S. KIMMEL

Tree Method: use trees to solve recurrence relations

max(A)

Input: Array A of length n

Output: Max value in array

```
if (length of A is 1) return A[1]
for i = 1 to n^3 : Print "Hey!"
max1 = max(1st half of A)
max2 = max(2nd half of A)
return maximum{max1, max2}
```

Recurrence Relation for Time Complexity:

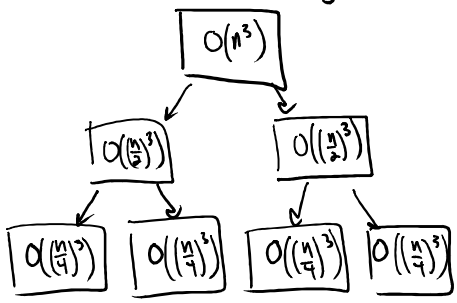
$T(n)$ = runtime on length n array

- Base case: $T(1) = O(1)$
- Recurrence: $T(n) = O(n^3) + 2T\left(\frac{n}{2}\right)$

($O(n^3)$: work done here and now; 2: # of recursive calls; $T\left(\frac{n}{2}\right)$: runtime of recursive call)

Level of Recursion

In box: amount of work done at this call not including work done by recursive calls



Level of Recursion	Size of Input	Recurrence
1	n	$T(n) = O(n^3) + 2T\left(\frac{n}{2}\right)$
2	$n/2$	$T\left(\frac{n}{2}\right) = O\left(\left(\frac{n}{2}\right)^3\right) + 2T\left(\frac{n}{4}\right)$
3	$n/4$	$T\left(\frac{n}{4}\right) = O\left(\left(\frac{n}{4}\right)^3\right) + 2T\left(\frac{n}{8}\right)$
4		
⋮		
F	1	

O(1) O(1) O(1) O(1) O(1) O(1)

Idea: count all work done in all boxes... that will be all the work.

S. KIMMEL

Tree Method (Master Method)

Can be used to solve recurrences of the form:

$$T(n) = a\,T\left(\frac{n}{b}\right) + O(n^d)$$

$$T(n) \le C \quad \text{for } n < n^*$$

a, b, d don't depend on n

Q: If $T(n)$ is runtime of an algorithm,

What are a, b, d in words?

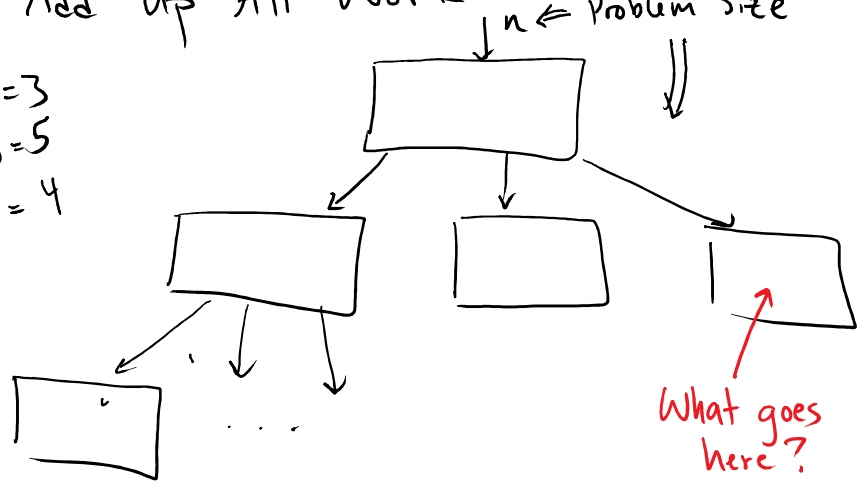
A: a: # of recursive calls

b: factor by which problem shrinks in recursive call

d: characterizes extra work outside recursive call

Let's Add Up All Work

ex: $a=3$, $b=5$, $d=4$



Input size

n

$\frac{n}{5}$

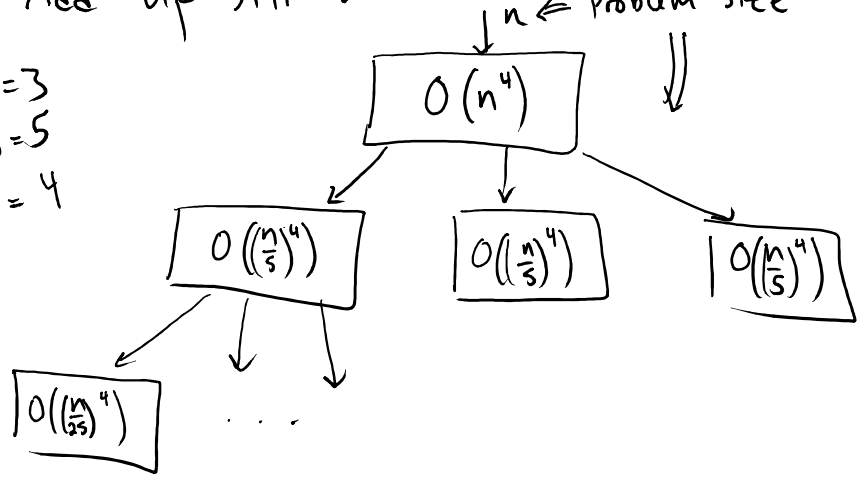
$\frac{n}{25}$

A) $O\left(\frac{n}{5}\right)$ B) $O\left(\frac{n}{2}\right)$ C) $O\left(\left(\frac{n}{5}\right)^4\right)$ D) $O\left(\left(\frac{n}{3}\right)^4\right)$

S.KIMMEL

Let's Add Up All Work

ex: $a=3$
$b=5$
$d=4$



Input size

n

$\frac{n}{5}$

$\frac{n}{25}$

.
.
.