Data Structures and Algorithm Analysis

16

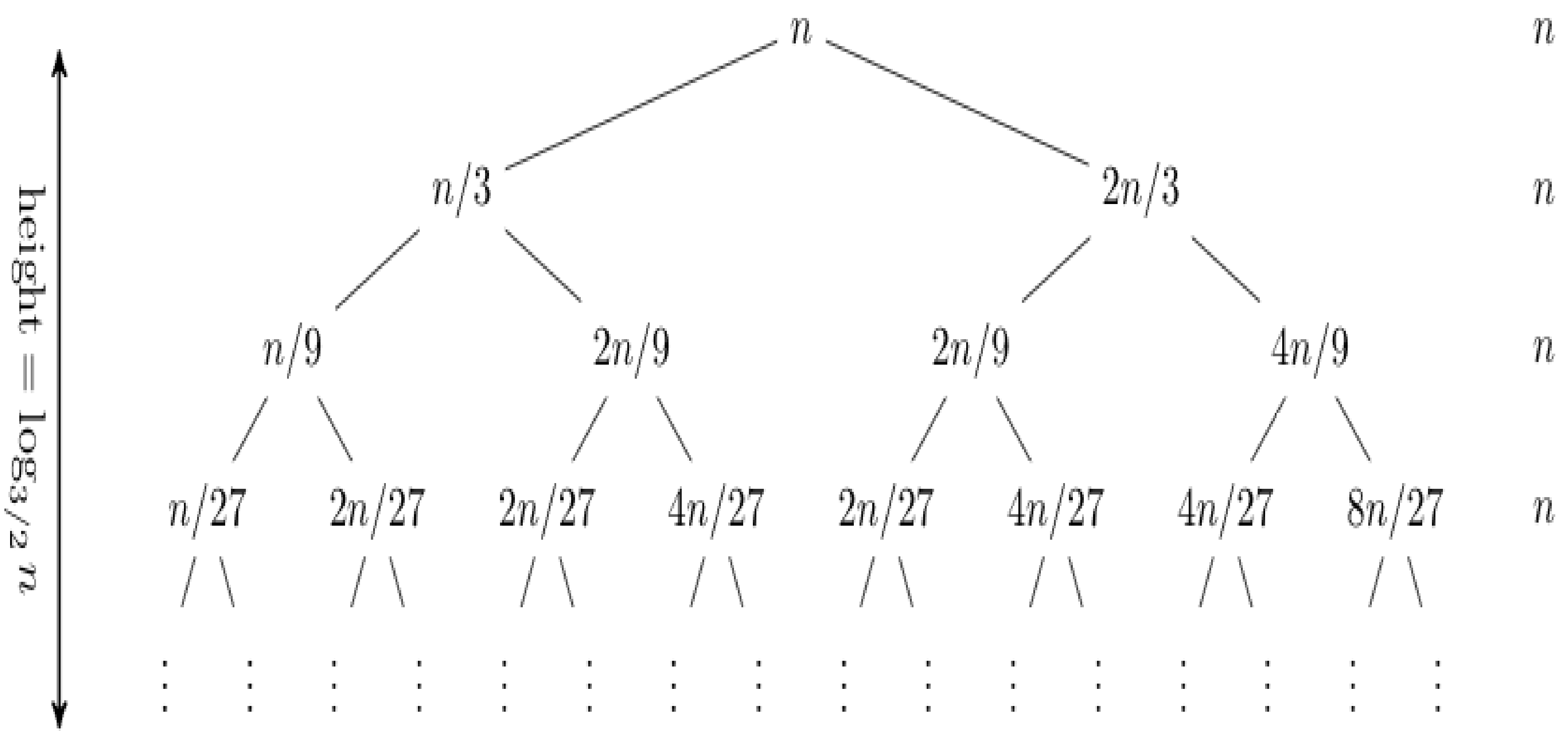
Dr. Syed Asim Jalal
Department of Computer Science
University of Peshawar

Solving Recurrence Using Merge-Sort as an Example

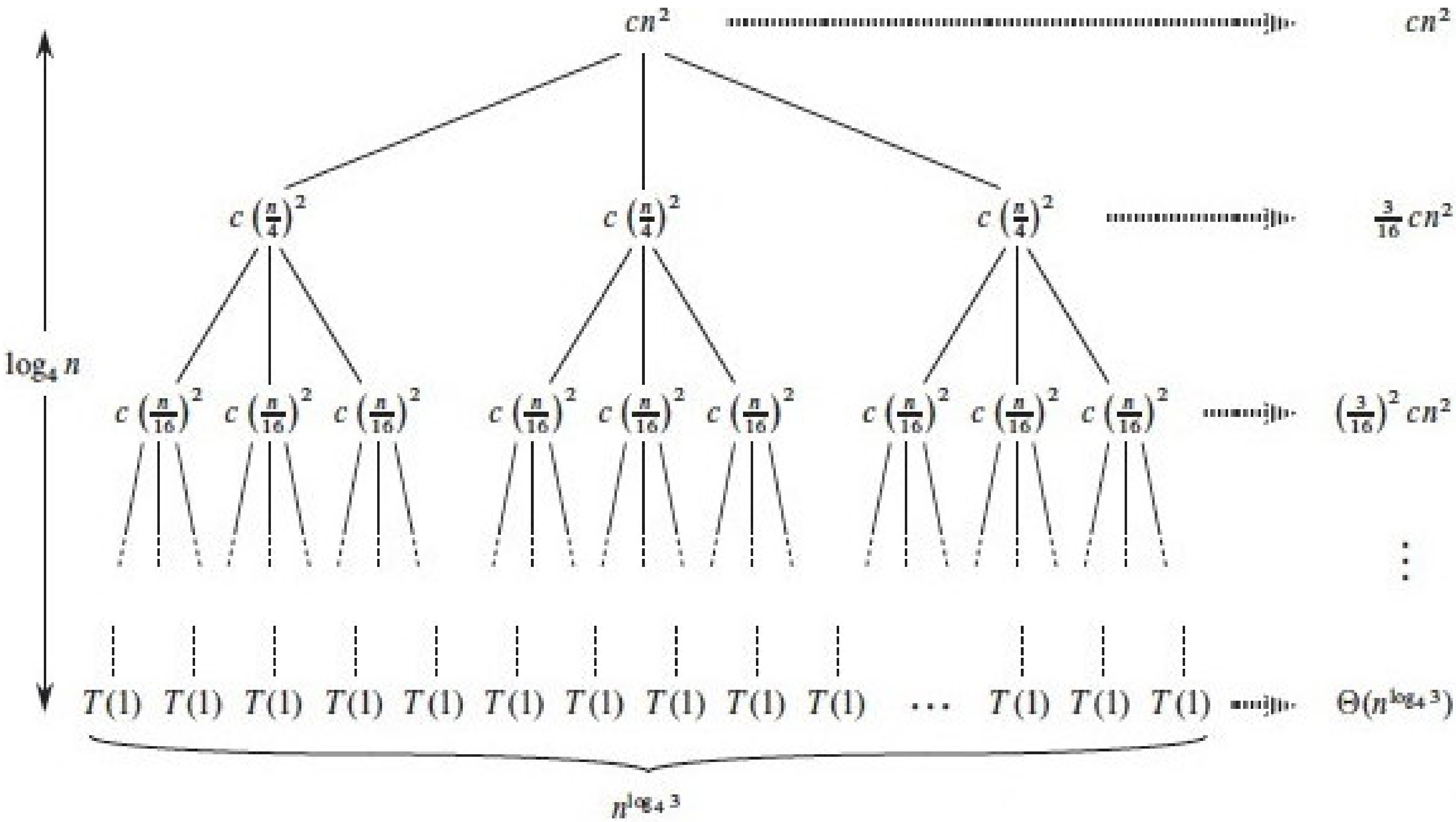
Recursion Tree Method

- We can describe any recurrence in terms of a tree, where each expansion of the recurrence takes us one level deeper in the tree.
- We can then sum running time at each level.

Sample/Example only



Sample/Example only

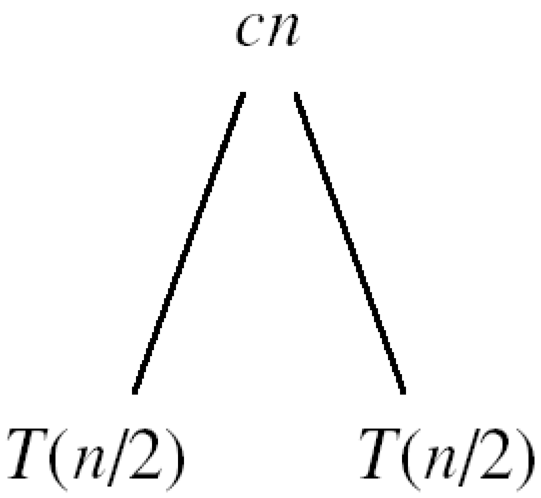


$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 . \end{cases}$$

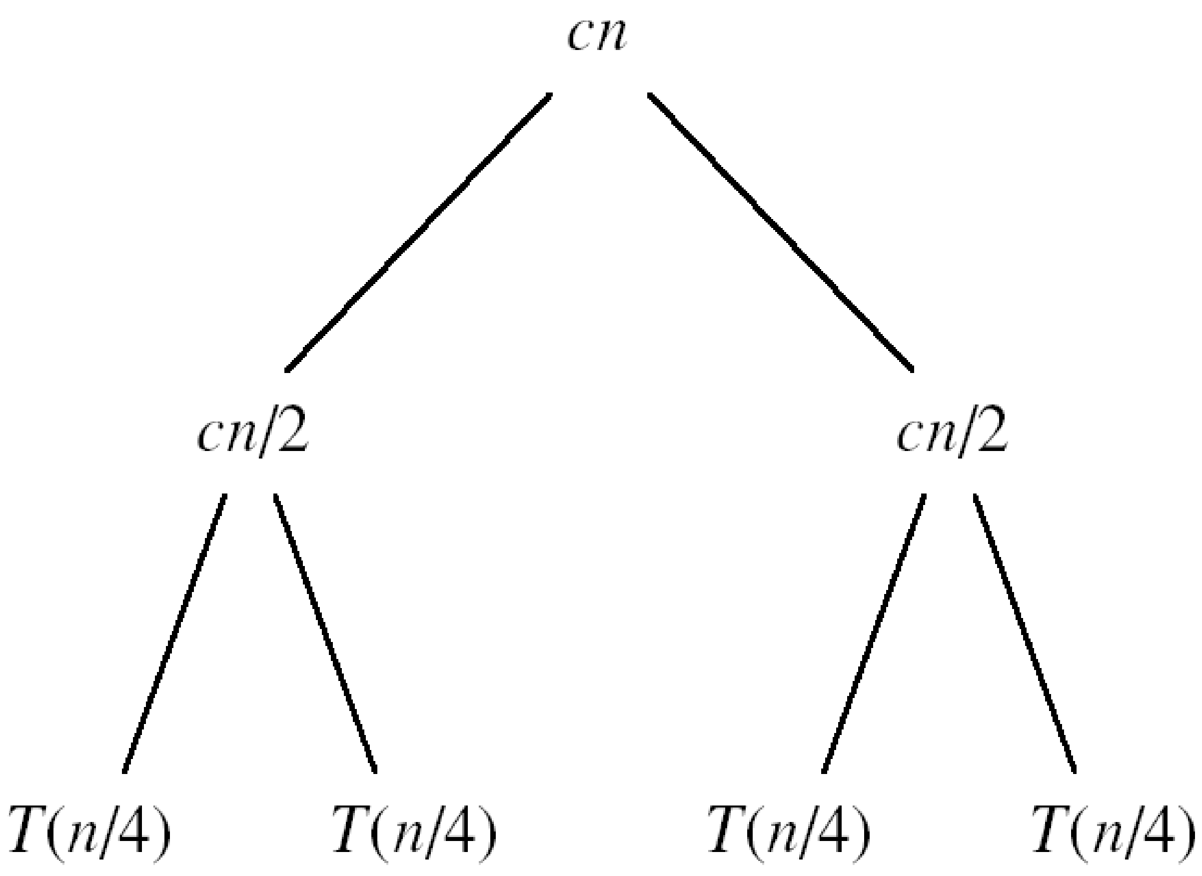
Recursion Tree Method

- As we derived earlier
  - Time to combine solutions be ***C(n)***
  - In divide and conquer, **'*a*'** sub-problems takes *aT(n/b)* time. For merge Sort a =2 and b=2, therefore, Solving sub-problems takes ***2T (n/2)***
  - *c* is the constant time to solve *base case in merge sort*

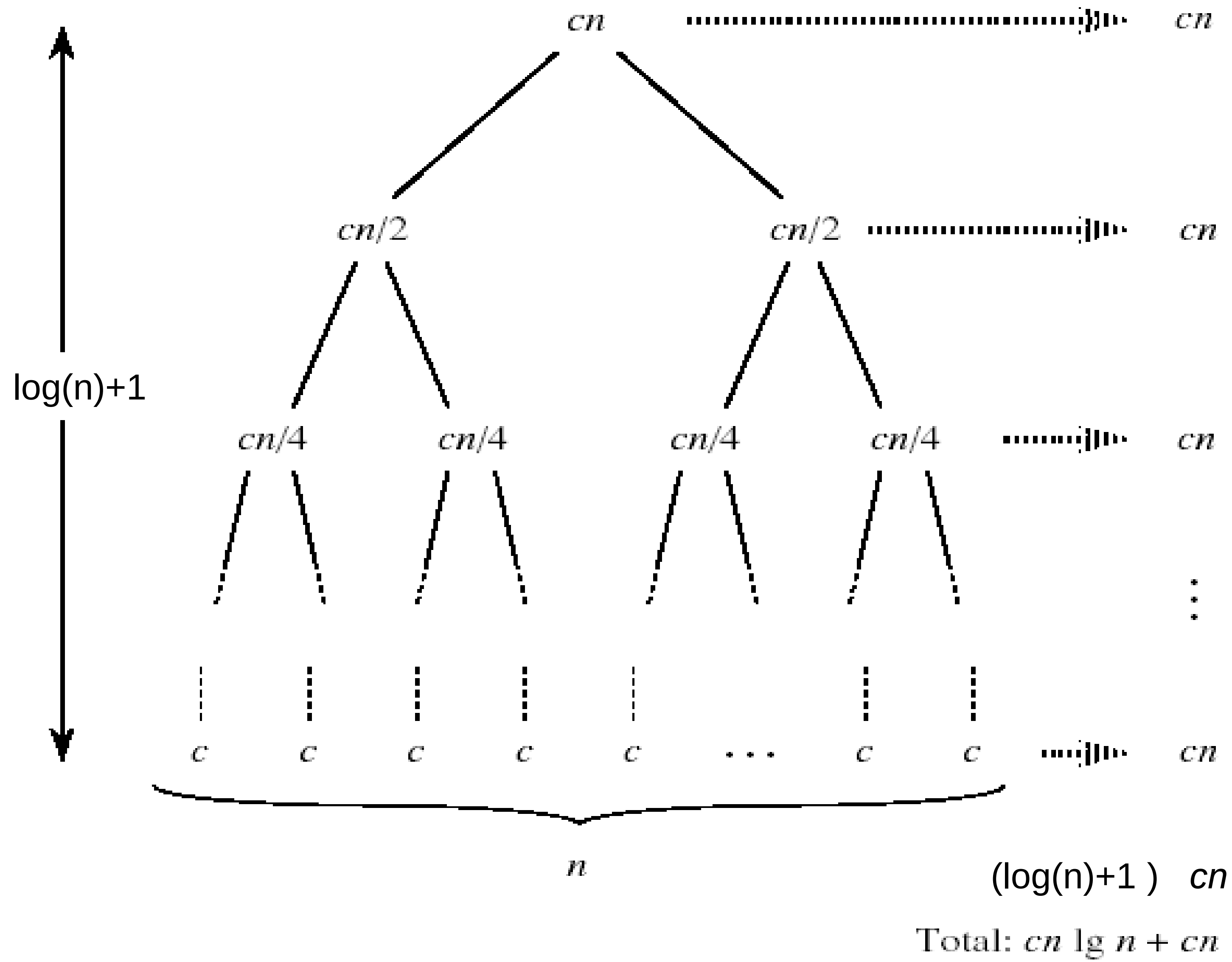
- For the original problem, we have a cost of ***cn***, plus the two sub-problems, each costing *T (n/2)*:



- For each of the size-*n*/2 subproblems, we have a cost of *cn*/2, plus two subproblems, each costing *T (n/4)*:



- Continue expanding until the problem sizes get down to 1:



Basic concept is to find the costs involved at each level and add them all.

- Each level has cost ***cn***.
  - The top level has cost *cn*.
  - The next level down has 2 sub-problems, each contributing cost *cn*/2.
  - The next level has 4 sub-problems, each contributing cost *cn*/4.
  - Each time we go down one level, the number of sub-problems doubles but the cost per sub-problem halves
  - However total cost per each level stays the same.

- If we observe the pattern, we see that there are $\log n + 1$ levels (height is $\log n$).
- Total cost is

  $$(\log n + 1) * cn = cn \log n + cn$$
- Ignoring c's and lower order terms, we get nlog(n)
- So the running time of MERGE SORT is
  - $\Theta(n \log n)$

The Master Method

The Master Method

- It is a “cookbook” for the solving many divide-and-conquer recurrences of the form

$$T(n) = \begin{cases} \Theta(1) \\ aT(n/b) + f(n) \end{cases},$$

where $a \geq 1$, $b > 1$, and $f(n) > 0$.

- *f(n)* is *D(n) + C(n) as given in the figure below.*

$$T(n) = \begin{cases} \Theta(1) \\ aT(n/b) + \overbrace{D(n) + C(n)}^{f(n)} \end{cases}$$

Example of f(n) in merge sort

- Recurrence relation of *merge sort* is

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1\,. \end{cases}$$

- *f(n) = n, because in merge sort D(n) + C(n) = Θ(n),*
  - *Also in merge sort a = 2, b = 2*

Master Method: Three cases

- Master Method has three cases
  - You have to check which case your recurrence relation falls in and then apply the corresponding solution.
  - For Master Method, you need knowledge of asymptotic notations (Θ,O,Ω) and *Log*.

Master Method: Case 1

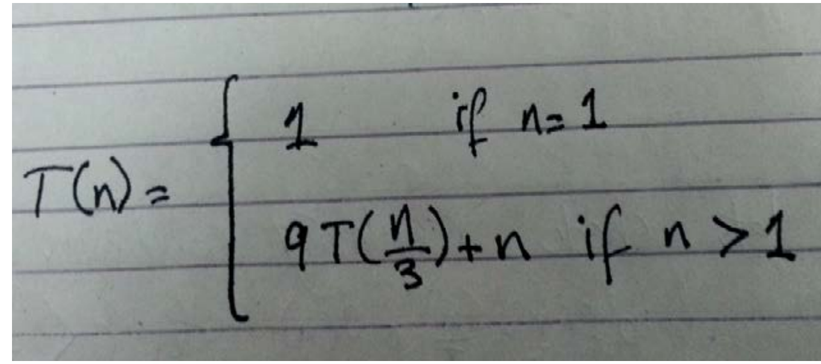
- If

$$f(n) = O(n^{\log_b a - \epsilon}) \text{ for some constant } \epsilon > 0.$$

- Then

$$T(n) = \Theta(n^{\log_b a}).$$

Example of Case 1:



- Suppose, we have:

  $$T(n) = 9T(n/3) + n.$$

  - i.e. a = 9, b = 3, f (n) = n
  - *Then check if*
  - $f(n) = O(n^{\log_3 9 - \varepsilon})$?

    ➔ $n = O(n^{2-\varepsilon})$, as $\log_3 9 = 2$

    ➔ for $\varepsilon = 1$, $n = O(n)$, *which is true*

Therefore, according to Case 1 the solution is

$T = \Theta(n^2)$

Master Method: Case 2

- If

$$f(n) = \Theta(n^{\log_b a})$$

- Then

$$T(n) = \Theta(n^{\log_b a} \lg n)$$

Example of Case 2 *(Merge Sort case)*

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 . \end{cases}$$

- For case 2 example, we consider MergeSort where we have, **$T(n) = 2T(n/2) + n$**

  i.e. $a=2$, $b=2$, $f(n) = n$

- Now check if

  $$f(n) = \Theta(n^{\log_2 2}).$$

  ➔ **$n = \Theta(n)$,** which is true **$c_1=1/2$, $c_2=2$, $n_0=1$**

- **Therefore according to case 2 the solution is:**

  $T(n) = \Theta(n \log n)$

Master Method: Case 3

- If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and *f (n)* satisfies the *regularity condition* $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large *n*.
- Then

$$T(n) = \Theta(f(n))$$

Regularity condition always holds whenever $f(n) = n^k$

Example: Case3

- $T(n) = 4T(n/2) + n^3$.
  - $a=4$, $b=2$, $f(n) = n^3$

- Here, check the following 2 things:

1. Regularity condition holds if

$$af(n/b) \leq cf(n)$$

2. if $n^3 = \Omega(n^{\log_2 4 + \varepsilon})$

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3$$

Here $a = 4$, $b = 2$, $f(n) = n^3$

first Lets check if

$$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right) \text{ for } \epsilon > 0$$

$$\Rightarrow n^3 = \Omega\left(n^{\log_2 4 + \epsilon}\right)$$

$$\Rightarrow n^3 = \Omega\left(n^{2 + \epsilon}\right)$$

Now, if we take $\epsilon = 1$

$$\Rightarrow n^3 = \Omega\left(n^{2+1}\right)$$

$$\Rightarrow n^3 = \Omega(n^3),$$ which is true.

Now, lets check regularity condition.

i-e $a f(\frac{n}{b}) \le c f(n)$ for $c < 1$

$$\Rightarrow 4\left(\frac{n}{2}\right)^3 \le c n^3$$

$$\Rightarrow 4 \frac{n^3}{8} \le c n^3$$

$$\Rightarrow \frac{n^3}{2} \le c n^3$$ it is TRUE $\boxed{c = \frac{1}{2}}$

Both checks satisfied, therefore Solution is

$$T(n) = \Theta(f(n))$$

$$\Rightarrow \boxed{T(n) = \Theta(n^3)}$$

Iteration Method???

Basics: Expand terms and look for pattern

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

T(1) = 1

T(2) = T(1) + T(1) + 2 = 1 + 1 + 2 = 4

T(3) = T(2) + T(1) + 3 = 4 + 1 + 3 = 8

T(4) = T(2) + T(2) + 4 = 4 + 4 + 4 = 12

T(5) = T(3) + T(2) + 5 = 8 + 4 + 5 = 17

. . .

T(8) = T(4) + T(4) + 8 = 12 + 12 + 8 = 32

. . .

T(16) = T(8) + T(8) + 16 = 32 + 32 + 16 = 80

What is the pattern here ?

T(1)	1
T(2)	4
T(4)	12
T(8)	32
T(16)	80

- To understand the pattern, let's consider the ratios T(n)/n for powers of 2:

T(1) / 1	1	0 + 1	log(1) + 1
T(2) /2	2	1 + 1	log(2) + 1
T(4) /4	3	2 + 1	log(4) + 1
T(8) / 8	4	3 + 1	log(8) + 1
T(16) / 16	5	4 + 1	log(16) + 1
T(n) / n			log(n) + 1

- This suggests, $T(n) = (\log n + 1)*n$, or $T(n) = n \log n + n$
  which is $T(n) = \Theta(n \log n)$

Another way!!!

- The iteration method turns the recurrence into a summation. Let's expand the recurrence:

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

$= 2(2T(n/4) + n/2) + n$

$= 4T(n/4) + n + n$

$= 4(2T(n/8) + n/4) + n + n$

$= 8T(n/8) + n + n + n$

$= 8(2T(n/16) + n/8) + n + n + n$

$= 16T(n/16) + n + n + n + n$

...

The pattern is: $\mathbf{T(n) = 2^x\ T(n / 2^x) + x * n}$

Now, let 'n' is some power of 2, i.e. $n = 2^k$, we get

$T(n) = 2^k\ T(n / 2^k) + k * n$

➔ $T(n) = 2^k\ T(n / n) + k * n$

➔ $T(n) = 2^k\ T(1) + k * n$

➔ $T(n) = 2^k\ (1) + k * n$

➔ $T(n) = 2^k + k * n$

➔ $T(n) = n + k * n$

If $n = 2^k$, then $k = \log n$

We get,

➔ $T(n) = n + \log(n) * n$

➔ $T(n) = n + n(\log n)$

or

$$T(n) = \Theta(n \log n)$$

$$T(n) = 2\,T\left(\frac{n}{2}\right) + n$$

$$= 2\left(2T\left(\frac{\frac{n}{2}}{2}\right) + \frac{n}{2}\right) + n$$

$$= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

$$= 4T\left(\frac{n}{4}\right) + n + n$$

$$= 4T\left(\frac{n}{4}\right) + 2n \qquad \boxed{2^2 T\left(\frac{n}{2^2}\right) + 2n}$$

$$= 4\left(2T\left(\frac{\frac{n}{4}}{2}\right) + \frac{n}{4}\right) + 2n$$

$$= 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n$$

$$= 8T\left(\frac{n}{8}\right) + n + 2n$$

$$= 8T\left(\frac{n}{8}\right) + 3n \qquad \boxed{2^3 T\left(\frac{n}{2^3}\right) + 3n}$$

$$= 8\left(2T\left(\frac{\frac{n}{8}}{2}\right) + \frac{n}{8}\right) + 3n$$

$$= 16T\left(\frac{n}{16}\right) + 4n \qquad \boxed{2^4 T\left(\frac{n}{2^4}\right) + 4n}$$

⋮

Now, suppose $n = 2^k$

$$T(n) = 2^k$$

it is of the form

$$= 2^x T\left(\frac{n}{2^x}\right) + xn$$

suppose $n = 2^k$ then

$$= 2^k T\left(\frac{n}{2^k}\right) + kn$$

$$= 2^k T\left(\frac{n}{n}\right) + kn$$

$$= 2^k T(1) + kn$$

$$= 2^k(1) + kn$$

$$= 2^k + kn \quad , \text{as } n = 2^k.$$

$$= n + kn \quad , \text{if } n = 2^k \Rightarrow k = \log n$$

$$= n + (\log n)n$$

$$\Rightarrow T(n) = n + n\log n$$

or

$$\Rightarrow T(n) = \Theta(n \log n).$$