Michael Howard

CSII Lab4

Queues and Radix Sort

Answer the following questions:
What does *your* graph's running-time shape look like? Does your graph believably demonstrate the internet's expected execution-time for a RadixSort algorithm? Explain ***in detail*** why or why not.

My program uses the class myQueue that is based off of vectors. This myQueue class is used in the main.cpp file. At first I test the classes methods using assert, this confirms that the class is running as expected. Then I worked on the Radix Sort Timings by making a function that will generate random 9 digit numbers. It takes an input of size to determine the length of the 9 digit number vector. I do this because then I can make a loop in the main.cpp that makes everything easier. I loop over the length of the vectorSizes (10), then in the loop I grab the actual value of the vectorSizes using i. Then I initialize a mainBin and generateRandNums. Then I will run the generated number queue called mainBin in the function timeRadixSort.

timeRadixSort will then time how long it takes to sort the pre generated queue from start to end displaying relevant information. To do this it calls the radixSort function that implements Radix Sort.

The first step in Radix Sort is to make 10 queues. Then we will loop over each digit in the 9 digit random numbers. We dequeue it to the queue with the correct digit value. This took me a while but it will check the digit spot it's at in the loop and then sort it into a queue that matches that number. Then next step involves moving everything back into the main bin by dequeuing all the queues into the sortedObject bin which in this case is the sorted "mainBin" that is returned to the timeRadixSort function to record how long it takes to sort the random 9 digit numbers given to it from the mainBin.

I ran Radix Sort 5 times, recording the 10 timings in an excel file. Then I averaged the timings and displayed the graph. The black line with the x's is the average line. Most of the timings were quadratic, but one of them looked much flatter. I think that Radix sort is mostly linear. Though sometimes it can be N log n if a lot of the randomly generated numbers are either very low digits or are repeated. This would speed up the algorithm since there is less work.

(Here is the data I gathered and the graph it makes)

X-Axis	intervals	First Time	Second Time	Third Time	Fourth Time	Fifth Time	Average
	1000	0.009951	0.008717	0.013929	0.009266	0.009847	0.010342
	2000	0.081412	0.094273	0.031155	0.112254	0.082804	0.0803796
	3000	0.197119	0.272238	0.047066	0.2614	0.193982	0.194361
	4000	0.360077	0.427287	0.066508	0.398139	0.348736	0.3201494
	5000	0.567929	0.553416	0.089509	0.566654	0.553069	0.4661154
	6000	0.819817	0.809502	0.11939	0.796544	0.791425	0.6673356
	7000	1.11953	1.07523	0.164526	1.12653	1.09199	0.9155612
	8000	1.62855	1.41141	0.200008	1.42009	1.43302	1.2186156
	9000	1.80784	1.79801	0.228857	1.79128	1.77672	1.4805414
	10000	2.2104	2.27918	0.271269	2.19884	2.21078	1.8340938