CTEC 243 Data Structures

Jeffrey Walley | 11/29/2024

Sorting and Searching Algorithms

1. Describe the similarities and differences between how Selection Sort works and how Bubble Sort works.

When bubble sorting we compare adjacent items starting with the item at index 0 and 1, the larger of the two values ‘bubbles’ to the right and then index 1 is compared to index 2. This process continues until the largest value eventually makes its way to the end. With the largest value now at the end the 2nd iteration begins, and the process starts over making certain that the largest value is now next to the end and algorithm has now sorted the 2 largest values into their correct positions at the end. Bubble sort will continue to iterate through elements until items are sorted correctly, with the larger values to the right.

For a selection sort we take the item at the first index and then compare it with each of the other items one-by-one. We consider the 0-index value to be the minimum item value and compare it against each additional item, if we find a smaller value then we will swap that value to the 0-index. As the algorithm sorts through the values then the minimum values are sorted left.

Both bubble sort and selection sort are considered slow for large datasets, both with O(N^2). Both algorithms are comparison-based. Bubble sort will end up with more swaps in most scenarios and will require more steps to complete.

1. Describe the similarities and differences between how Bubble Sort works and how Insertion Sort works.

The Insertion Sort varies from the selection sort we discussed before and the bubble sort, in that Insertion sort builds its sorted group of values as it goes through comparisons. First, insertion sort looks at index-0, the item farthest to the left, since that is the only value so far, that is considered sorted and value at index-0 is considered in the correct spot. The algorithm then moves to index-1, let’s say for example’s sake this item is smaller than index-0, the algorithm swaps the 2 values and considers both index-0 and index-1 to be sorted. As we iterate through the values each index is marked as sorted, if a smaller value is encountered the algorithm compares with the items before it until it ‘inserts’ the new value into the correct pre-sorted position.

Both bubble sort and insertion sort compare items to sort through values and both have an O(N^2) time complexity in worst case. Insertion sort requires fewer steps than bubble sort and would therefore be more efficient in most cases.

1. Describe the similarities and differences between how Insertion Sort works and how Merge Sort works.

Insertion sort builds it’s ordered group as it goes element by element, it is a slower sorting method than merge sort. Merge sort is a ‘divide-and-conquer’ algorithm which splits the group into smaller sub-groups, compares the values in those subgroups and correctly sorts the small groups. Then the algorithm merges the subgroups, compares that values are sorted correctly and makes any changes that need to be completed, before re-merging the sub-groups again, until the values are sorted in their correct index position and the set is fully merged.

Merge sort has an O(n log n) time complexity and is useful for larger datasets, while Insertion sort is very useful for smaller datasets.

1. Describe the similarities and differences between how Merge Sort works and how Quick Sort works.

Both merge sort and quick sort are ‘divide-and-conquer’ algorithms and both have a time complexity of O(n log n), making them ideal for larger datasets. Whereas merge sort divides the original group in half, and into half again until values are split into groups of pairs, the algorithm compares those values, sorts them correctly and merges the sub-groups back comparing and sorting each time before merging into larger groups again.

Quick sort on the other hand first assigns a ‘pivot’ value from the group and then compares that value to the other elements. Values that are less than the pivot will be moved to the left of the pivot, and items that are greater than the pivot go to the right. Quick sort then recursively sorts the partitions. Although fast quick sort’s efficiency falls if the programmer chooses a bad ‘pivot’, then the complexity can become O(n^2). Merge sort will require more memory than quick sort but guarantees even performance.