**Master Theorem**

In this tutorial, you will learn what master theorem is and how it is used for solving recurrence relations.

The master method is a formula for solving recurrence relations of the form:

T(n) = aT(n/b) + f(n),

where,

n = size of input

a = number of subproblems in the recursion

n/b = size of each subproblem. All subproblems are assumed

to have the same size.

f(n) = cost of the work done outside the recursive call,

which includes the cost of dividing the problem and

cost of merging the solutions

Here, a ≥ 1 and b > 1 are constants, and f(n) is an asymptotically positive function.

An asymptotically positive function means that for a sufficiently large value of n, we have f(n) > 0.

The master theorem is used in calculating the time complexity of recurrence relations ([divide and conquer algorithms](https://www.programiz.com/dsa/divide-and-conquer)) in a simple and quick way.

**Master Theorem**

If a ≥ 1 and b > 1 are constants and f(n) is an asymptotically positive function, then the time complexity of a recursive relation is given by

T(n) = aT(n/b) + f(n)

where, T(n) has the following asymptotic bounds:

1. If f(n) = O(nlogb a-ϵ), then T(n) = Θ(nlogb a).

2. If f(n) = Θ(nlogb a), then T(n) = Θ(nlogb a \* log n).

3. If f(n) = Ω(nlogb a+ϵ), then T(n) = Θ(f(n)).

ϵ > 0 is a constant.

Ad

Each of the above conditions can be interpreted as:

1. If the cost of solving the sub-problems at each level increases by a certain factor, the value of f(n) will become polynomially smaller than nlogb a. Thus, the time complexity is oppressed by the cost of the last level ie. nlogb a
2. If the cost of solving the sub-problem at each level is nearly equal, then the value of f(n) will be nlogb a. Thus, the time complexity will be f(n) times the total number of levels ie. nlogb a \* log n
3. If the cost of solving the subproblems at each level decreases by a certain factor, the value of f(n) will become polynomially larger than nlogb a. Thus, the time complexity is oppressed by the cost of f(n).

**Solved Example of Master Theorem**

T(n) = 3T(n/2) + n2

Here,

a = 3

n/b = n/2

f(n) = n2

logb a = log2 3 ≈ 1.58 < 2

ie. f(n) < nlogb a+ϵ , where, ϵ is a constant.

Case 3 implies here.

Thus, T(n) = f(n) = Θ(n2)

**Master Theorem Limitations**

The master theorem cannot be used if:

* T(n) is not monotone. eg. T(n) = sin n
* f(n) is not a polynomial. eg. f(n) = 2n
* a is not a constant. eg. a = 2n
* a < 1