Recurrences: Master Method

Instructor: Dr. Tarunpreet Bhatia
Assistant Professor, CSED
Thapar Institute of Engineering and Technology

The Master Theorem

- Given: a *divide and conquer* algorithm
  - An algorithm that divides the problem of size n into a subproblems, each of size n/b
  - Let the cost of each stage (i.e., the work to divide the problem + combine solved subproblems) be described by the function $f(n)$
- Then, the Master Theorem gives us a cookbook for the algorithm’s running time.
- We can apply Master's Theorem for Dividing functions and Decreasing Functions

Master Theorem

- Suppose you have a recurrence of the form

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

Time to divide & recombine

Number of sub-problems

Time required to solve a sub-problem

- **This recurrence would arise in the analysis of a recursive algorithm.**
- When input size $\boldsymbol{n}$ is large, the problem is divided up into $\boldsymbol{a}$ sub-problems each of size $\boldsymbol{n/b}$. Sub-problems are solved recursively and results are recombined.
- The work to split the problem into sub-problems and recombine the results is $\boldsymbol{f(n)}$.

Master Theorem for Dividing Function

- if T(n) = aT(n/b) + f(n) then

$$T(n)=\left\{\begin{array}{ll}\Theta\left(n^{\log_b a}\right) & f(n)=O\left(n^{\log_b a-\varepsilon}\right)\\ \Theta\left(n^{\log_b a}\log n\right) & f(n)=\Theta\left(n^{\log_b a}\right)\\ \Theta\left(f(n)\right) & f(n)=\Omega\left(n^{\log_b a+\varepsilon}\right)\text{ AND}\\ & af(n/b)<cf(n)\ \text{ for large } n\end{array}\right\}\begin{array}{l}\varepsilon>0\\ c<1\end{array}$$

Extension of Master Theorem for Dividing Function

$$T(n) = aT(n/b) + \theta(n^k \log^p n)$$

1) If $a > b^k$, then $T(n) = \Theta\left(n^{\log_b^a}\right)$
2) If $a = b^k$
   a. If $p > -1$, then $T(n) = \Theta\left(n^{\log_b^a} \log^{p+1} n\right)$
   b. If $p = -1$, then $T(n) = \Theta\left(n^{\log_b^a} \log\log n\right)$
   c. If $p < -1$, then $T(n) = \Theta\left(n^{\log_b^a}\right)$
3) If $a < b^k$
   a. If $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$
   b. If $p < 0$, then $T(n) = O(n^k)$

Examples

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

$T(n) = aT(n/b) + \theta(n^k \log^p n)$

1) If $a > b^k$, then $T(n) = \Theta\left(n^{\log_b^a}\right)$
2) If $a = b^k$
   a. If $p > -1$, then $T(n) = \Theta\left(n^{\log_b^a} \log^{p+1} n\right)$
   b. If $p = -1$, then $T(n) = \Theta\left(n^{\log_b^a} \log\log n\right)$
   c. If $p < -1$, then $T(n) = \Theta\left(n^{\log_b^a}\right)$
3) If $a < b^k$
   a. If $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$
   b. If $p < 0$, then $T(n) = O(n^k)$

Examples

$T(n) = 2T(n/2) + n$

$T(n) = aT(n/b) + \theta(n^k \log^p n)$

1) If $a > b^k$, then $T(n) = \Theta\left(n^{\log_b^a}\right)$
2) If $a = b^k$
   a. If $p > -1$, then $T(n) = \Theta\left(n^{\log_b^a} \log^{p+1} n\right)$
   b. If $p = -1$, then $T(n) = \Theta\left(n^{\log_b^a} \log\log n\right)$
   c. If $p < -1$, then $T(n) = \Theta\left(n^{\log_b^a}\right)$
3) If $a < b^k$
   a. If $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$
   b. If $p < 0$, then $T(n) = O(n^k)$

Examples

$$T(n) = 2T(n/2) + n \log n$$

$$T(n) = aT(n/b) + \theta(n^k \log^p n)$$

1) If $a > b^k$, then $T(n) = \Theta\left(n^{\log_b^a}\right)$
2) If $a = b^k$
   a. If $p > -1$, then $T(n) = \Theta\left(n^{\log_b^a} \log^{p+1} n\right)$
   b. If $p = -1$, then $T(n) = \Theta\left(n^{\log_b^a} \log\log n\right)$
   c. If $p < -1$, then $T(n) = \Theta\left(n^{\log_b^a}\right)$
3) If $a < b^k$
   a. If $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$
   b. If $p < 0$, then $T(n) = O(n^k)$

Examples

$T(n) = 3T(n/2) + n^2$

$T(n) = aT(n/b) + \theta(n^k \log^p n)$

1) If $a > b^k$, then $T(n) = \Theta\left(n^{\log_b^a}\right)$
2) If $a = b^k$
   a. If $p > -1$, then $T(n) = \Theta\left(n^{\log_b^a} \log^{p+1} n\right)$
   b. If $p = -1$, then $T(n) = \Theta\left(n^{\log_b^a} \log\log n\right)$
   c. If $p < -1$, then $T(n) = \Theta\left(n^{\log_b^a}\right)$
3) If $a < b^k$
   a. If $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$
   b. If $p < 0$, then $T(n) = O(n^k)$

Examples

$$\boldsymbol{T(n) = 2T(n/2) + 1}$$

Given, a = 2, b = 2,

f(n) = $\theta(1)$

$= \theta(n^0 (\log n)^0)$

$k = 0, p = 0$

$log_2^2 = 1 > k = 0$

Case1 :$\theta(n^{log_2^2}) = \theta(n^1)$

Examples

$$\boldsymbol{T(n) = 4T(n/2) + n}$$

$log_2^4 = 2 > k = 1$, *p = 0*

therefore case 1: $\theta(n^2)$

Master Theorem for dividing function

<u>Case 1:</u>

$T(n) = 2T(n/2) + 1$	$O(n)$
$T(n) = 4T(n/2) + 1$	$O(n^2)$
$T(n) = 4T(n/2) + n$	$O(n^2)$
$T(n) = 8T(n/2) + n^2$	$O(n^3)$
$T(n) = 16T(n/2) + n^2$	$O(n^4)$

Case 2:

$T(n) = T(n/2) + 1$ O($logn$)

$T(n) = 2T(n/2) + n$ O($nlogn$)

Solve for:

$T(n) = 2T(n/2) + nlogn$

Case 3:

$T(n) = T(n/2) + n$ $O(n)$

$T(n) = 2T(n/2) + n^2$ $O(n^2)$

$T(n) = 2T(n/2) + n^2\, logn$ $O(n^2\, logn)$

Try this!!

1. $T(n) = \sqrt{2}T(n/2) + \log n$
2. $T(n) = 8T(n/4) - n^2\log n$
3. $T(n) = 3T(n/3) + n/2$
4. $T(n) = T(\sqrt{n}) + 1$
5. $T(n) = 2T(n/4) + n^{0.51}$
6. $T(n) = 8T(n/2) + n$
7. $T(n) = 3T(n/4) + n\log n$

Try this!!

1. $T(n) = 4T(n/2) + n^2$
2. $T(n) = 4T(n/2) + n^2 \log n$
3. $T(n) = T(2n/3) + 1$
4. $T(n) = 9T(n/3) + n$

Master Theorem For Subtract and Conquer/Decreasing Recurrences

Let T(n) be a function defined on positive n as shown below:

$$T(n) \leq \begin{cases} c, & if\ n \leq 1, \\ aT(n-b) + f(n), & n > 1, \end{cases} ,$$

For some constants c, a>0, b>0, k>=0 and function f(n). If f(n) is $O(n^k)$, then

1. If a<1 then $T(n) = O(n^k)$
2. If a=1 then $T(n) = O(n^{k+1})$
3. If a>1 then $T(n) = O(n^k a^{n/b})$

Examples

$$T(n) = T(n-1) + 1$$

For some constants c, a>0, b>0, k>=0 and function f(n). If f(n) is $O(n^k)$, then

1. If a<1 then $T(n) = O(n^k)$
2. If a=1 then $T(n) = O(n^{k+1})$
3. If a>1 then $T(n) = O(n^k a^{n/b})$

Examples

$$T(n) = T(n-2) + 1$$

For some constants c, a>0, b>0, k>=0 and function f(n). If f(n) is $O(n^k)$, then

1. If a<1 then $T(n) = O(n^k)$
2. If a=1 then $T(n) = O(n^{k+1})$
3. If a>1 then $T(n) = O(n^k a^{n/b})$

Examples

$$T(n) = T(n-1) + n^2$$

For some constants c, a>0, b>0, k>=0 and function f(n). If f(n) is $O(n^k)$, then

1. If a<1 then $T(n) = O(n^k)$
2. If a=1 then $T(n) = O(n^{k+1})$
3. If a>1 then $T(n) = O(n^k a^{n/b})$

Example

```
int fib(int n)
{
    if (n <= 1)
            return n;
    return fib(n-1) + fib(n-2);
}
```

Recursive Equation

$T(n) = T(n-1) + T(n-2)$

- For Worst Case, Let $T(n-1) \approx T(n-2)$
  $T(n) = 2T(n-1) + c$
  where, $f(n) = O(1)$
  $\therefore k=0, a=2, b=1;$
  $T(n) = O(n^0 2^{n/1})$
  $= O(2^n)$

- For Best Case, Let $T(n-2) \approx T(n-1)$
  $T(n) = 2T(n-2) + c$
  where, $f(n) = O(1)$
  $\therefore k=0, a=2, b=2;$
  $T(n) = O(n^0 2^{n/2})$
  $= O(2^{n/2})$

Master Theorem for Dividing Function Limitations

The master theorem cannot be used if:

1. $T(n)$ is not monotone. eg. $T(n) = \sin n$
2. a is not a constant. eg. $a = 2n$
3. Can't have less than 1 subproblem $a < 1$
4. $f(n)$ which is a combination time is not positive

Query??

- $T(n) = 2T(n/2) + n \log n$
- $T(n) = 2T(n/2) + n / \log n$

Can you solve using Extended Master's Theorem? Justify your answer