Master Theorem for Divide and Conquer Recurrences

According to master theorem the runtime of the above algorithm can be expressed as:

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

where n = size of the problem
a = number of subproblems in the recursion and a >= 1
n/b = size of each subproblem
f(n) = cost of work done outside the recursive calls like dividing into subproblems and cost of combining them to get the solution.

Not all recurrence relations can be solved with the use of the master theorem i.e. if

T(n) is not monotone, ex: T(n) = sin n
f(n) is not a polynomial, ex: $T(n) = 2T(n/2) + 2^n$

This theorem is an advance version of master theorem that can be used to determine running time of divide and conquer algorithms if the recurrence is of the following form :-

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

where n = size of the problem
a = number of subproblems in the recursion and a >= 1
n/b = size of each subproblem
b > 1, k >= 0 and p is a real number.

Then,

Case 1: if $a > b^k$, then $T(n) = \theta(n^{\log_b a})$

Case 2: if $a = b^k$, then
(a) if p > -1, then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$
(b) if p = -1, then $T(n) = \theta(n^{\log_b a} \log\log n)$
(c) if p < -1, then $T(n) = \theta(n^{\log_b a})$

Case 3: if $a < b^k$, then
(a) if p >= 0, then $T(n) = \theta(n^k \log^p n)$
(b) if p < 0, then $T(n) = \theta(n^k)$

Examples

Example 1:
Binary Search – $T(n) = T(n/2) + O(1)$
$a = 1, b = 2, k = 0$ and $p = 0$
$b^k = 1$. So, $a = b^k$ and $p > -1$ [Case 2.(a)]
$T(n) = \theta(n^{\log_b a} \log^{p+1} n)$
$T(n) = \theta(\log n)$

Example 2:
Merge Sort – $T(n) = 2T(n/2) + O(n)$
$a = 2, b = 2, k = 1, p = 0$
$b^k = 2$. So, $a = b^k$ and $p > -1$ [Case 2.(a)]
$T(n) = \theta(n^{\log_b a} \log^{p+1} n)$
$T(n) = \theta(n \log n)$

Example 3:
$T(n) = 3T(n/2) + n^2$
$a = 3, b = 2, k = 2, p = 0$
$b^k = 4$. So, $a < b^k$ and $p = 0$ [Case 3.(a)]
$T(n) = \theta(n^k \log^p n)$
$T(n) = \theta(n^2)$

Example 4:
$T(n) = 3T(n/2) + \log^2 n$
$a = 3, b = 2, k = 0, p = 2$
$b^k = 1$. So, $a > b^k$ [Case 1]
$T(n) = \theta(n^{\log_b a})$
$T(n) = \theta(n^{\log_2 3})$

Example 5:
$T(n) = 2T(n/2) + n\log^2 n$
$a = 2, b = 2, k = 1, p = 2$
$b^k = 2$. So, $a = b^k$ [Case 2.(a)]
$T(n) = \theta(n^{\log_b a} \log^{p+1} n)$
$T(n) = \theta(n^{\log_2 2} \log^3 n)$
$T(n) = \theta(n \log^3 n)$

Example 6:
$T(n) = 2^n T(n/2) + n^n$
This recurrence can't be solved using above method since function is not of form $T(n) = aT(n/b) + \theta(n^k \log^p n)$

Master Theorem for Subtract and Conquer Recurrences

Master Theorem For Subtract and Conquer Recurrences:
Let T(n) be a function defined on positive n as shown below:

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

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

Example 1:
T(n) = 3T(n-1), n>0
= c, n<=0
Sol:a=3, b=1, f(n)=0 so k=0;
Since a>0, $T(n) = O(n^k a^{n/b})$
$T(n)= O(n^0 3^{n/1})$
$T(n)= 3^n$

Example 2:
T(n) = T(n-1) + n(n-1), if n>=2
= 1, if n=1
Sol:a=1, b=1, f(n)=n(n-1) so k=2;
Since a=1, $T(n) = O(n^{k+1})$
$T(n)= O(n^{2+1})$
$T(n)= O(n^3)$

Example 3:
T(n) = 2T(n-1) – 1, if n>0
= 1, if n<=0
Sol: This recurrence can’t be solved using above method
since function is not of form T(n) = aT(n-b) + f(n)