P3) Algorithm:

n-to-1(n) {

if (n==1) return n

if n%3==0

n = ~~n/=3~~ n-to-1(n/3)

~~n to~~

if n%2==0

n = n-to-1(n/2)

else ~~a~~ n=n-to-1(n-1)

return n

Runtime Analysis:

add $O(1)$
loop $O(n)$ times
operations take $O(1)$

Proof:
BC: n==1 we return 1 and done
IH: assume true for all inputs
IS: either n is divisible by 3, by 2, or we subtract 1
1) n%3==0 we recursively call func so if ~~3%3~~ n is divisible by 3 all the way to 1 which causes less ops operations to be run than if we were to divide by 2
2) n%2==0 in the event n is never divisible by 3 we can divide by 2 until n==1 if n is always divisible by 2
3) n-1 we ~~can~~ ~~divide~~ subtract one from n last so we ~~only have to~~ are least likely to use this, this is used as a fail-safe in which we can't use n/3 or n/2
So we minimize the number of ~~steps~~ operations for any n by using the biggest decrementer first