CS 0445 Spring 2022
Recitation Exercise 9

Introduction:

In lecture we discussed RadixSort and its asymptotic run-time of O(N). However, we also said that RadixSort has a lot of overhead – there is some preprocessing of the data as well as a lot of copying from the array to queues and back to the array. So it would be interesting to do an empirical test and compare RadixSort with a good divide and conquer comparison based sort such as MergeSort or QuickSort.

Details:

This exercise will have two parts, which together will enable you to do some empirical testing of RadixSort and MergeSort.

Part One:
We want to test our sorts on arrays of Strings, but to do this we need a source of arbitrary String data. For the first part of this exercise, you will write a static method called

public static String randWord(int maxSize)

This method will generate and return a random "word" containing at most maxSize upper case letters. The actual length of the word and the actual characters in it will be randomly chosen, but the length should not exceed maxSize and the characters must all be capital letters. As a hint for getting started with this method, I recommend the following approach: first determine the (random) length, then generate a char array of that length, then fill the char array with randomly generated letters, then create a String from the array and return it. Test your method with a simple driver program to fill an array and print it out. You should see random words of various lengths in your output. Note that your results from Part Two depend on this method being implemented correctly.

Part Two:
Once you have your randWord() method working you should write a short main program to empirically time the mergeSort() and RadixSort() methods discussed in lecture. The code for these methods can be found in files TextMergeQuick.java and RadixDemo.java. You can call the methods directly from those files – you do not have to re-code them.

Your main program, CS445Rec9.java, should be similar to that of Recitation Exercise 8:

1) Ask the user to enter an array size and input it
2) Ask the user to enter a maximum word length and input it
3) Create two arrays of String equal in length to size value input by the user
4) Fill the arrays with random Strings using the method you wrote in Part One and the maximum length input by the user.
5) Time a single call to mergeSort() and a single call to RadixSort(), using System.nanoTime() before and after and using the difference of the calls as the runtime.
6) Print out the results for each sort.

Once you have your program complete and working, experiment with it in the following two ways:

1) Keep the max word length constant and increase the array size and see the results for both algorithms. For example, make the max word length 4 and try array sizes 500000, 1000000, 2000000 and 4000000. Compare the results of the algorithms as the array sizes grow. Try several runs for each size and if you wish average the results.
2) Keep the array size constant and try various maximum word lengths and see the results for both algorithms. For example, make the array sizes 2000000 and try maximum word lengths 3, 4, 5, 6, 7, 8, 9, 10. Compare the results of the algorithms as the maximum word lengths grow. Try several runs for each length and if you wish average the results.

Toward the end of the recitation period, share your impressions and conclusions with your classmates. How do the algorithms compare as the array sizes increase and as the word lengths increase? Explain your results and reasoning to the rest of the class.