Question 2:

a)
Time complexity of 3 way merge sort

Dividing the arrays gives constant time complexity of 3 way merge sort O(1)
The size of 3 sub-arrays are N/3, so the relation should be T(N/3) and it recursively sort three sub-arrays so it is 3T(N/3).
While merging the sub-arrays, the time complexity is linear as it place the elements in correct order using comparison fo 3 sub-arrays so its O(N)

So,
T(N) = 3T(N/3) + N
T(N) = 3(3T(N/9) + N/3) + N
T(N) = 3(3(3T(N/27) + N/9) + N/3) + N

T(N) = 3^k(T(N/3^k)) + N/3^(k-1) + N/3^(k-2) + ... + N

Here, N/3^k = 1 for base case
so, k = log(N) of base 3.

So, T(N) = 3^log(N)*T(1) + N/3^(logN - 1) + ... + N

Now, 3^logN = N
and
N/3(logN - 1) = N/(N/3) = 3
So, N/3^(logN - 1) + ... + N gives us complexity O(N)

T(N) = NT(1) + O(N)
T(N) = O(N)

So, the complexity of 3 way merge sort is O(N)

b)
2 way merge sort has better time complexity than 3 way merge sort because 2 way merge sort divide arrays more quickly than 3 way merge sort mostly if the size is large. So, 2 way merge sort has better time complexity and performaNce than 3 way merge sort.

c)

In place merge sort algorithm using space complexity O(n)

```
merge(array[], start, mid, end)
   int a1[start to mid]
   int a2[mid+1 to end]

   for(i from 0 to size of a1)
      a1[i] = array[start+i]
   for(i from 0 to size of a2)
      a2[i] = array[mid+i]

  // Merging array a1 and a2 to original array
```

```
    i and j = 0 and k = start
    while(i < size of a1 and j < size of a2)
        if(a1[i] <= a2[j])
            array[k++] = a1[i]
            i++
        else
            array[k++] = a2[j]
            j++


    //coping remaining elements to original array
    while(i is less than size of a1) {
        array[k++] = a1[i++]
    }
    while(j is less than size of a2) {
        array[k++] = a2[j++]
    }


mergeSort(array[], start, end)
    if(start >= end)
        return
    else
        mid =  half of the size of array

        mergeSort(array, start, mid)
        mergeSort(array, mid+1, end)

        merge(array, start, mid, end)
```

Here, merge function merge the two arrays to original array using two arrays a1 and a2
mergeSort function divides the array into two halves and then sort them