CS62: Spring 2025 | Lecture #13 (Mergesort) worksheet | Jingyi Li

```java
private static <E extends Comparable<E>> void merge(E[] a, E[] aux, int lo, int mid,
int hi) {
        for (int k = lo; k <= hi; k++){
            aux[k] = a[k];
        }
        int i = lo, j = mid + 1;
        for (int k = lo; k <= hi; k++) {
            if (i > mid) { // ran out of elements in the left subarray
                a[k] = aux[j++];
            } else if (j > hi) { // ran out of elements in the right subarray
                a[k] = aux[i++];
            } else if (aux[j].compareTo(aux[i]) < 0) { // j is smaller
                a[k] = aux[j++];
            } else { // i is smaller or equal
                a[k] = aux[i++];
            }
        }
    }
```

1. How many calls does merge() make to compareTo() in order to merge two already sorted subarrays, each of length n/2 into a sorted array of length n?
   Hint: Think of a best case and worst case array example and work through the code.

A. ~1/4n to ~1/2n
B. ~1/2n
C. ~1/2n to n
D. ~n

2. Which of the following subarray lengths will occur when running mergesort on an array of length 10?

A. { 1, 2, 3, 5, 10 }
B. { 2, 4, 6, 8, 10 }
C. { 1, 2, 5, 10 }
D. { 1, 2, 3, 4, 5, 10}

3a. Is Mergesort in place?

3b. Is Mergesort stable?

3c. “Inconclusive evidence” is the idea that the data being sorted and `.compareTo()`’d might not paint the whole picture. Sorting numbers is one task, but in the real world, we sort things like candidates for jobs, search result rankings, financial risk assessments, or movie/song recommendations.
Pick one of these “real world” categories (or choose your own) and discuss how we might make the “evidence” more “conclusive.” Should you change what “evidence” gets sorted? Add more evidence? Avoid sorting altogether?