Teacher's Explanation:

Alright class, today we're going to learn about **Bubble Sort**, which is one of the simplest sorting algorithms used in computer science.

Now, let's break it down. **Sorting** simply means arranging a collection of items (numbers, words, etc.) in a particular order, either ascending (from smallest to largest) or descending (from largest to smallest). For example, when you organize your books from the smallest to the largest, you're sorting them.

Now, how does **Bubble Sort** work? Imagine you have a list of numbers, and your job is to arrange them from the smallest to the largest. **Bubble Sort** does this by repeatedly comparing each pair of adjacent numbers and swapping them if they're in the wrong order. We repeat this process until no more swaps are needed, meaning the list is fully sorted.

Let's go through an example to make it clearer:

Suppose you have the following list of numbers:
`[5, 2, 9, 1, 5, 6]`

- **First Pass**:
  - Compare 5 and 2 → since 5 > 2, swap them → `[2, 5, 9, 1, 5, 6]`
  - Compare 5 and 9 → since 5 < 9, no swap → `[2, 5, 9, 1, 5, 6]`
  - Compare 9 and 1 → since 9 > 1, swap them → `[2, 5, 1, 9, 5, 6]`
  - Compare 9 and 5 → since 9 > 5, swap them → `[2, 5, 1, 5, 9, 6]`
  - Compare 9 and 6 → since 9 > 6, swap them → `[2, 5, 1, 5, 6, 9]`
- After the first pass, the largest number, 9, "bubbled up" to the end of the list.
- **Second Pass**:
  - Compare 2 and 5 → no swap → `[2, 5, 1, 5, 6, 9]`
  - Compare 5 and 1 → swap → `[2, 1, 5, 5, 6, 9]`
  - Compare 5 and 5 → no swap → `[2, 1, 5, 5, 6, 9]`
  - Compare 5 and 6 → no swap → `[2, 1, 5, 5, 6, 9]`
- After the second pass, the second-largest number, 6, has bubbled up to its correct position.
- **Third Pass**:
  - Compare 2 and 1 → swap → `[1, 2, 5, 5, 6, 9]`
  - Compare 2 and 5 → no swap → `[1, 2, 5, 5, 6, 9]`
- Now, the list is sorted! We stop the sorting process once we go through the entire list and no swaps are needed.

Key points to remember:

1. **Bubble Sort** works by repeatedly swapping adjacent elements that are out of order.
2. After each pass through the list, the largest unsorted element is placed at its correct position.
3. It continues this process until no swaps are needed, meaning the list is sorted.
4. The worst-case time complexity of Bubble Sort is **O(n²)**, which makes it inefficient for large lists, but it's very simple and easy to understand.

Student's Notes:

Bubble Sort Algorithm

Introduction:

- **Bubble Sort** is a simple comparison-based sorting algorithm.
- It works by comparing adjacent elements and swapping them if they are in the wrong order.
- It continues this process for each element until the entire list is sorted.

How Bubble Sort Works:

- The algorithm repeatedly steps through the list, compares adjacent items, and swaps them if they are in the wrong order.
- The process is repeated until the list is sorted. The largest unsorted element "bubbles up" to its correct position after each pass through the list.
- It is called **Bubble Sort** because smaller elements "bubble up" to the top (or start) of the list, while larger elements sink to the bottom (or end).

Algorithm Steps:

1. Start with the first element in the list.
2. Compare the current element with the next element.
    - If the current element is greater than the next, swap them.
    - If not, move to the next pair.
3. Continue this for the entire list. After the first pass, the largest element will be placed at the end of the list.
4. Repeat the process for the rest of the list, ignoring the last element (since it's already sorted).
5. Stop when no swaps are needed, indicating the list is sorted.

Example: Let's take the list of numbers: `[5, 2, 9, 1, 5, 6]`

- **First Pass:**
    1. Compare 5 and 2 → swap → `[2, 5, 9, 1, 5, 6]`
    2. Compare 5 and 9 → no swap → `[2, 5, 9, 1, 5, 6]`
    3. Compare 9 and 1 → swap → `[2, 5, 1, 9, 5, 6]`
    4. Compare 9 and 5 → swap → `[2, 5, 1, 5, 9, 6]`
    5. Compare 9 and 6 → swap → `[2, 5, 1, 5, 6, 9]`
- After the first pass, 9 is at its correct position.
- **Second Pass:**
    1. Compare 2 and 5 → no swap → `[2, 5, 1, 5, 6, 9]`
    2. Compare 5 and 1 → swap → `[2, 1, 5, 5, 6, 9]`
    3. Compare 5 and 5 → no swap → `[2, 1, 5, 5, 6, 9]`
    4. Compare 5 and 6 → no swap → `[2, 1, 5, 5, 6, 9]`
- After the second pass, 6 is at its correct position.
- **Third Pass:**
    1. Compare 2 and 1 → swap → `[1, 2, 5, 5, 6, 9]`
    2. Compare 2 and 5 → no swap → `[1, 2, 5, 5, 6, 9]`
- The list is now fully sorted: `[1, 2, 5, 5, 6, 9]`.

Time Complexity:

- The worst-case time complexity of Bubble Sort is **O(n²)**, where **n** is the number of elements in the list.
    - This is because, in the worst case, each element is compared with every other element.
    - The best-case time complexity is **O(n)**, which occurs if the list is already sorted, as no swaps will be needed.

Space Complexity:

- The space complexity of Bubble Sort is **O(1)**, because it only requires a constant amount of extra space to store temporary variables for swapping elements.

Advantages:

- **Simple to implement** and understand.
- Useful for small datasets or when sorting is not a frequent task.

Disadvantages:

- **Inefficient** for large datasets due to its O(n²) time complexity.
- Other algorithms like **Quick Sort** or **Merge Sort** are preferred for larger datasets.

Conclusion: Bubble Sort is a basic sorting algorithm that, although inefficient for large lists, provides a clear and easy-to-understand method for sorting data. It's best used for teaching sorting concepts and for small or nearly sorted lists.