HQ1

Name: Debnath Kundu (MT22026) & Manvendra Kumar Nema (MT22038)

Answer for Q1:

```
// A & B are the 2 given n-sized sorted arrays & C is the final output array for 2n elements
def Merge(A,B,C,n):
    i=n
    j=n
   Combine(A,i,B,j,C,n)

// A & B are the 2 given n-sized sorted arrays & C is the final output array for 2n elements
// i & j denotes the pointer to the smallest element of A & B respectively.

def Combine(A,i,B,j,C,n):
    if i==0 and j==0:
        return
    if i>0 and j>0:
        if A[ n - i ] < B[ n - j ]:
            C[2*n -i - j ]= A [n-i]
            Combine(A,i-1,B,j,C,n)
            return
        else:
            C[2*n -i - j ] = B[ n -j ]
            Combine(A,i,B,j-1,C,n)
            return
    if i != 0:
            C[2*n -i - j ]= A [n-i]
            Combine(A,i-1,B,j,C,n)
            return
    if j !=0 :
            C[2*n -i - j ] = B[ n -j ]
            Combine(A,i,B,j-1,C,n)
            return
```

Answer for Q2:

Let the time complexity of **Combine** be **T(i,j)**, where **i, j & k** denote the index of the arrays **A, B & C** respectively.

$T(i, j) \quad = T(i-1, j) \;\; [when\ A[\,i\,] \; < \; B\,[\,j\,]\,]$

or

$T(i, j-1) \;\; [when\ A[\,i\,] \; > \; B\,[\,j\,]\,]$

i >=0, j >= 0;

Base Case: $T(0, 0) = 1$
The **Combine** function is designed such, that at a time only one element either from array **A** or **B** will be copied to **C**. Hence, in the above recurrence relation, only one of **i** and **j** changes at a time, while the other remains constant.

Since **i** and **j** are moving independently, it's better to solve this problem in terms of the array size. Here

=> i+j = n + n; => i+j = 2*n;

If the array **A** & **B** has n elements each, the merged array **C** will have **n + n = 2n** elements. The above **Combine** function traverses the two given arrays (each of size n) and copies them to a single sorted array depending upon which element is smaller. Therefore, the **Combine** function will have to traverse **2*n** elements in total for the entire operation.

Let **2n = m, T(m)** denote the time complexity of the **Combine** function.
Base case: when there are no elements left in either of the array i.e. $T(0) = 1$

Each time an element is traversed in **A** or **B**, the size of the array **C** is decreased until the complete array is filled.

$T(m) = T(m - 1) + d$ *[where d is a constant]*
$T(m - 1) = T(m - 2) + d$
$T(m) = T(m - 2) + (d + d)$
....
$T(m) = T(m - k) + k(d)$

$Let\, m - k = 0, \; => \; m = k$
$T(m) = T(0) + m * d$ => $T(m) = 1 + m * d$ *[where m=2n]*
$T(m) = 1 + 2 * m * d$
$Let\, d' = 2 * d'$
$T(m) = 1 + n * d'$
$T(m) = O(n)$
Hence complexity of **Combine** function is order of n i.e. **O(n)**, where n is the number of elements in the given arrays.

Let the complexity of the **Merge** function is **M(n)**, where n is the number of elements in the given arrays.
M(n) = 2 + O(n) because the **Merge** function is calling the **Combine** function whose complexity is O(n). So complexity of **Merge** function is O(n)