst	1 useful	n − 1	partition: n	T(n) = T(n − 1) + n
Hanoi	2	n − 1	one move	T(n) = 2T(n − 1) + 1
quickselect, average	1	n/2	partition: n	T(n) = T(n/2) + n

? **Think:** which rows divide n (n/b) and which subtract (n − 1)?

Recurrences and the Master Theorem

From Code to Recurrence

→ **Solving by Unrolling**

Recursion Trees: Three Shapes

The Master Theorem

When It Does Not Apply

Demo and Practice

Unrolling T(n) = T(n – 1) + n

IIT Madras Zanzibar

$$
\begin{aligned}
T(n) &= T(n-1) + n \\
&= T(n-2) + (n-1) + n \\
&= T(n-3) + (n-2) + (n-1) + n \\
&= \dots \\
&= T(1) + 2 + 3 + \dots + n \\
&= 1 + 2 + \dots + n = n(n+1)/2 \;\rightarrow\; O(n^2)
\end{aligned}
$$

n	T(n)	n(n+1)/2
4	10	10
8	36	36
16	136	136

? **Think:** unroll T(n) = T(n – 1) + 1, with T(1) = 1. Result?

Unrolling T(n) = T(n/2) + 1

IIT Madras Zanzibar

$$T(n) = T(n/2) + 1$$
$$= T(n/4) + 1 + 1$$
$$= T(n/8) + 1 + 1 + 1$$
$$= T(n/2^k) + k$$

$n/2^k = 1$ when $k = \log_2 n$

$T(n) = T(1) + \log_2 n = \log_2 n + 1 \rightarrow O(\log n)$

n	T(n)	$\log_2 n+1$
8	4	4
1,024	11	11
10^6	20	20.9

? **Think:** unroll T(n) = T(n/2) + n. What do you get?

Recurrences and the Master Theorem

From Code to Recurrence

Solving by Unrolling

→ **Recursion Trees: Three Shapes**

The Master Theorem

When It Does Not Apply

Demo and Practice

The Recursion Tree, Level by Level

IIT Madras Zanzibar

Level	Calls	Size of each	Work at the level
0	1	n	n^d
1	a	n/b	$a\ (n/b)^d$
2	a^2	n/b^2	$a^2\ (n/b^2)^d$
k	a^k	n/b^k	$n^d\ (a/b^d)^k$
log_b n	a^(log_b n)	1	leaves

Ratio: $r = a / b^d$. Each level does r times the work of the level above.

1. $r < 1$: shrinking: the top wins.
2. $r = 1$: equal: levels × top.
3. $r > 1$: growing: the leaves win.

? **Think:** for merge sort (a = 2, b = 2, d = 1), what is r? What does it mean?

Three Shapes of Tree

IIT Madras Zanzibar

2T(n/2) + n (r = 1)



equal levels: n × $\log_2 n$
→ O(n log n)

2T(n/2) + n^2 (r = 1/2)



top dominates: < $2n^2$
→ O(n^2)

4T(n/2) + n (r = 2)



leaves dominate: n^($\log_2 4$) = n^2
→ O(n^2)

? **Think:** in the middle shape, why may we ignore every level below the top?

Counting the Leaves: n^(log_b a)

IIT Madras Zanzibar

leaves = a^(log_b n) = n^(log_b a)

a	b	leaves n^(log_b a)	Example
2	2	n	merge sort: n leaves of size 1
4	2	n^2	4T(n/2) + n
3	2	n^1.585	Karatsuba multiplication of large numbers
7	2	n^2.807	Strassen matrix multiplication
1	2	$n^0 = 1$	binary search: one path, one leaf

? **Think:** a = 8, b = 2: how many leaves?

Recurrences and the Master Theorem

From Code to Recurrence

Solving by Unrolling

Recursion Trees: Three Shapes

→ **The Master Theorem**

When It Does Not Apply

Demo and Practice

The Master Theorem

IIT Madras Zanzibar

$$T(n) = a \cdot T(n/b) + O(n^d), \quad a \geq 1, \; b > 1, \; d \geq 0$$

$a < b^d$	$a = b^d$	$a > b^d$
top dominates	every level equal	leaves dominate
$O(n^d)$	$O(n^d \log n)$	$O(n^{\log_b a})$

1. Recipe: 1. read a, b, d. 2. compute b^d. 3. compare with a. 4. read off the case.

? **Think:** why compare a with b^d, and not a with b?

Worked Examples

IIT Madras Zanzibar

Recurrence	a, b, d	b^d	Compare	T(n)
binary search: T(n/2) + 1	1, 2, 0	1	$a = b^d$	O(log n)
merge sort: 2T(n/2) + n	2, 2, 1	2	$a = b^d$	O(n log n)
quickselect avg: T(n/2) + n	1, 2, 1	2	$a < b^d$	O(n)
$2T(n/2) + n^2$	2, 2, 2	4	$a < b^d$	$O(n^2)$
4T(n/2) + n	4, 2, 1	2	$a > b^d$	$O(n^2)$
Karatsuba: 3T(n/2) + n	3, 2, 1	2	$a > b^d$	O(n^1.585)
Strassen: $7T(n/2) + n^2$	7, 2, 2	4	$a > b^d$	O(n^2.807)

? **Think:** T(n) = 3T(n/3) + n: a three-way merge sort. Which case? Answer?

Recurrences and the Master Theorem

From Code to Recurrence

Solving by Unrolling

Recursion Trees: Three Shapes

The Master Theorem

→ **When It Does Not Apply**

Demo and Practice

When the Master Theorem Does Not Apply

IIT Madras Zanzibar

Recurrence	Why the theorem fails	What to do	Answer
$T(n) = T(n-1) + n$	subtracts: no n/b	unroll	$O(n^2)$
$T(n) = 2T(n-1) + 1$	subtracts	unroll / tree (Hanoi)	$O(2^n)$
$T(n) = T(n/3) + T(2n/3) + n$	two different sizes	recursion tree: n per level, about $\log_{3/2} n$ levels	$O(n \log n)$
$T(n) = 2T(n/2) + n \log n$	f(n) is not a plain n^d	recursion tree: n log n per level, log n levels	$O(n \log^2 n)$

Check before you apply: equal-sized calls? input n/b (not n − 1)? own work a plain power n^d? If not: unroll or draw the tree.

? **Think:** can the Master Theorem solve T(n) = 2T(n − 1) + 1? What is the answer?

Checking a Prediction by Measuring

IIT Madras Zanzibar

n	work counted	work / (n $\log_2$ n)	work / n^2
1,000	9,976	1.001	0.0100
10,000	133,616	1.006	0.0013
100,000	1,668,928	1.005	0.0002

1. Divide the measured work (or time) by the predicted growth.
2. Constant ratio: the prediction is right. Growing: too optimistic. Falling to 0: too pessimistic.

? **Think:** work / (n log n) keeps growing as n grows. What does that tell you?

Recurrences and the Master Theorem

From Code to Recurrence

Solving by Unrolling

Recursion Trees: Three Shapes

The Master Theorem

When It Does Not Apply

→ **Demo and Practice**

Live Demo

IIT Madras Zanzibar

```
cd ~/PDS/Class_Demos/Week_05
python3 W05_Fri_Nov06_recurrences.py
python3 W05_Fri_Nov06_recurrences.py master 3 3 1
python3 W05_Fri_Nov06_recurrences.py master 9 3 1
```

What to watch

1. master a b d prints b^d, the case and the answer.
2. Use it to check your answers to the exercise.

In-Class Exercise (7 minutes, in pairs)

IIT Madras Zanzibar

IN-CLASS TASK: a, b, d first; then compare

1. def g(n): if n <= 1: return; for i in range(n): for j in range(n): pass; g(n // 2); g(n // 2). Write the recurrence; solve it.
2. Solve T(n) = 3T(n/3) + n with the Master Theorem. Name the case.
3. Solve T(n) = 9T(n/3) + n. Name the case.
4. Solve T(n) = 2T(n/4) + 1.
5. Can the Master Theorem solve T(n) = T(n − 1) + 1? If not, solve it another way.

Solution

IIT Madras Zanzibar

1

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

$a = 2$, $b^d = 4$: $a < b^d$, the top level (the double loop) dominates.

2

2: $3T(n/3) + n \rightarrow O(n \log n)$

$a = 3$, $b^d = 3$: equal; every level does n.

3

3: $9T(n/3) + n \rightarrow O(n^2)$

$a = 9 > b^d = 3$: leaves; $n^{(\log_3 9)} = n^2$.

4

4: $2T(n/4) + 1 \rightarrow O(\sqrt{n})$

$a = 2 > b^d = 1$: leaves; $n^{(\log_4 2)} = n^{(1/2)}$.

5

5: no: it subtracts

unroll: $T(n) = T(n - 1) + 1 = \ldots = n \rightarrow O(n)$.

Common Mistakes

IIT Madras Zanzibar

1 Comparing a with b

compare a with b^d: d is the power in the own work

2 T(n − 1): theorem applied

subtracting recurrences need unrolling

3 log_a b instead of log_b a

leaves = n^(log_b a): base b, the shrink factor

4 Dropping the log

$a = b^d$ gives $n^d \log n$, not n^d

Summary and Exam Checkpoints

IIT Madras Zanzibar

From code

a = calls, n/b = size, f(n) = n^d = own work

Unrolling

for n − 1 recurrences: substitute until the pattern shows

Recursion tree

level work grows by r = a/b^d; depth log_b n; leaves n^(log_b a)

Master Theorem

a < b^d: n^d; a = b^d: n^d log n; a > b^d: n^(log_b a)

Next

Mon 9 Nov: The comparison lower bound; counting and radix sort; Quiz 1 preview

12–14 Nov: Quiz 1 window: everything up to and including today

Demo code: Class_Demos/Week_05/W05_Fri_Nov06_recurrences.py

References and Further Reading

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2022). Introduction to Algorithms, 4th ed. MIT Press. Sections 4.3–4.5 (substitution, recursion trees, the master method).

[2] Dasgupta, S., Papadimitriou, C. H. & Vazirani, U. V. (2008). Algorithms. McGraw-Hill. Section 2.2 (the master theorem in the a, b, d form).

[3] Karatsuba, A. & Ofman, Y. (1962). Multiplication of many-digital numbers by automatic computers. Proceedings of the USSR Academy of Sciences, 145, 293–294.

[4] Strassen, V. (1969). Gaussian elimination is not optimal. Numerische Mathematik, 13(4), 354–356.