10/23/18

DIVIDE AND CONQUER

$\Longrightarrow$ Merge-Sort (A) $\quad |A| = n$

$T(n) = 2T(n/2) + O(n) \longrightarrow$ worst-case running

$\Downarrow$?

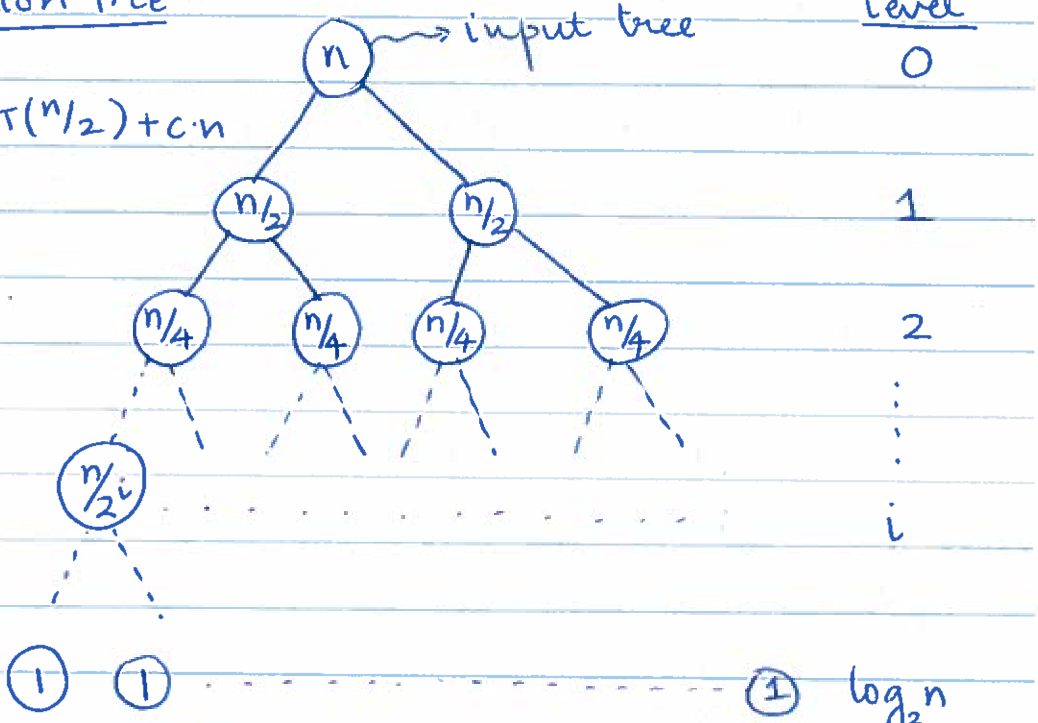
$T(n) = O(n \log n)$

Asymptotic Optimal Sorting Algorithm. (1st one actually)

Recursion Tree

$T(n) = 2T(n/2) + c\cdot n$

$T(1) = d$



Level	Work Per Node	No of nodes in layer	Total work in layer
0	$c\cdot n$	1	$c\cdot n$
1	$c\,n/2$	2	$c\cdot n$
2	$c\cdot n/4$	4	$c\cdot n$
⋮	⋮	⋮	⋮
i	$c\cdot n/2^i$	2^i	$c\cdot n$
⋮	⋮	⋮	⋮

? Why is it 'd' and not 'c'?
- Both are constants so not important
- But since constant work in last level i.e., $T(1)=d$ we use 'd'.

Total Work $= T(n) = (c \cdot n) \cdot \log_2 n + d \cdot n$
$= O(n \log n)$

- merge-sort (A)
  LS = merge-sort(left half) $\longrightarrow T(n/2)$
  RS = merge-sort (right half) $\longrightarrow T(n/2)$
  return (merge (LS, RS)); $\longrightarrow c \cdot n$

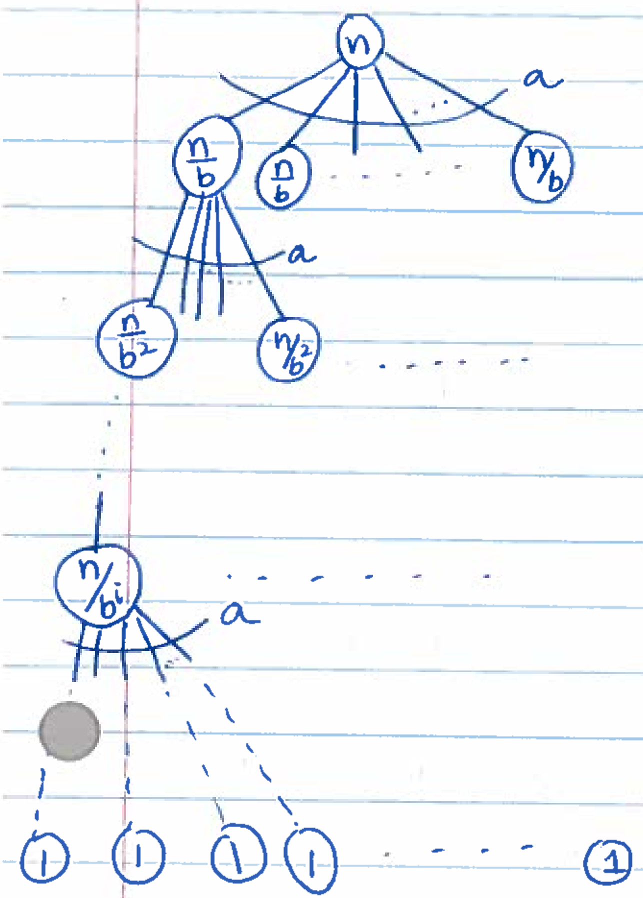
$\Rightarrow$ MASTER THEOREM: (Refer slides)

$T(n) = aT(n/b) + f(n)$
$T(1) = d$

? Why is $a \neq b$?
- Can happen (will see later on)

We will modify recursion tree for the general case.

Recursion Tree



Level	work per node	#nodes in level	[illegible]
0	$f(n)$	1	
1	$f(n/b)$	a	a
2	$f(n/b^2)$	a^2	a
⋮	⋮	⋮	
i	$f(n/b^i)$	a^i	a
⋮	⋮	⋮	
$\log_b n$	$f(\frac{n}{n}) = d$	$a^{\log_b n}$	d

Total Work

→ Case (2): Suppose $f(n) = n^{\log_b a}$ then total work in layer i:

$$a^i f(n/b^i) = a^i \left(\frac{n}{b^i}\right)^{\log_b a} = \frac{a^i n^{\log_b a}}{b^{i \log_b a}} = \frac{\cancel{a^i}\, n^{\log_b a}}{\cancel{a^i}}$$

$$= n^{\log_b a} \longrightarrow \text{Independent of 'i'!}$$

every other layer this much work

(Similar notion of same amount of work in every layer)

Total Work in last layer: $d \cdot a^{\log_b n} = d\, n^{\log_b a}$

$$\therefore\ T(n) = n^{\log_b a} \cdot \log_b n + d \cdot n^{\log_b a}$$
$$= O(n^{\log_b a} \log n)$$

? $a^{\log_b n} = n^{\log_b a}$. How???

$\rightarrow$ ~~let $x = a^{\log_b n}$~~ $a^{\log_b n} = a^{\frac{\log n}{\log b}} = a^{\frac{\log n}{\log a} \cdot \frac{\log a}{\log b}}$

$= a^{\log_a n \cdot \log_b a} = (a^{\log_a n})^{\log_b a}$

$= n^{\log_b a}$

(case 1): $f(n)$ is "polynomially smaller" than $n^{\log_b a} \Rightarrow T(n) = \Theta(n^{\log_b a})$

most work in last layer

(case 3): $f(n)$ is "polynomially larger" than $n^{\log_b a} \Rightarrow T(n) = \Theta(f(n))$

most work done at the top.

Don't get confused with other representations!

Example: Merge-sort: $T(n) = \underset{\overset{\|}{a}}{2}T(n/\underset{\overset{\|}{b}}{2}) + O(n)$

$f(n)$ vs. $n^{\log_b a}$

$O(n)$ $\quad$ n $\rightarrow$ Asymptotically equal

Case (2) $\Rightarrow T(n) = \Theta(n^{\log_b a} \cdot \log n) = \Theta(n \log n)$

→ Matrix Multiplication:

(1) $$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{O(n^2)}_{\text{time to add and form submatrix}}$$

$a = 8$, $b = 2$

$f(n)$ v.s $n^{\log_b a}$

$O(n^2)$ $\quad n^{\log_2 8} = n^3$

Naive algo do(e)s[n't] help !! ☹

(Case 1) $\Longrightarrow T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$

(with $\varepsilon = 1$)

(2) $T(n) = 8T(n/2) + O(n^2)$ is now $T(n) = 7T(n/2) + O(n^2)$

$f(n)$ v.s $n^{\log_b a}$

$O(n^2)$ $\quad n^{\log_2 7} = n^{2.81}$

(Case 1) $\Longrightarrow T(n) = \Theta(n^{2.81})$ → Success!

(with $\varepsilon = 0.8$)