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) 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 $f(n) = O(n^{\log_b a-\epsilon})$, then $T(n) = \Theta(n^{\log_b a})$.

2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} * \log n)$.

3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$, then $T(n) = \Theta(f(n))$.

$\epsilon > 0$ is a constant.

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 $n^{\log_b a}$. Thus, the time complexity is oppressed by the cost of the last level ie. $n^{\log_b a}$
2. If the cost of solving the sub-problem at each level is nearly equal, then the value of `f(n)` will be $n^{\log_b a}$. Thus, the time complexity will be `f(n)` times the total number of levels ie. $n^{\log_b a} * \log n$
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 $n^{\log_b a}$. 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

log_b a = log_2 3 ≈ 1.58 < 2

ie. f(n) < n^(log_b 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) = 2^n`
- `a` is not a constant. eg. `a = 2n`
- `a < 1`