Lab 6 writeup

1. Record data in table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **10** | **100** | **500** | **5000** | **25000** |
| Bubble | 1286.4 | 78627.4 | 1956620.8 | 180222129.6 | 370616171.7 |
| Insertion | 875.2 | 7840.5 | 197531 | 15605676 | 379255334 |
| Quick | **1095** | **12410** | **74460** | *446760* | *2680560* |
| Merge | **6933** | **18278** | **89,240** | *270000* | *801000* |
| Radix | 36224.33333 | 213333 | 749038 | 9196130.5 | 56169968.5 |

Figure 1: Data table. Note: Bold Values were counted manually (explained later); italics were estimated

1. Graph this data

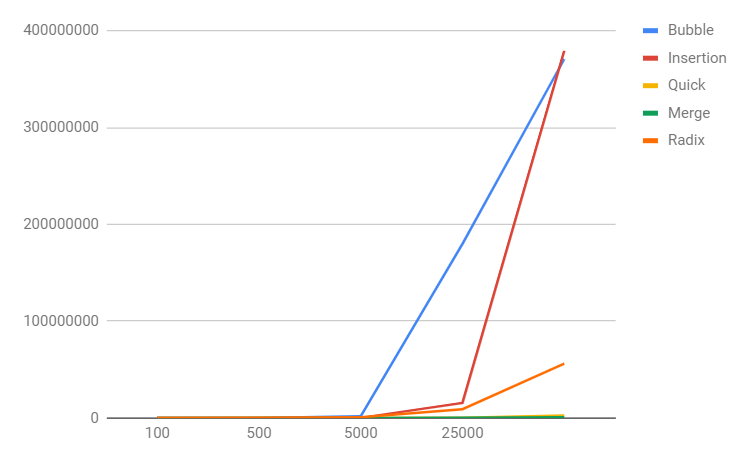


Figure 2: Graph of the data presented in Figure 1

1. Explain how well or poorly it matches your expectations for performance given the known Big O notation for the given sort algorithms
   1. Bubble – Worst case was O(n^2); it did take a long time but not as long as the quick sort, despite having the same big O notation for the worst case. It took about as long as the insertion sort. I expected them to be about the same.
   2. Insertion – Worst case was O(n^2); it did take a long time but not as long as the quick sort, despite having the same big O notation for the worst case.
   3. Merge – Worst case was that it had to sort every single item, and it felt like the speed did about that
   4. Quick – With a worst case scenario of O(n^2), the fact that it took the longest to finish makes a lot of sense; this was about what I expected (I counted manually and it took about 15s to complete the max array sorting size).
   5. Radix – Worked as well as I expected it to, given that it has to go through each element the number of times equal to the max number of digits in the largest number.
2. Expected for time for each of the array sizes based on the results for array size of 10.
   1. My expectations for the each of the array sizes based on the original array of size 10 were that they would increase by the new size divided by ten.
      1. Bubble – for the max, I expected about 3million ns and was off by a power of 10
      2. Insertion - for the max, I expected about 3million ns and got it
      3. Merge – unsure; expectation is in the above table.
      4. Quick– unsure; expectation is in the above table.
      5. Radix – Not anywhere close to the above statement; actually took less time.

Explaining the data in Figure 1:

I ran out of time before the submission was due/I had to get to class. In retrospect, I would have tried to pass the previous time as/with a pointer to each instance of the recursive method functions (Merge and Quick) rather than doing what I did, which was outputting the time per step and then having to physically add them (the italics are estimates based on the trends from the 1st 3 outputs, which I hand counted and added). While I did not plan to do that from the start, that is how it worked out by the time I needed to get ready to submit and come to class.

While no solution took over 5 minutes or ran out of memory, the way in which I wrote this code would probably struggle to detect that for the above mentioned methods. I did try with smaller cases and the code does detect those, output an error and abort the process, though.