Quinn Roemer

CISP - 430

Assignment 8

4/5/2018

Part 0 - HeapSort Hand Execution

Description:

The goal for this assignment was to hand execute several new sorting algorithms. I was to create several detailed computer-drawn diagrams that outlined and described how the function worked. The first diagram I tackled was that of Heapsort. This sorting method works by dumping the data into a Binary Heap and then deleting off the Heap. This orders the data.

HeapSort: (90, 60 ,30 ,35 ,40 ,45 ,85, 55 ,50 ,75 ,80 ,70 ,25 ,20 ,65 ,10)

Note: This diagram is split over two diagrams. The first diagram covers the insertion into the Binary Heap. The second, the deletion from the Binary Heap. In addition both diagrams are large and will take 4 and 8 pages respectively.

Part 1 - QuickSort Hand Execution

Description:

The next section of this assignment required me to diagram the QuickSort algorithm in great detail. This algorithm works by partitioning the array in such a way that splits it into sub-arrays. These sub-arrays are then sorted using the same method that was used to create them. In this case that will be QuickSort.

QuickSort: (90, 60 ,30 ,35 ,40 ,45 ,85, 55 ,50 ,75 ,80 ,70 ,25 ,20 ,65 ,10)

Note: This diagram is split over two diagrams. The first diagram covers the sorting of the actual array. The second, the recursive tree created by the qsort() function. In addition both diagrams are large and will take 6 and 6 pages respectively.

Part 2 - MergeSort Hand Execution

Description:

The last sorting algorithm that I was required to diagram was MergeSort. This algorithm works by splitting the array in half numerous times until it has one element sub-arrays. It then combines the adjacent sub-arrays in an ordered fashion until the array is back to its original size but with ordered data.

MergeSort: (90, 60 ,30 ,35 ,40 ,45 ,85, 55 ,50 ,75 ,80 ,70 ,25 ,20 ,65 ,10)

Note: This diagram is split over two diagrams. The first diagram covers the sorting of the actual array. The second, the recursive tree created by the msort() function. In addition both diagrams are large and will take 3 and 6 pages respectively.

Part 3 - BigO of Each Sorting Algorithm

HeapSort:

The time complexity of HeapSort must be O(nlogn). This is because one insertion into the Binary Heap requires O(logn) time and we must perform N insertions. In addition, each deletion takes O(logn) time. However, we must perform N deletions. Thus the final time complexity would be O(2*nlogn) which is just O(nlogn) time.

QuickSort:

On average this algorithm calls the partition() function (where the work is done) O(logn) times. However, just like HeapSort this algorithm must perform this N times. This results in a final time complexity of O(nlogn). Worthy of note, this algorithm can perform much worse if it makes a bad choice as its pivot point. If this is the case it will have an overall time complexity of O(n^2) due to it having to call partition() N times.

MergeSort:

Just like HeapSort and QuickSort this algorithm does O(logn) work at each level of the sort. In other words, it takes the algorithm O(logn) time to perform each split. However, this split is proportional to how many items there are in the sort. Thus the final time complexity would be O(nlogn).

Conclusion

This assignment took me some time. I probably spent a total of 9 hours working on the diagramming for this assignment. I put my Spring Break to good use. When I began this assignment I was slightly confused on how some of the algorithms work. But by painstakingly diagramming them these confusions were resolved. One thing that I found interesting as I was delving deeper into these algorithms was the difference between QuickSort and the other two sorting algorithms. After some research I found that QuickSort is more widely used since it does not require any extra memory to perform. Unlike the other two, QuickSort does not need a temporary location to store the data it is sorting. Looking forward to the next assignment!