Matthew Wong
CS325
04/08/21
Assignment 2:

1. This cannot be written as a recursive function. This is because we are already at the base case and there needs to be no more recursive calls.
2. Recurrence Method

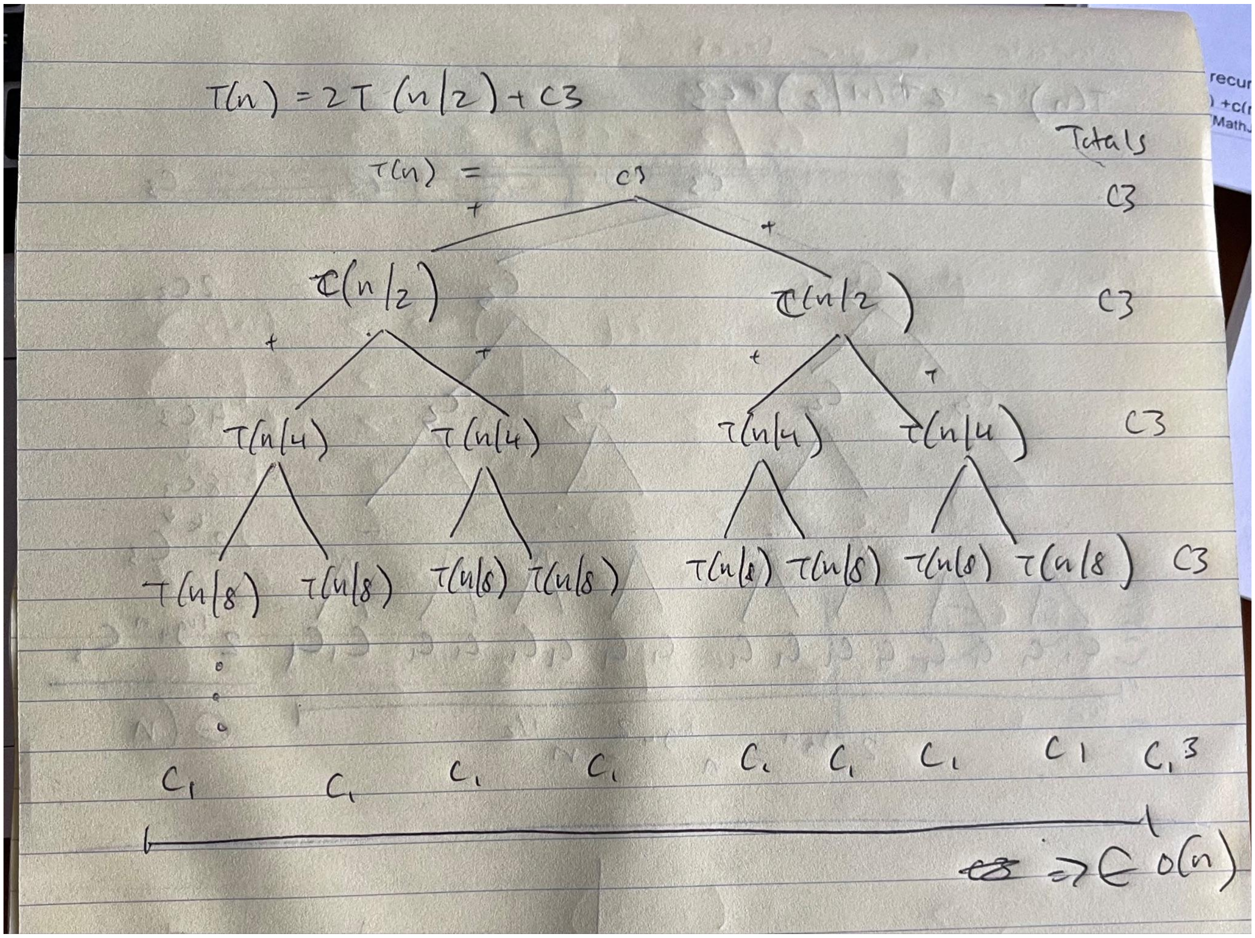
```
power2(x,n):
    if n==0:
        return 1
    if n==1:
        return x
    if (n%2)==0:
        return power2(x, n//2) * power2(x,n//2)
    else:
        return power2(x, n//2) * power2(x,n//2) * x
```

a.

i. **Substitution method:**

T(n) = 2T(n/2) + c3
=> T(n/2) = 2T(n/4) + c3 ##all we did was /2
=> T(n) = 2[2T(n/4) + c3]c3 ##subtitute back in to t(n)
=> 4T(n/4) + 3c3 ##simplify
=>T(n/4) = 2T(n/8) + c3 ##we need n/4 same as n/2
=> T(n) = 4[2T(n/8) + c3] + 3c3 #substitute into our equation
=> = 8T(n/8) + 7c3 #simplified from above
=> = 8[2T(n/16) + c3] + 7c3 #same as above for n/16
=> = 16T(n/16) + c3] + 15c3 #simplified out
=> = 32T(n/32) + c3] + 31c3 . . . #same as above for n/32
=> = 2kT(n/2k) + (2k-1)c3 ##our base case
=> T(0) = c1
=> T(1) = c2
=> T(n) = 2kT(n/2k) + (2k-1)c3
If n/2k = 1
Then n = 2klogn which is also = k
=> T(n) = 2lognT(n/2logn) + (2logn-1)c3 #sub log n as k for base
=> nT(n/n) + (n-1)c3 #2klog(n)=k then 2log(n) = 1
=> nT(n/n) + (n-1)3
=> nc2 + (n-1)c3
∈Θ(n)
=> T(n) = Θ(n)

ii. **Recurrence tree method:**



iii. **Master method**

Case 1: a = b^k

a=3, b=2, k=0

Time complexity: $\ominus(n^{log(baseb)a}) = \ominus(n^{log2}) = \ominus(n)$

b. T(n) = 4T(n/2) + n

= 4[4T(n/4)+n/2] + n

=16T(n/4) + n/2 +n

= 16[4T(n/8)+n/4] + 2n+n

=64(n/8)+4n+2n +n

= ….

= $4^{logn}T(n/2^{logn})$ +... + $4n + 2n + n$

= $n^{log4}T(n/n^{log2})$ +... + $4n + 2n + n$

= $n^2T(1)$ +... + $n(1 + 2 + 3 + ...)$

=⊖(n^2)

= time complexity ⊖(n^2)

3. A
    a. **Pseudocode**
       Declare array = [ ]
       #use loop and user input portion grab user input and store into array
       Declare temp variable
       #Store length of array into temp
       #divide portion
       Take index of array
       All even numbers add up
       All odd numbers add up
       #combine and conquer
       Now add up all and divide by index round up
    b. Since divide and conquer always halves it should be: O(nLogn)