

**Bubble sort algorithm**



**Manal alsallal Alqadri**

What is the bubble sort algorithm?

The bubble sort algorithm is a simple sorting algorithm that repeatedly steps through a list of elements, compares adjacent elements, and swaps them if they are in the wrong order. The process is repeated until the entire list is sorted.

How the bubble sort algorithm works:

1. Start with an unsorted list of elements.

2. Compare the first and second elements. If the first element is greater than the second element, swap them.

3. Move to the next pair of elements (the second and third elements) and compare them. Again, if the second element is greater than the third element, swap them.

4. Continue this process, comparing and swapping adjacent elements, until you reach the end of the list.

5. Once you reach the end of the list, the largest element will have "bubbled" to the end (i.e., it will be in its correct sorted position).

6. Repeat steps 2-5 for the remaining unsorted portion of the list, excluding the last element which is already in its correct position.

7. Keep repeating the process until the entire list is sorted.

The name "bubble sort" comes from the way larger elements gradually "bubble" to the end of the list during each iteration.

Bubble sort is straightforward to implement and understand, but it is not an efficient algorithm for large lists. It has an average and worst-case time complexity of O(n^2), where n is the number of elements in the list. This means that the time it takes to sort the list grows quadratically with the number of elements. Therefore, bubble sort is not recommended for sorting large datasets. However, it can be useful for educational purposes or for sorting small lists with a known limited size

The time complexity of bubble sort in more detail:

The time complexity of the bubble sort algorithm is an important factor to consider when analyzing its efficiency. The time complexity describes how the algorithm's performance scales with the size of the input.

In the case of bubble sort, the worst-case time complexity is O(n^2), where n is the number of elements in the list. This means that the time it takes to sort the list grows quadratically with the number of elements.

To understand why the worst-case time complexity is O(n^2), let's consider a scenario where the list is in reverse order.

In each pass, the largest element in the unsorted portion of the list is "bubbled" to its correct position at the end. In the worst case, this requires n-1 comparisons and swaps for each element, resulting in (n-1) + (n-2) + ... + 2 + 1 = n(n-1)/2 comparisons and swaps.

This arithmetic series simplifies to (n^2 - n)/2, which is still in the order of n^2. Therefore, the worst-case time complexity of bubble sort is O(n^2).

It's worth noting that the best-case time complexity of bubble sort is O(n) when the list is already sorted. In this case, the algorithm only needs to make a single pass through the list to confirm that it's sorted. However, bubble sort's average-case time complexity is still O(n^2), making it inefficient for large lists.

Overall, bubble sort is not recommended for sorting large datasets due to its relatively slow performance compared to more efficient sorting algorithms like quicksort or merge sort, which have average-case time complexities of O(n log n).

the time complexity analysis of bubble sort and explore its best-case, worst-case, and average-case:

1. Best-case time complexity:

In the best-case scenario, the input list is already sorted. Bubble sort will still perform a pass through the list to confirm

that it's sorted, but no swaps will be necessary. This results in a time complexity of O(n), where n is the number of elements. However, even in the best case, bubble sort still performs unnecessary iterations, making it less efficient compared to other sorting algorithms.

2. Worst-case time complexity:

The worst-case scenario occurs when the input list is in reverse order. In each pass, the largest element in the unsorted portion of the list needs to be moved to its correct position at the end. As mentioned earlier, this requires (n-1) + (n-2) + ... + 2 + 1 = n(n-1)/2 comparisons and swaps. This simplifies to O(n^2) time complexity.

3. Average-case time complexity:

The average-case time complexity of bubble sort is also O(n^2). This means that, on average, bubble sort will take a quadratic number of operations to sort the list. The exact average-case time complexity analysis involves considering all possible permutations of the input list and calculating the average number of comparisons and swaps required. The detailed derivation is beyond the scope of this discussion, but it suffices to say that bubble sort's average-case performance is not efficient for large lists.

Bubble sort's time complexity, particularly its worst-case and average-case complexities, make it inefficient for sorting large datasets. As the number of elements increases, bubble sort's performance deteriorates quickly. Other sorting algorithms, such as quicksort, merge sort, and heapsort, offer better average and worst-case time complexities, making

them more suitable for large-scale sorting tasks.

**Advantages and disadvantage**

1.The primary advantage of the bubble sort is that it is popular and easy to implement.

2.The main disadvantage of the bubble sort is the fact that it does not deal well with a list containing a huge number of items.

3.In the bubble sort, elements are swapped in place without using additional temporary storage.

4.The bubble sort requires n-squared processing steps for every n number of elements to be sorted.

6.The space requirement is at a minimum