Assignment Solution: Recursion, Recurrence Relations and Divide & Conquer

1. IMPLEMENT ALGORITHM:
   Write a function doBoxesOverlap(box1, box2) that take the coordinate positions of each box as input and return whether they overlap or not. Name your file **BoxAlgorithm.py**

Explanation: This problem cannot be solved using recursion as there are no recurring subproblems that we could use. <Optionally you could come up with any interesting recursive solution. That would be acceptable too>

One possible non-recursive solution:

```
def do_intersect(point1_left, point1_right, point2_left, point2_right):
    return (min(point1_right, point2_right) > max(point1_left,point2_left))

def doBoxesOverlap(box1, box2):

    return (do_intersect(box1[0],box1[2],box2[0],box2[2]) and
            do_intersect(box1[1],box1[3], box2[1], box2[3]))
```

2. SOLVE RECURRENCE RELATION USING THREE METHODS:

a.

```
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
```

constant $T(0) = c_1$, $T(1) = c_2$

$T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right)$

recurrence relation:

$T(n) = T(n/2) + T(n/2) + c$ for $n>1$

Base case:

$T(n) = c_1$ for $n=0$

$T(n) = c_2$ for $n=1$

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + C$$

$$T(n) = 2T\left(\frac{n}{2}\right) + C$$

Substitution Method:

$$T(n) = 2T\left(\frac{n}{2}\right) + C$$

$$= 2\left[2T\left(\frac{n}{2^2}\right) + C\right] + C$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + (2+1)C$$

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + C\right] + (2+1)C$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + \cdot\ (2^2 + 2 + 1)C.$$

If it reaches base case at k^{th} substitution

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + \underbrace{(2^{k-1} + 2^{k-2} + \cdots 2 + 1)}\, C$$

To find the sum of the first S_n terms of a geometric sequence use the formula
https://www.varsitytutors.com/hotmath/hotmath_help/topics/sum-of-the-first-n-terms-of-a-geometric-sequence

$$\frac{1(1-2^k)}{1-2} = 2^k - 1$$

Base case

$\frac{n}{2^k} = 0$ $\quad n = 0 \quad k = ?$ So, We cannot use this base case.

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow \boxed{k = \log_2 n}$

$$T(n) = 2^{\log_2 n} C_2 + (2^k - 1) C$$

$$= n C_2 + (2^{\log_2 n} - 1) C$$

$$= n C_2 + (n - 1) C$$

$$= n(C_2 + C) - C$$

$$= \Theta(n)$$

Master's method:

$a = 2, \; b = 2, \; f(n) = c$

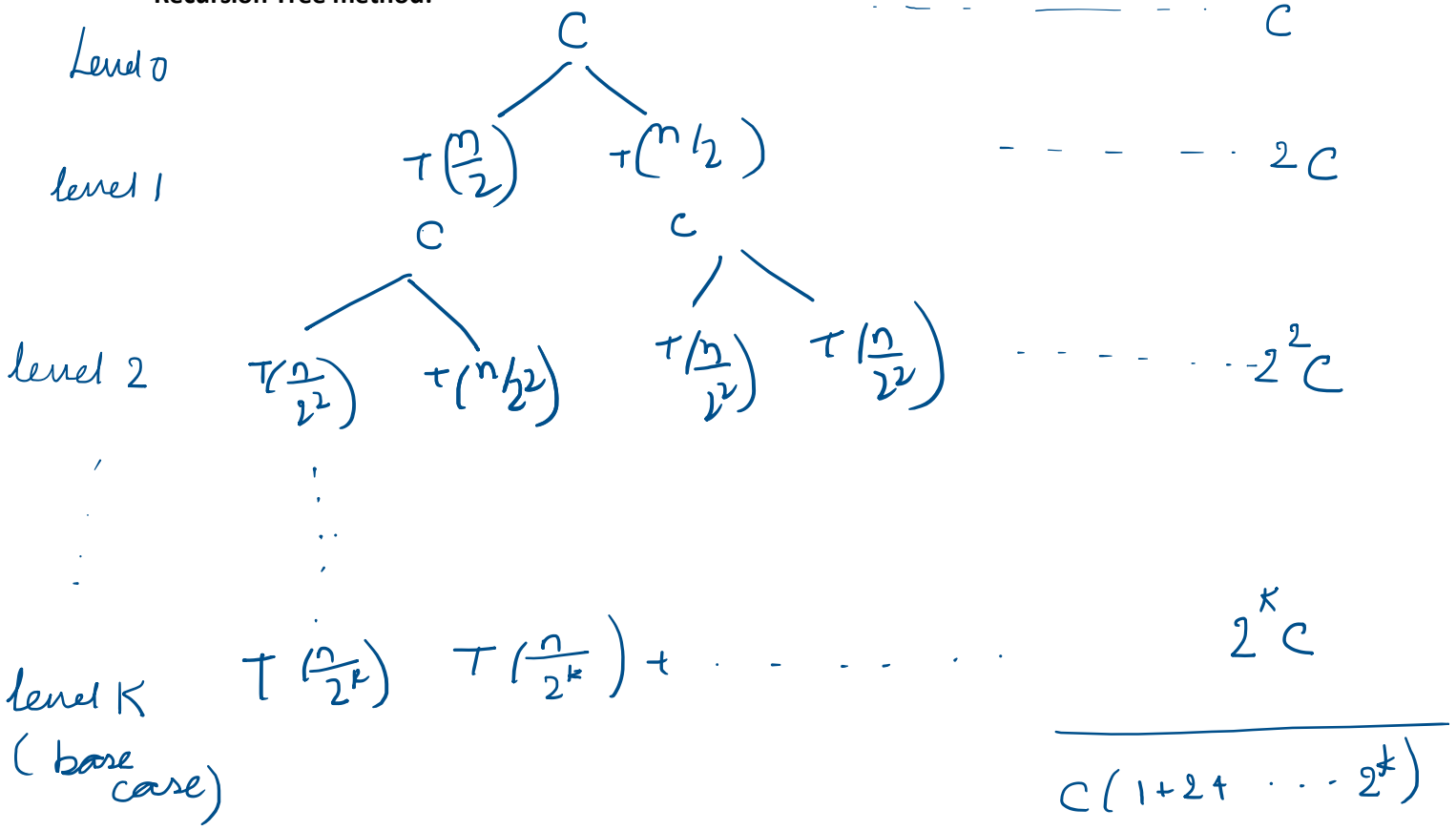
$f(n) = c \quad \text{vs} \quad n^{\log_2 2} = n$

$\Rightarrow$ case 1

$T(n) = \Theta(n^{\log_2 2})$

$= \Theta(n)$

Recursion Tree method:



$$T(n) = (1 + 2 + \cdots 2^k)c$$

$$= \frac{1(1 - 2^{k+1})}{1 - 2} c = (2^{k+1} - 1)c$$

$$= (2 \cdot 2^k - 1)c.$$

$$\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$$

$$T(n) = (2 \cdot 2^{\log_2 n} - 1)c$$

$$= (2n - 1)c$$

$$\in \Theta(n).$$

b. Give the asymptotic bounds for T(n) in each of the following recurrences. Make your bounds as tight as possible and justify your answers. Assume the base cases T(0)=1 and/or T(1) = 1.

a) $T(n) = 4T(n/2) + n$

Using Master Method: a = 4, b= 2; f(n) = n

$n^{\log_b a} = n^{\log_2 4} = n^{\log_2 (2)^2} = n^{2 \log_2 2} = n^2$

Comparing $n^{\log_b a} = n^2$ vs f(n) =n

Case 1: So, T(n) = $\theta(n^2)$

b) $T(n) = 2T(n/4) + n^2$

Using Master Method: a = 2, b= 4; f(n) = n^2

$log_b\, a = log_4\, 2 = log_4(4)^{1/2} = \frac{1}{2} = 0.5$ Since, $4^{\frac{1}{2}} = \sqrt{4} = 2$

Comparing $n^{log_b\, a} = n^{0.5}$ vs f(n) = n^2

Case 3:

Checking for regularity Condition

$$2f\left(\frac{n}{4}\right) = 2\frac{n^2}{4^2} = \frac{n^2}{8} \leq n^2 \quad , c = \frac{1}{8}$$

So, T(n) = $\theta(n^2)$

3. IMPLEMENT AN ALGORITHM USING DIVIDE AND CONQUER TECHNIQUE:
   a. Pseudocode:

```
MajorityBirthdays(days):
    return helper(days, 0, len(days)-1)

def helper(days, low, high):
    #base case
    if (low == high):
        return days[low]

    #find mid in the array

    # recursively call each half
    left_majority = helper(days, low, mid)
    right_majority = helper(days, mid + 1, high)

    #Same majority returned by left and right half
    if left_majority == right_majority: return left_majority # Or
return right

    # Find the majority (left_counter) in the left half that match with
left returned by previous divide step
    # Find the majority (right_counter) in the right half that match
with right returned by previous divide step

    #return the majority : combine step
    if left_counter > right_counter:
        return left
    else:
        return right
```

b. Proof:

Recursion Invariant: After each recursive call the helper() function returns the most common birthday found so far.

Base case: When low=high, when we reach single element in the days array, helper returns it. Since it is the only element it forms the majority birthday. Recursion invariant holds true for the base case.

Inductive Case: Assume that recursion invariant holds true for the i^{th} execution and for all values before i^{th} execution. In $(i+1)^{th}$ the majority element in the left half is in "left_majority" and the majority element in the right half is in "right_majority". In the combined array [low to high] we count the number of "left_majority" element and the number of "right_majority" element and return the one that occurs most commonly. The loop invariant holds true.

Termination: When the recursion terminates the most common element in the entire array is returned by the left_majority or right_element element that has been found so far, which is the solution for the problem.

c. Code:

```
def MajorityBirthdays(days):
    return helper(days, 0, len(days)-1)

def helper(days, low, high):
    #base case when days arr size is 1
    if(low == high):
        return days[low]

    #find mid in the array
    mid = (high - low) //2 + low

    #recursively call each half
    left = helper(days, low, mid)
    right = helper(days, mid+1, high)

    # if left and right have same majority element return either left
or right
    if left == right: return left

    #Find the majority in the left half and right half that match the
majority element found returned by previous divide steps
    left_counter = 0
    for i in range(low, high+1):
        if days[i] == left: left_counter += 1

    right_counter = 0
    for i in range(low, high + 1):
        if days[i] == right: right_counter += 1

    if left_counter > right_counter: return left
    else: return right
```