Assignment: 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
```

2. **Solve recurrence relation using any one method**: Find the time complexity of the recurrence relations given below using any one of the three methods discussed in the module. Assume base case T(0)=1 or/and T(1) = 1.

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

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 combined sorted 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 in the combined sorted array.
   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

PROBLEM # 1

RECURRENCE RELATION

BASE: $T(n) = c_1$ or $\Theta(1)$ when $n \leq 1$

RECURSIVE CALLS: $2T(n//2)$ when $n > 1$

$$T(n) = 2T(n/2) + \Theta(1)$$

#1 SUBSTITUTION

$$T(n) \begin{cases} c_1 & n \leq 1 \\ 2T(n/2) & n > 1 \end{cases}$$

FIRST SUB

$n \to n/2$ $\quad T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right)$

$$T(n) = 2\left[2T\left(\frac{n}{4}\right)\right] = 4T\left(\frac{n}{4}\right)$$

SECOND SUB

$n \to n/4$ $\quad T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right)$

$$T(n) = 4\left[2T\left(\frac{n}{8}\right)\right] = 8T\left(\frac{n}{8}\right)$$

K^{th} SUB

$$T(n) = 2^k T\left(\frac{n}{2^k}\right)$$

BASE: $T(1) = T\left(\frac{n}{2^k}\right) = c_1$; $1 = \frac{n}{2^k} \to k = \log_2 n$

$$T(n) = 2^k c_1 = 2^{\log_2 n} c_1 = n^{\log_2 2} c_1 = n c_1$$

$$T(n) = \Theta(n)$$

#2 RECURSION TREE METHOD

Level 0 : $T(n)$ . . . . c $\longrightarrow$

Level 1 : $T\left(\frac{n}{2}\right)$ $\quad T\left(\frac{c}{2}\right)$ $\quad T\left(\frac{c}{2}\right)$

Level 2: $T\left(\frac{n}{4}\right)$ $\quad T\left(\frac{c}{4}\right)$ $\quad T\left(\frac{c}{4}\right)$

Level i: $T\left(\frac{n}{2^i}\right)$ $\quad T(1)$ $\quad T(1)$

at level i, $T(1) = T\left(\frac{n}{2^i}\right)$; $i = \log_2 n$

Total cost = cost per level * # levels $= c \times i$

$= c \times \log_2 n$

$T(n) = \Theta(\log n)$

#3 MASTER METHOD

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$a \geq 1 \quad b > 1 \quad f(n) > 0$ polynomial

$$T(n) = 2T(n/2) + \Theta(1)$$

$a = 2 \quad b = 2 \quad f(n) = \Theta(1)$

$n^{\log_b a} = n^{\log_2 2} = n$

Case 1: $f(n)$ grows asymptotically slower than $n^{\log_b a}$

$T(n) = \Theta(n)$

PROBLEM #2

a) $T(n) = 4T\left(\frac{n}{2}\right) + n$

$a = 4, \ b = 2, \ f(n) = n$

$n^{\log_2 4} = n^2$

CASE 1: $f(n)$ grows slower than $n^{\log_b a}$

$T(n) = \Theta(n^2)$

b) $T(n) = 2T\left(\frac{n}{4}\right) + n^2$

$a = 2, \quad b = 4, \quad f(n) = n$

$n^{\log_4 2} = n^{\frac{1}{2}}$

CASE 3: $f(n)$ grows faster than $n^{\log_b a}$

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

PROBLEM #3

a) DIVIDE: Check k//2 element in each array

Kthelement (Arr 1, Arr 2, k):

  i = min(length(Arr 1), k//2)

  j = min(length(Arr 2), k//2)

CONQUER: Discard smaller elements until you reach the k^{th} element

k^{th}element (Arr1, Arr 2, k):

  ...

  if Arr 1[kth //2 element] < Arr 2[kth//2 element]:

    k^{th}element (Arr1[k^{th} element to end], Arr 2, k-i)

discarded element

  else:

    k^{th}element (Arr 1, Arr2[k^{th}element to end], k-j)