2) (10 pts) ALG (Sorting)

(a) (5 pts) Consider using Merge Sort to sort the array shown below. What would the state of the array be right before the ***last*** call to the Merge function occurs?

index	0	1	2	3	4	5	6	7	8	9
value	20	15	98	45	13	83	66	51	88	32

Answer:

index	0	1	2	3	4	5	6	7	8	9
value	**13**	**15**	**20**	**45**	**98**	**32**	**51**	**66**	**83**	**88**

Grading: 5 pts for the correct answer, 2 pts if each pair is sorted, 1 pt if the whole array is sorted, otherwise, count how many of the blanks are correct and divide by 2, rounding down.

(b) (5 pts) An inversion in an array, *arr,* is a distinct pair of values *i* and *j*, such that *i* < *j* and *arr[i]* > *arr[j]*. The function below is attempting to count the number of inversions in its input array, *arr*, of size *n*. Unfortunately, there is a bug in the program. Given that the array passed to the function has all distinct values, what will the function always return (no matter the order of values in the input array), in terms of n? Also, suggest a quick fix so that the function runs properly. (Note: analyzing inversions is important to studying sorting algorithm run times.)

```
int countInversions(int arr[], int n) {    // line 1
    int i, j, res = 0;                     // line 2
    for (i = 0; i < n; i++) {              // line 3
        for (j = 0; j < n; j++) {          // line 4
            if (arr[i] > arr[j])           // line 5
                res++;                     // line 6
        }                                  // line 7
    }                                      // line 8
    return res;                            // line 9
}                                          // line 10
```

Return value of the function in terms of n: $\frac{n(n-1)}{2}$, the sum of the first n-1 non-negative integers.

Line number to change to fix the function: 4

Line of code to replace that line: `for (j = i+1; j < n; j++) {`

As it's currently written, the code compares each value arr[i] to all other values in the array and adds 1 to res each time arr[i] is bigger. So, for the largest value in the array n-1 is added to res, for the second largest value in the array n-2 is added, and so forth, so res will simply be the sum of the integers from 0 to n-1. The issue with the code is that it doesn't enforce the restriction i < j given in the definition of an inversion. To enforce this, we force j > i by making j's starting value i+1, the smallest integer greater than j.

Grading: 2 pts return value, 1 pt line to change, 2 pts for the change.