Solution: Recursion, Recurrence Relations and Divide & Conquer

1. **Solve recurrence relation using three methods**:

   Write recurrence relation of below pseudocode that calculates x^n, and solve the recurrence relation using three methods that we have seen in the explorations.

```
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) = c1 for n=0
T(n) =c2 for n=1

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

Substitution method:

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

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

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

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

If it reaches base case at k[th] substitution.

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + (\underbrace{2^{k-1} + 2^{k-2} + \cdots 2^2 + 2 + 1}_{\text{Geometric progression}})c$$

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

use Base case to find k

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

$$T(n) = 2^{\log_2 n} \cdot c_2 + (2^k - 1)c$$

$$= n c_2 + (2^{\log_2 n} - 1)c$$

$$= n c_2 + nc - c$$

$$= n(c_2 + c) - c$$

$$= \Theta(n)$$

Master Method:

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

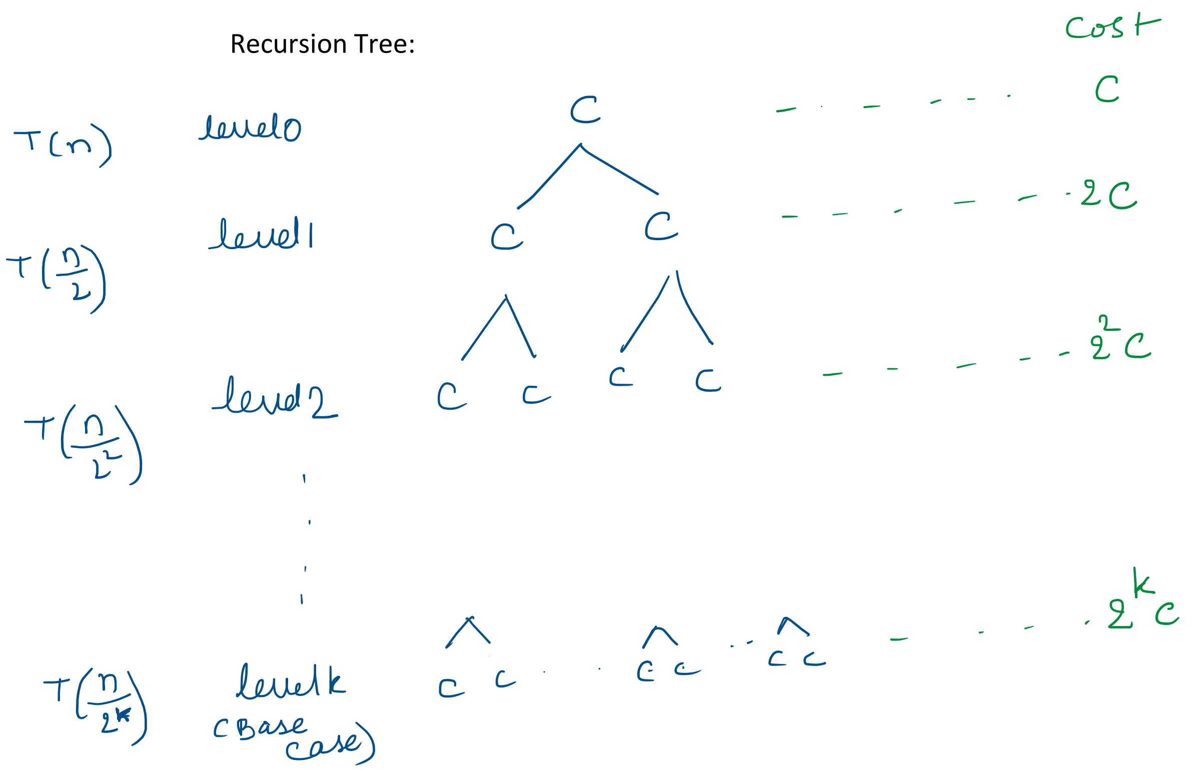
$$c \quad \text{vs} \quad n^{\log_2 2}$$

$$\Rightarrow \text{case 1}$$

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

$$\in \Theta(n)$$

Recursion Tree:



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

$$= \left(\frac{2^{k+1} - 1}{2-1}\right)c \quad \text{Geometric progression}$$

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

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

Base case to find k :

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

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

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

$$\in \theta(n)$$

2. **Solve recurrence relation using any one method**:

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: $a = 2$, $b = 4$; $f(n) = n^2$

$log_b\, a = log_4\, 2 = log_4(4)^{l/2} = \frac{1}{2} = 0.5$ 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**: Given two sorted arrays of size m and n respectively, find the element that would be at the k^{th} position in the final array.
   a. Write a pseudocode/describe your strategy for a function kthelement(Arr1, Arr2, k) that uses the concepts mentioned in the divide and conquer technique. The function would take two sorted arrays Arr1, Arr2 and position k as input and returns the element at the k^{th} position.

      One straight forward approach is to use merge sort to combine two arrays and find the kth item in the merged array. This will take O(m) time complexity if m>n or (n) if n>m.

   b. Implement the function kthElement(Arr1, Arr2, k) that was written in part a. Name your file **KthElement.py**

      Examples:
      Arr1 = [1,2,3,5,6] ; Arr2= [3,4,5,6,7] ; k= 5
      Returns: 4
      Explanation: 5^{th} element in the combined sorted array [1,2,3,3,4,5,5,6,6,7] is 4

```
def kthelement(arr1, arr2, k):
    m = len(arr1)
    n = len(arr2)
    sorted1 = [0] * (m + n)
    i = 0
    j = 0
    d = 0
    while (i < m and j < n):

        if (arr1[i] < arr2[j]):
            sorted1[d] = arr1[i]
            i += 1
        else:
            sorted1[d] = arr2[j]
```

```python
            j += 1
        d += 1

    while (i < m):
        sorted1[d] = arr1[i]
        d += 1
        i += 1
    while (j < n):
        sorted1[d] = arr2[j]
        d += 1
        j += 1
    return sorted1[k - 1]


arr1 = [1,2,3,5,6]
arr2 = [3,4,5,6,7]
k = 5
print(kthelement(arr1, arr2, k))
```