Quiz 3 – CS323

Tamer Avci #2105024

Steps

1- I wrote the ProbabilityBucketSort class first. In that class I defined a constructor with the bucket size. Here the bucket size represents the number of buckets. G_buckets must have 10 lists(buckets in it) because we have 10 intervals which are: [0.0, 0.1), [0.1, 0.2), [0.2, 0.3), [0.3, 0.4), [0.4, 0.5), [0.5, 0.6), [0.6, 0.7), [0.7, 0.8), [0.8, 0.9), [0.9, 1). The constructor did not need to have any input parameters, because min and max are already pre-determined (0 and 1). Super method is called with the second parameter *true*, because we will need to sort the items within each bucket as we have now an interval assigned to each bucket. The second method in the ProbabilityBucketSort class is the getBucketIndex. Here I changed the input key from int to double as we are now dealing with probabilities. However, since we still want to return the right index of the bucket I changed the return value to key*10 and then I casted the result into an int. By doing so, I access the first decimal value of that double number which represents the right bucket index of that key.

2- Then I created a class ProbabilityBucketSortTest under the sort package, and defined a method test to evaluate the accuracy of my algorithm using 10 different lists of 100 random double values between 0 and 1. To do that I used the Junit accuracy test. I called the test method with iterations=10 times and size 100 and an engine being ProbabilityBucketSort. Notice that sorting algorithm does not have any input. What the test method actually does is that it creates 10 different times two identical random arrays of size 100 using getRandomDoubleArray method which I created inside the class. These arrays are called original and sorted. The "original" array gets sorted using my ProbabilityBucketSort and the array "sorted" gets sorted using Java's default sorting algorithm with Arrays.sort method. Finally assertTrue tests whether these two arrays are still identical after we applied the sorting algorithms on them. I print both arrays in order to be able to see them after they are sorted.

Analysis, key ideas and observations

I have quickly realized that although my sorting algorithm was passing the test, there was no way to be sure that the buckets were properly created and items were placed in the right buckets. Because after all they would definitely get sorted correctly inside their individual buckets because we are using Java's sorting algorithm to do that. For this reason, I print the buckets before they get sorted in the AbstractBucketSort class. if (b_sort):Collections.sort(bucket); **Note: You have to run my test code with the AbstractBucketSort class in the directory whose content I changed a little bit.** So if you have a look at the console after you run my ProbabilityBucketSortTest you will see the following:

10 different lists in the order of:

a) Their buckets in unsorted order. This helped me check if they were placed correctly.
b) List sorted by ProbabilityBucketSort.
c) List sorted by Java sorting algorithm.