2-2-19

**Benefits of LSD Radix Sort for Managing Spreadsheets**

I have been hired by PNC Bank to develop and maintain a spreadsheet application for storing customer data, such as: account numbers, telephone numbers, names, and transaction info. For my development of this spreadsheet I have decided to use least significant digit (LSD) radix sort for the purpose of organizing the user data within the spreadsheet. To emphasize the benefits of LSD radix sort, it is crucial to analyze it in comparison to other sorting methods such as: insertion and bubble sort. Evaluating aspects of performance such as: efficiency and reliability, are important for comparing these sorting algorithms. Efficiency, has to do with the amount of time a certain sorting algorithm takes to sort a given group of items. Say in this case we were building the spreadsheet application to sort customer info by account number. Assuming this bank has somewhere around 10,000 customer records to store, LSD radix sort on average would perform a 1,000 times faster sort compared to both bubble and insertion sort. If the number of customer records increased to 100,000, LSD radix sort would perform a 10,000 percent faster sort compared to both bubble and insertion sort. This is clearly a major advantage for LSD radix sort as far as efficiency aspects go. Being that spreadsheets are normally used to store large databases of info, much like bank records, it’s important that LSD radix gains efficiency as the amount of items to sort increases. By saying it gains efficiency I mean in comparison to bubble and insertion sort. Reliability is the idea that a sorting algorithm sorts items in an order which we would expect and does so at fairly consistent speed. All three of these sorting methods will sort records in correct order, however, bubble and insertion sort will not always have a consistent speed. LSD radix sort, on the other hand, has a consistent runtime in all situations, making it a very reliable algorithm. LSD radix sort, will increase user experience for this spreadsheet application on the basis of efficiency and reliability.

**Formal Explanation of LSD Radix Sort**

In computer programming, it is a crucial skill to be able to point out and implement the sorting algorithm that is best for the job based on info given. The concepts of efficiency and reliability allow us as programmers to make this decision; this is ultimately how I decided that LSD radix sort is a much better algorithm for sorting spreadsheet data compared to insertion and bubble sort.

The characteristic of efficiency is more formally known as runtime of a program. I used the word efficiency throughout the first part of the paper, because I assumed that the term runtime would confuse a non-technical audience. Runtime is one of the most important aspects to developing a good program and is the most important reason as to why LSD radix sort performs better than both insertion and bubble sort. Since we are using this sorting algorithm to manage a spreadsheet application, I assumed that most spreadsheets (but not all) would consist of a large amount of data items to sort. For this reason, I gave the example of managing a spreadsheet application for customers of PNC bank because this would yield a large N value. N in this case is the size of the input array that we are trying to sort. The worst case runtime of LSD radix sort is O(WxN), where W is the length of the input (how many digits) we are trying to sort and N is the size of the input array. Essentially, this means that the runtime of LSD sort is linear-time for a vast majority of applications. This is because no matter how large the value of N, it will only make W passes through the data. The only time this won’t be the case is if W is a higher value than N, cause then you would be looking at more values of W than there are for N. However, for the purpose of managing a spreadsheet application this wouldn’t really become a problem because spreadsheets normally store data with a relatively small W value.

- Worst Case Runtime of Bubble Sort(array is sorted in descending order):

(n-1) + (n-2) + (n-3) + (n-4) … + 2 + 1 = (n\* (n-1))/2 = O(n2) because in the worst case n-1 comparisons will be done in first pass, n-2 comparisons in second pass, etc…

-Worst Case Runtime of Insertion Sort (array is sorted in reverse order):

(n) + (n-1) + (n -2) … + 2 + 1 = (n \* (n - 1))/2 = O(n2) because in the worst case insertion sort does ~N2/2 compares and ~N2/2 exchanges

-Runtime comparison for storing n = 10,000 customer records:

-Bubble and Insertion Sort: O(10,0002) = O(100,000,000)

-LSD radix sort (W = 10): O(10 \* 10,000) = O(100,000)

-Runtime comparison for storing n = 100,000 customer records:

-Bubble and Insertion Sort: O(100,0002) = O(10,000,000,000)

-LSD radix sort (W = 10): O(10 \* 100,000) = O(1,000,000)

Expanding on the bank account example, I compared the runtime of LSD radix sort to those of insertion and bubble sort for sorting user information by bank account number. All PNC bank account numbers are ten digits which is why W = 10 in this case. For storing 10,000 customer records I found that LSD sort was O(100,000,000)/O(100,000) = 1,000 times faster and for 100,000 customer records I found that LSD sort was O(10,000,000,000)/O(1,000,000) = 10,000 times faster. One reason why LSD radix sort has such a better performance than bubble and insertion sort is because it sorts data by its individual digits and not by the value of the data, making it a non-comparison based sort.

Finally, LSD radix sort is superior to bubble and insertion sort because it has a reliable runtime. By this, I mean that the worst, average, and best case runtime for LSD radix sort are all O(W\*N). This provides a convincing argument against bubble and insertion sort who both have an average and worst case of O(n2) and a rare best case of Ω(n).