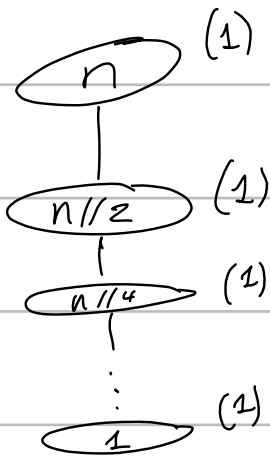


A hand-drawn recursion tree for calculating the n th Fibonacci number. The root node is n with a (1) next to it. It branches into $n-1$ (1) and $n-1$ (1). Each $n-1$ branches into $n-2$ (1) and $n-2$ (1). This pattern continues down to $n-3$ (1), which then branches into $n-3$ (1) and $n-3$ (1). The tree is drawn on lined paper, and the nodes are represented by hand-drawn ovals. The number of recursive calls for each level is indicated by the number of nodes at that level, which increases as the index decreases.

Level	calls per level	size per call	Cost per call	total cost per level
0	1	n	1	1
1	2	$n-1$	1	2
2	4	$n-2$	1	4
3	8	$n-3$	1	8
k	2^k	$n-k$	1	2^k
$n-1$	2^{n-1}	1	1	2^{n-1}

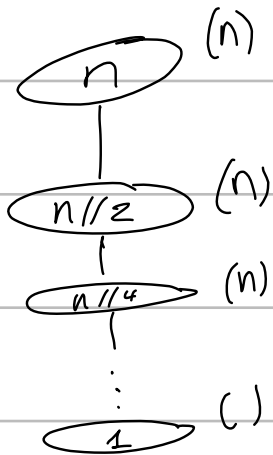
$$\text{Total Cost} = 1 + 2 + 4 + 8 + \dots + 2^k + \dots + 2^{n-1} = 2^n - 1 = \Theta(2^n)$$

Part B



$$T(n) = \underbrace{1+1+1+\dots+1}_{\log(n)} = \log(n)$$

Part C



$$T_{\text{per call}}(n) = 1+1+1+1+\dots+1 = n$$

Because of the
for loop repeating from
1 to $n+1$

$$T_{\text{total}}(n) = n+n+n+\dots+n = n \cdot n = \theta(n^2)$$