CS590 Assignment 2

1. **Abstract**

This assignment is two parts; the first part finds recurrence function growth using the substitution method. The second part of this assignment explores the growth rate of two sorting functions with respect input size and input dimension. The results show that, for this set of string inputs, the radix with counting sort algorithm is across the board more performant (in duration) than radix with insertion sort. The radix insertion sort algorithm grows at O(n2), whereas the radix counting sort grows linearly. It is also shown that both algorithms grow linearly with string length when keeping the number of strings fixed. This matches the theoretical expectation, where radix sort grows at ϴ(dn), or ϴ(n) where n >> d.

1. **Solving Recurrences with Substitution**

*Please see the attached pdf*

1. **Results**

The two sorting algorithms are both “radix” sort algorithms, that differ in the underlying sort method. The two algorithms are a radix-insertion sort, and a radix-counting sort. The run time of these algorithms are measured for three different input lengths, each with 21 different input dimensions, n. Each configuration is run 10 times to reduce random variance. The results are shown below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **algorithm** | | **radix + counting sort** | | | **radix + insertion sort** | | |
| **string length** | | **25** | **35** | **45** | **25** | **35** | **45** |
| **number of strings** | **10** | 0 | 0 | 0 | 0 | 0 | 0 |
| **25** | 0 | 0 | 0 | 0 | 0 | 0 |
| **50** | 0 | 0 | 0 | 0 | 0 | 0 |
| **75** | 0 | 0 | 0 | 0 | 0 | 0 |
| **100** | 0 | 0 | 0 | 0 | 0 | 0 |
| **250** | 0 | 0 | 0 | 0 | 0 | 0 |
| **500** | 0 | 0 | 0 | 7 | 10 | 10 |
| **750** | 0 | 0 | 0 | 20 | 20 | 30 |
| **1000** | 0 | 0 | 0 | 30 | 45 | 59 |
| **2500** | 0 | 0 | 0 | 223 | 311 | 400 |
| **5000** | 0 | 10 | 10 | 1047 | 1479 | 1951 |
| **7500** | 10 | 10 | 20 | 2640 | 3745 | 4978 |
| **10000** | 10 | 20 | 27 | 5113 | 6871 | 8539 |
| **12000** | 20 | 20 | 20 | 7291 | 7767 | 9712 |
| **14000** | 17 | 20 | 24 | 7744 | 10349 | 12670 |
| **16000** | 10 | 20 | 29 | 9215 | 12420 | 16324 |
| **18000** | 15 | 20 | 25 | 11562 | 16307 | 25792 |
| **20000** | 20 | 24 | 30 | 18268 | 25167 | 30769 |
| **22000** | 20 | 20 | 30 | 22475 | 32935 | 43970 |
| **24000** | 22 | 22 | 30 | 28379 | 40773 | 52778 |
| **25000** | 20 | 31 | 47 | 31312 | 43773 | 57379 |

Table 1. Sort time (ms) with varying input conditions.

It’s apparent the major performance of counting sort over insertion sort.

1. **Analysis**

The results shown in Table 1 above are in agreement with theory. Counting sort operates at ϴ(n+k), where k represents the number of characters in the ‘set’. In our case, k=26, which gives us ϴ(n). When combining this with radix sort, this yields have ϴ(d(n))= ϴ(dn). For this experiment, the majority of the iterations had at least an order of magnitude more strings than digits per string. That is to say, n >> d. As a result, the final expected function growth rate is ϴ(n). This matches the data reasonably well; however, a better characterization of this function would include a much larger dataset. The counting sort performs so quickly, the measurements are near the nose floor.

The radix sort with insertion sort also matches the expectation. Insertion sort average performance is O(n2). Combined with radix sort, this gives us O(dn2), where d is the number of digits per string. As with the counting sort, n >> d, giving us our final O(n2). Looking at the data, the theory appears to match the measured performance. Both of these functions performance is plotted below, visually indicating the performance.

Chart, line chart

Description automatically generated

**Figure 1** Sort time vs varying input lengths for radix+insertion (blue) and radix+counting (red).

Finally, how the length of string inputs impacts run time for a fixed number of inputs is examined. To analyze this relationship, the percent change in run time when increasing the string length, keeping the number of strings in the data set constant is calculated. Next, the measurements are averaged in order to reduce the noise. Plotting the results in figure 2 below, both sort algorithms appear to have a linear relationship between string length and runtime. At first, this result might be surprising as insertion sort is O(n2), but one has to remember that in this scheme, insertion sort is only run at each location in the string, one offset at a time. Thus, by changing only string length, the growth rate falls back on the growth rate of radix sort portion, giving ϴ(n) performance.

Chart, line chart

Description automatically generated

**Figure 3** Insertion sort runtime dependency on vector dimension for naïve sort (red) and improved sort (blue).

1. **Conclusion**

This experiment shows off the power of radix sort, and in particular, counting sort. Counting sort often cannot be used to sort numerical data, as it both requires discrete data, and it requires a lot of storage space to sort larger numbers. However, this experiment shows the power of radix sort with counting sort for strings, as the character set (for lower case) is only 26 characters long. This same technique would work very well for a larger character set that included upper case and numerals. Compared to radix sort with insertion sort, there were cases where the counting sort algorithm performed more than 1000x faster. Finally, examining both algorithms growth rate in respect to string length shows that both functions grow linearly with n. This matches the theoretical growth rate, which is ϴ(n) for radix sort.