# Master Theorem – What to Remember

## ✅ 1. The Standard Form

Check if the recurrence looks like:  
T(n) = aT(n/b) + f(n)  
Where:  
- a = number of subproblems  
- b = size reduction factor  
- f(n) = cost of work outside recursion

## ✅ 2. Calculate n^(log\_b a)

Use:  
n^(log\_b a) = n^(log a / log b)  
(log can be log base 10 or natural log — ratio remains the same.)

## ✅ 3. Compare f(n) with n^(log\_b a)

- If f(n) grows slower → Case 1  
- If f(n) grows same → Case 2  
- If f(n) grows faster → Case 3 (check regularity condition)

## ✅ 4. The 3 Cases

Case 1: f(n) = O(n^(log\_b a - ε)) → Θ(n^(log\_b a))  
Case 2: f(n) = Θ(n^(log\_b a)(log n)^k) → Θ(n^(log\_b a)(log n)^(k+1))  
Case 3: f(n) = Ω(n^(log\_b a + ε)) and regularity holds → Θ(f(n))

## ✅ 5. Regularity Condition for Case 3

Check:  
a \* f(n/b) ≤ c \* f(n)  
for some constant c < 1 and large n.

## ✅ 6. When Master Theorem Cannot Be Used

- Subproblem sizes are different (e.g., T(n) = T(n/2) + T(n/3) + n)  
- a or b is not constant  
- f(n) is irregular  
- Recurrence is not standard form

## ✅ 7. Common Recurrences and Results

T(n) = 2T(n/2) + n → Θ(n log n) (Merge Sort)  
T(n) = 2T(n/2) + 1 → Θ(n)  
T(n) = T(n/2) + 1 → Θ(log n) (Binary Search)  
T(n) = 4T(n/2) + n → Θ(n^2)  
T(n) = 8T(n/2) + n^2 → Θ(n^3)

## ✅ 8. Shortcut Tip

If a = b^k and f(n) = n^k, then:  
- If f(n) matches n^(log\_b a), answer is n^k log n (Case 2).

# 📌 Quick Memory Summary

Form: T(n) = aT(n/b) + f(n)  
Step 1: Calculate n^(log\_b a)  
Step 2: Compare with f(n)  
Step 3: Pick Case 1, 2, or 3  
Step 4: Apply time complexity rule  
Step 5: Check if Master Theorem is allowed







