Master's Theorem
(Reading Material)

Master's Method or Master's Theorem is a direct way to get the solution for a recursive code's Time Complexity. Master's theorem works only for the following type of recurrences or for recurrences that can be transformed into the following type :

$T(n) = aT(n/b) + f(n)$ where $a >= 1$ and $b > 1$

There are the following three cases:

- If $f(n) = O(n^c)$ where $c < Log_b a$ then $T(n) = \Theta(n * Log_b a)$
- If $f(n) = \Theta(n^c)$ where $c = Log_b a$ then $T(n) = \Theta(n^c * Log\ n)$
- If $f(n) = \Omega(n^c)$ where $c > Log_b a$ then $T(n) = \Theta(f(n))$

How does this work?

This is how a recursive tree for such problems will be like :

```
              T(n)
             / | \
            /  |  \
      T(n/b)  T(n/b)  ... T(n/b)
            |     |
            .     .
            .     .
            .     .
         T(n/b^2) T(n/b^2)
           |     |
           .     .
           .     .
           .     .

        ... (recursive calls) ...
```

In the recurrence tree method, we calculate the total work done.
Case 1 : If the work done at leaves is polynomially more, then leaves are the dominant part, and our result becomes the work done at leaves

Case 2 : If work done at leaves and root is asymptotically the same, then our result becomes height multiplied by work done at any level

Case 3 : If work done at the root is asymptotically more, then our result becomes work done at the root

Examples of some standard algorithms whose time complexity can be evaluated using the Master's Theorem :

- **Merge Sort**: $T(n) = 2T(n/2) + \Theta(n)$. It falls in case 2 as c is 1 and $\text{Log}_b a$ is also 1. So the solution is $\Theta(n * \text{Logn})$

- **Binary Search**: $T(n) = T(n/2) + \Theta(1)$. It also falls in case 2 as c is 0 and $\text{Log}_b a$ is also 0. So the solution is $\Theta(\text{Logn})$