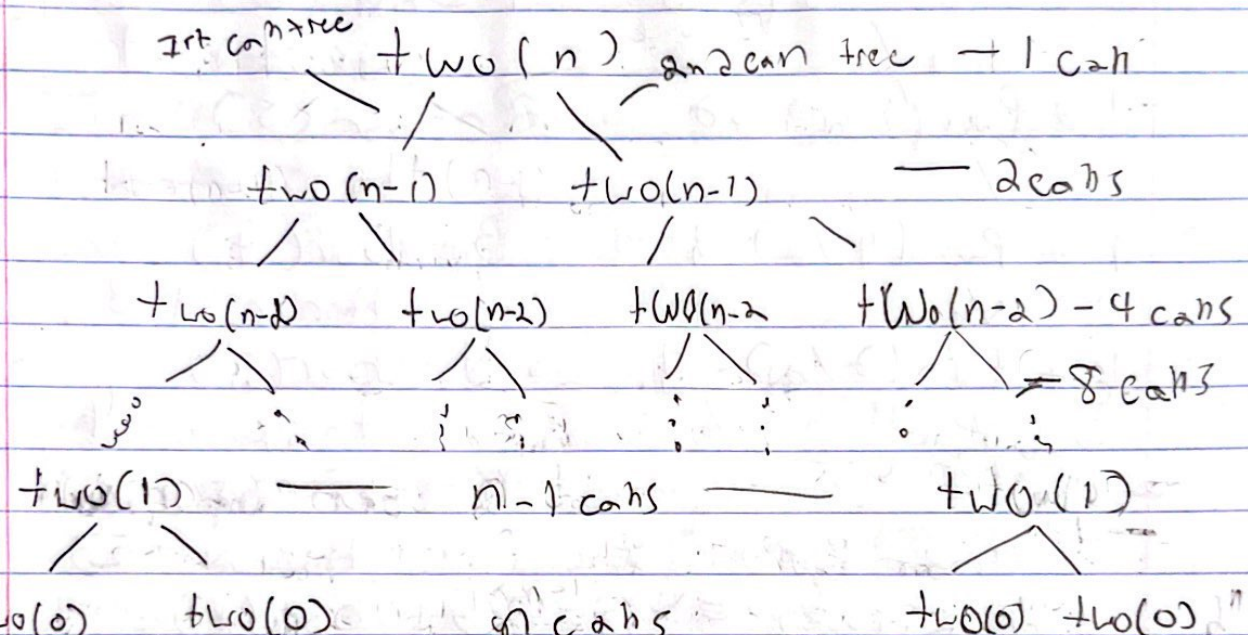
Johanne McKenahan
02/11/2022

CSCD 300-040 HW4

1. recursion tree for two(int n)



The recursive tree is the same no matter if n is positive or negative. Each time the method is called, the amount of calls double eventually equalling n calls. You can represent this like the equation

$$1 \times 2 \times 2 \times 2 \times 2 \times 2 \times \ldots = n$$

There are n amount of 2's for n calls so you can write the graf as

$$f(n) = 2^n$$

so the big oh time complexity is $O(2^n)$

2. analyze each loop by themselves

- 1st loop - for(i = n/2 ; i > 0 ; i = i/2)
  - each loop iteration, n is halved after being initially halved. This is done until the condition of i > 0 is false. rewriting the division as multiplication
    $2 \cdot n = \underbrace{2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot \ldots \cdot 1}_{\text{\# of 2's that equal n}}$
    (initial halving)
  - The # of 2's are based on n input values
  - you can rewrite the right half of the equation as $n = 2^x$ (after dividing the initial half)
  - Convert to log form → $\log_2 n = x$ (x: # of loop iterations)

  So $\underline{O(\log n)}$ is the time complexity for 1st loop
- 2nd loop - for (j = 0; j < n ; j++)
  - the # of loop is controlled by the n input value due to conditional statement being j < n.

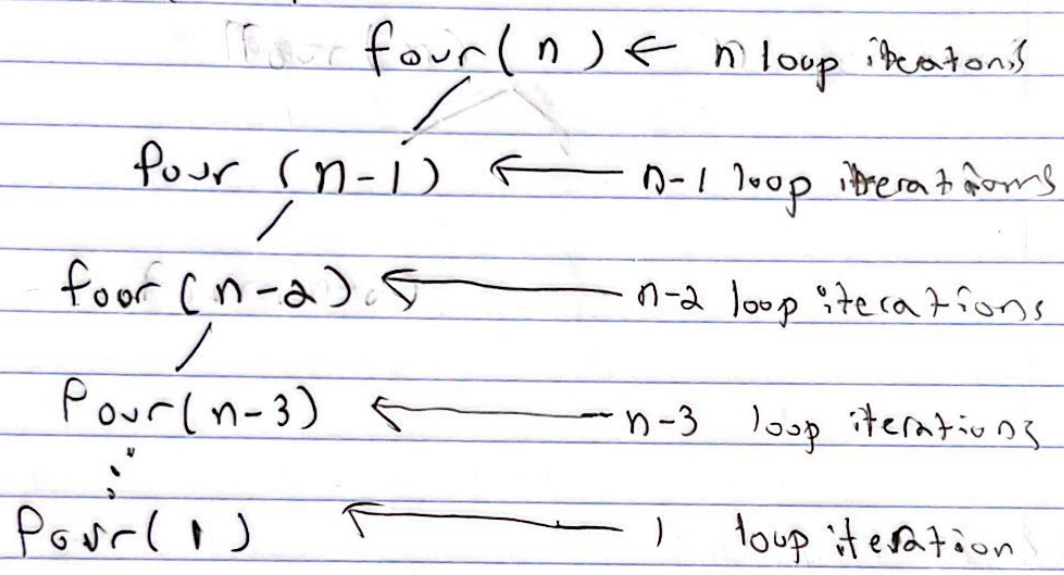
  So $\underline{O(n)}$ is the time complexity for the 2nd loop

3rd loop - For (k=0 k<n; k++)
- The # of loops is controlled by the n input value due to the conditional statement k < n. This means that O(n) is the time complexity for the 3rd loop

The loop structure has an outer loop (1st), an inner loop (2nd), and innermost loop (3rd)

Each time the 1st loop it creates, the second loop iterates n times. Because the 1st loop iterates log(n) times, the total number of iterations for the first and second loops will be log n on. The same property works for the second and third loop so the total # loops would be log n on on.
Simplifying this expression results in log n on^2. This is will be the dominant term so O(log(n)on^2) is the time complexity for three(n).

3. recursion tree



After the each return statement, the loop's # of iterations is based on the input value parameter of the method, if the input is 6 then the for loop will iterate 6 times, because each four() call has a different input, the total # of operations are all the for-loop iterations summed together.

$X = 1 + 2 + 3 + \dots + n-2 + n-1 + n$

↑ # of iterations

rewrite as the summation formula for X

$X = \sum \frac{n(n+1)}{2} \rightarrow \frac{n^2+n}{2}$ Dominant term

$O(n^2)$ is the time complexity for four(n) method