Gamez, David

80553747

CS2302

Prof. Olac Fuentes

Lab2 Report

Lab2 taught us how we can sort data in a list using many different approaches. During this lab, not only did we learn the different methods, but also all the advantages and disadvantages of using these methods. The different approaches to sorting data that we learned were, bubbleSort, mergeSort, quickSort, and a modified quickSort.

The first method we worked on was bubbleSort. The best-case runtime scenario for bubbleSort is O(n), its worst case is O(n^2). This approach will traverse through the whole list and switch up the two neighboring elements if they are not sorted (if n is less that n-1 they will be switched). The best case of bubbleSort is when the list is already sorted In which case it will to comparisons n amount of times. Its worst case is when the first smallest element is at the very end of the list. BubbleSort would have to make many traversals through the list before getting the smallest element into its right spot.

The second method we worked on was mergeSort. MergeSort is a recursive divide and conquer approach to sorting data. You must first divide the list in half over and over until you can make easy comparisons. The list will be broken down in half until there is only one element in each list and then you can compare the two lists easily. MergeSort is one of the best techniques out there to sort data, because its best case runtime is O(n(log n)) and so is its worst case scenario. This algorithm is very useful for both arrays and linked lists. The divide and conquer approach slashes the runtime in half as much as bubbleSort, because it does not need to traverse through the list to make these comparisons. That is also the beauty of recursion. We can sort of leave the problem on auto pilot as the recursive calls reach the base case on their own.

The third method we learned is quicksort. This method also uses a recursive divide and conquer technique to sort the data more easily. The difference between this approach and the last is that in quicksort we use a “pivot” this is an arbitrary element that we can choose. The pivot may be the first, last or a random element in the list. Once we pick a pivot we then compare that pivot to every other element in the list. If the element is lower than the pivot, then the element goes into a list filled with elements smaller than the pivot. If the elements are bigger they all go into a list containing all elements larger than the pivot. We then apply that same technique to all the lists until we have broken down the problem into single element lists which we can then compare easily, and merge together already sorted.

The runtime of quicksort is O(n(logn)) on its best case scenario.

Now we will compare the actual amount of instructions that each technique uses. To compare all of these techniques we will use recurrence relations to compare the amount of computations each one executes. The one with the least amount of computations executed is the fastest algorithm. For this comparison we will use the worst-case scenario for each.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sort Type | Bubble Sort | MergeSort | QuickSort | ModQuickSort |
| O(n) | O(n^2) | O(nlogn) | O(n^2) | O(nlogn) |
| N | 40, 80, 100 | 40, 80, 100 | 40, 80, 100 | 40, 80, 100 |
| Total Number of Computations (Respectively) | 1600, 6400, 10000 | 200, 520, 680 | 1600, 6400, 10000 | 200, 520, 680 |

Based on the results of the test we can conclude that the best sorting methods using linked lists are mergeSort and Modified quickSort. The both have the best worst-case scenario runtimes. The reason why they have the best run times is because they use the divide and conquer technique which will cut down the time by a lot. The reason why we want the most time efficient methods is because we do not want to waste precious memory space and of coarse loading time. The best kind of software is the kind that cleverly organized the data and in this case the divide and conquer approach has been the best technique.