**Course: C339 Data Fundamentals**

**Date: March 3, 2023**

**Title: Sorting Algorithms and Space Complexity**

**Bubble Sort**

Bubble sort is a simple sorting algorithm. This sorting algorithm is a comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of Ο(n^2) where n is the number of items.

How does bubble sort work?

|  | Bubble sort starts with the first two elements (i.e. 14 and 33), comparing them to check which is bigger. |
| --- | --- |
|  | In this case, 33 is bigger than 14. So, it is already sorted. Next, we compare 33 and 27. |
|  | Here, 27 is smaller than 33. So they must be swapped. |
|  | Next, we compare 33 and 35. Here, 35 is bigger than 33 so they are already sorted. |
|  | Next, we compare 35 and 10. Here, 10 is smaller than 35 so their positions need to be swapped.  After one iteration, 10 would be at the left of 35. |
|  | After each iteration, 10 keeps moving towards the left as it is smaller to the values to its right. |
|  | And when there's no swap required, bubble sorts learns that an array is completely sorted. |

**Quick Sort**

Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.

Quicksort partitions an array and then calls itself recursively twice to sort the two resulting subarrays. This algorithm is quite efficient for large-sized data sets as its average and worst-case complexity are O(n^2), respectively.

Quick Sort Pivot Algorithm

Based on our understanding of partitioning in quick sort, we will now try to write an algorithm for it, which is as follows.

Step 1 − Choose the highest index value has pivot

Step 2 − Take two variables to point left and right of the list excluding pivot

Step 3 − left points to the low index

Step 4 − right points to the high

Step 5 − while value at left is less than pivot move right

Step 6 − while value at right is greater than pivot move left

Step 7 − if both step 5 and step 6 does not match swap left and right

Step 8 − if left ≥ right, the point where they met is new pivot

Quick Sort Algorithm

Using pivot algorithm recursively, we end up with smaller possible partitions. Each partition is then processed for quick sort. We define recursive algorithm for quicksort as follows −

Step 1 − Make the right-most index value pivot

Step 2 − Partition the array using pivot value

Step 3 − Quicksort left partition recursively

Step 4 − Quicksort right partition recursively

**Merge Sort**

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being Ο(n log n), it is one of the most respected algorithms.

Merge sort first divides the array into equal halves and then combines them in a sorted manner.

How does merge sort work?

|  | To understand merge sort, we take an unsorted array as shown to the left. |
| --- | --- |
|  | We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4. |
|  | This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves. |
|  | We further divide these arrays and we achieve atomic value which can no more be divided. |
| Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists. | |
|  | We first compare the element for each list and then combine them into another list in a sorted manner.  We see that 14 and 33 are in sorted positions.  We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27.  We change the order of 19 and 35, whereas 42 and 44 are placed sequentially. |
|  | In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order. |
|  | After the final merging, the list should look like this − |

**Insertion Sort**

**Space complexity**

Space complexity refers to the total amount of memory space used by an algorithm/program, including the space of input values for execution.

For example, if we want to compare standard sorting algorithms on the basis of space, then Auxiliary Space would be a better criterion than Space Complexity. Merge Sort uses O(n) auxiliary space, Insertion sort, and Heap Sort use O(1) auxiliary space. The space complexity of all these sorting algorithms is O(n) though.

Space complexity is a parallel concept to time complexity. If we need to create an array of size n, this will require O(n) space. If we create a two-dimensional array of size n\*n, this will require O(n2) space.

In recursive calls stack space also counts.

Note: Space complexity depends on a variety of things such as the programming language, the compiler, or even the machine running the algorithm.