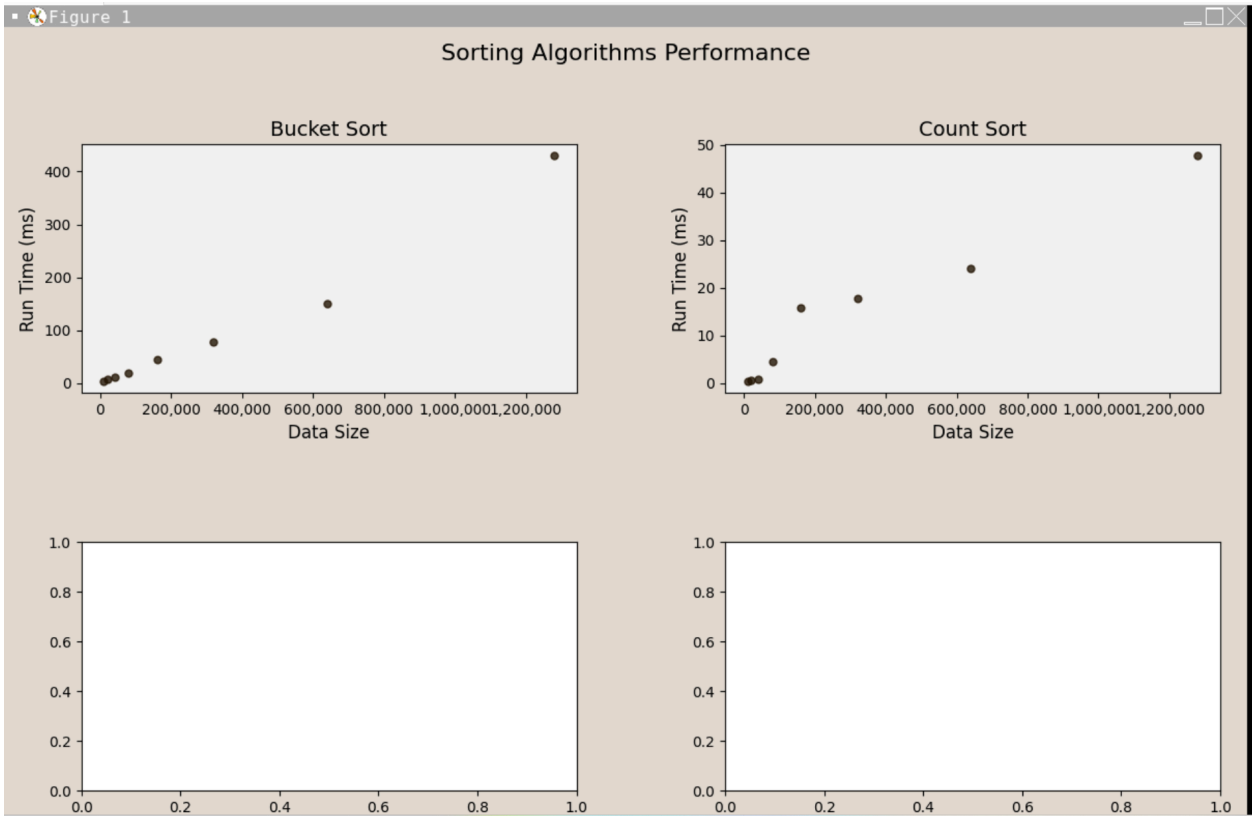
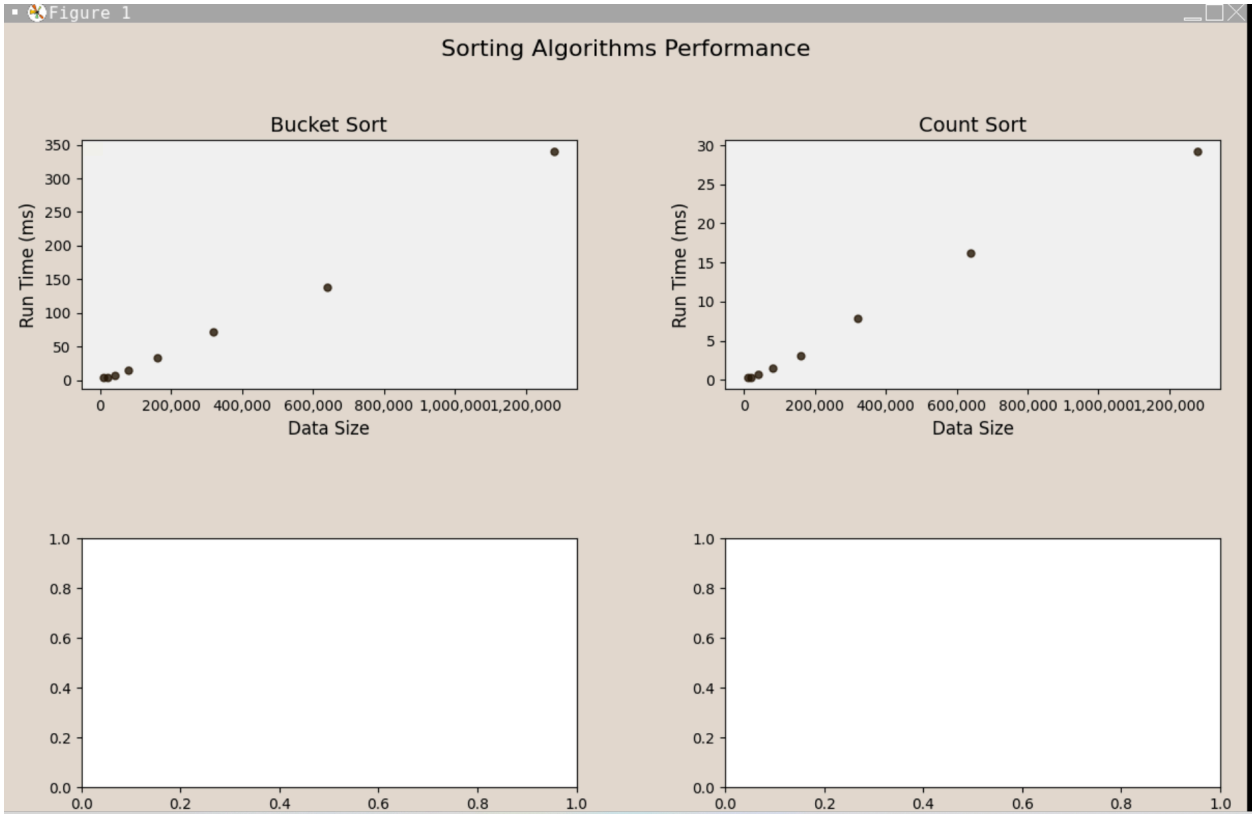


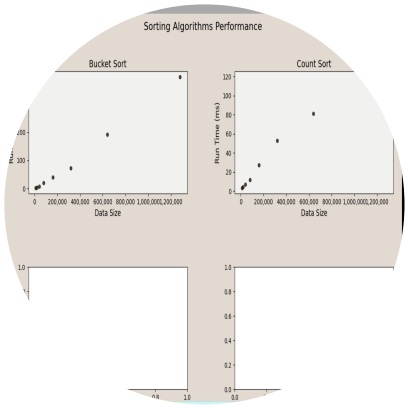
The Count Sort plot shows a more linear growth pattern compared to Bucket Sort. This aligns with Count Sort's known time complexity of O(n + k), where n is the number of elements and k is the range of input. The runtime increase is much less steep than Bucket Sort, especially for larger data sizes.




```java
public static ArrayList<Integer> generateArrayList(int size) {
        ArrayList<Integer> arr = new ArrayList<>();
        // correct loop
//          for (int i = 0; i < size; i++) {
//              arr.add((int) (Math.random() * 1000));
//          }
//          return arr;

        Random rand = new Random();
        for (int i = 0; i < size; i++) {
            // Generate numbers in a few tight clusters
            int cluster = rand.nextInt(3); // Choose one of 3 clusters
            if (cluster == 0) {
                arr.add(rand.nextInt(100)); // Numbers from 0-99
            } else if (cluster == 1) {
                arr.add(rand.nextInt(100) + 10000); // Numbers from 10000-10099
            } else {
                arr.add(rand.nextInt(100) + 20000); // Numbers from 20000-20099
            }
        }
        return arr;
```

The best I could get without figuring out how to set the random seed. Our Bucket sort works best when elements are evenly distributed across the entire range. By clustering the numbers into a few tight ranges, we're forcing most elements to fall into just a few buckets, leaving many buckets empty.

- 

- .

```
public static ArrayList<Integer> generateArrayList(int size) {
    ArrayList<Integer> arr = new ArrayList<>();



    Random rand = new Random();
    for (int i = 0; i < size; i++) {
        // Generate numbers with a large range to increase the count array size
        arr.add(rand.nextInt(1_000_000)); // Large range of values
    }
    return arr;
}
```

-

Count sort's time complexity is O(n + k), where n is the number of elements and k is the range of input values. By significantly increasing the range of possible values (k) while keeping the array size (n) the same, we've made the k term in O(n + k) much larger.