1. Recursive algorithm to merge two sorted arrays of equal size.

```
// A is a sorted array with n elements
// B is a sorted array with n elements
// C is an empty array with space for 2n elements
// given two sorted arrays A and B Merge method merges two
   sorted arrays into C.
// assume no duplicates anywhere.

    Merge (A, B, C, n):
        i = 1
        j = 1
        PutElement (2n)

// assume A, B, C are global arrays and i, j variables are global
// PutElement (k) puts the correct element at C[k], taking it
   from either A or B

    PutElement (k):
        if k = 0 : return
        PutElement (k-1)
        if i > n
            C[k] = B[j]
            j = j + 1
        else if j > n
            C[k] = A[i]
            i = i + 1
        else if A[i] < B[j]
            C[k] = A[i]
            i = i + 1
        else
            C[k] = B[j]
            j = j + 1
```

2. Time complexity analysis of recursive Merge algorithm.

Let- $T(k)$ denote the time complexity of PutElement(k).

$$T(k) = T(k-1) + c$$
$$= T(k-2) + c + c$$
$$= T(k-3) + c + c + c$$
$$\vdots$$
$$= T(k-m) + mc$$

for the base case, $k - m = 0 \Rightarrow m = k$

$$T(k) = T(0) + kc$$
$$T(k) = 1 + kc$$
$$T(k) \cong O(k)$$

Now, from Merge() method we know that- $k = 2n$

and time complexity of Merge() method $\cong$ time complexity of PutElement-()

Therefore, the time complexity of Merge() method is $O(n)$.