Lesson Plan: Introduction to Bubble Sort

Overview

This is a short, 30-minute lesson designed to introduce students to the basic logic of sorting algorithms using Bubble Sort. It's a foundational activity that can be easily extended or modified to fit different class lengths or depths.

1. Introduction

Begin by introducing sorting as a problem computers solve often—ranking data, ordering search results, etc. Explain that this lesson covers one of the simplest algorithms: **Bubble Sort**.

Use an everyday analogy (optional): imagine sorting items by comparing only two at a time and swapping if needed. Emphasize that while this approach is simple, it's not the fastest—but it's a great starting point for understanding how sorting works.

2. Build a Sample List

Work with the class to create a short, unsorted list of numbers (e.g., four values). You can gather suggestions from students or use this example:

[4, 1, 3, 2]
The goal is to sort the list in ascending order using Bubble Sort.

3. Bubble Sort Walkthrough

Walk through the sorting process, pass by pass, emphasizing comparisons and swaps.

Example with [4, 1, 3, 2]:

Pass 1:
4 > 1 → swap → [1, 4, 3, 2]
4 > 3 → swap → [1, 3, 4, 2]
4 > 2 → swap → [1, 3, 2, 4]

Pass 2:
1 < 3 → no swap
3 > 2 → swap → [1, 2, 3, 4]

Pass 3:
1 < 2 → no swap → Sorted

Highlight that with each full pass, the largest unsorted number "bubbles" to its correct position.

4. Student Activity

Distribute a new unsorted list to students.

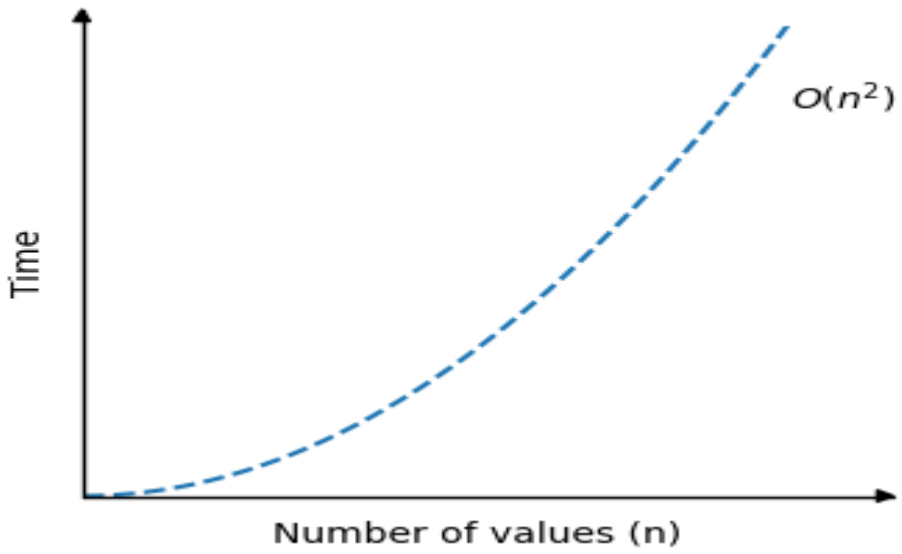
If available, **hand out 3D printed materials** to represent sortable elements (optional but helpful for tactile learners).

Students can work individually, in pairs, or small groups to apply Bubble Sort using paper, whiteboards, or hands-on materials. Encourage them to track each pass and swap.

5. Wrap-Up

Recap the key takeaways:

- Bubble Sort works by repeatedly comparing and swapping adjacent values.
  It's easy to understand but inefficient for large lists.
- The algorithm runs in O(n²) time—useful for teaching but not optimal for performance.
- Show students this chart to demonstrate the algorithm's runtime complexity. (Optional)



Introduce other efficient sorting methods (e.g., Insertion Sort,, Quick Sort, Merge Sort) as next steps.(optional)

6. Optional Code Example

For classes ready to see the algorithm in code (e.g., in Python):

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

my_list = [4, 1, 3, 2]
bubble_sort(my_list)
print(my_list)
```

Common Student Questions

Q: Why learn Bubble Sort if it's not efficient?
A: It builds understanding of comparisons, swaps, and algorithm logic which is great for beginners.

Q: What makes it slow?
A: It keeps comparing all values even if most of the list is already sorted.

Q: What's a better algorithm?
A: Merge Sort and Quick Sort are both much faster (O(n log n)) and used in real-world software.

Q: Will we learn those too?
A: Yes, this lesson is the foundation for those more advanced topics.

Q: Can we create our own sort?
A: Absolutely. Designing or modifying sorting strategies is a great way to deepen algorithmic thinking.

NOTE: A Bubble Sort activity handout using 3D printed props is included below. In this version, students are instructed to remove all 11 bars from the container, randomly select 5, and place them back in an unsorted order. This fits within a short 20-minute lesson. For a longer class period, you can extend the activity by having students sort all 11 bars or use the same materials to introduce a more efficient algorithm, such as Merge Sort or Quick Sort.

Bubble Sort Activity

Objective: You'll learn how the Bubble Sort algorithm works by physically sorting **bars of different heights**.

- **Taller bar = higher value**
- **Shorter bar = lower value**

Your goal: sort the bars from **shortest to tallest** (lowest to highest value).

What is Bubble Sort?

Bubble Sort compares and swaps items one step at a time, always working **left to right**. After each full pass, the **tallest unsorted bar moves to the end** — where it should go. Repeat passes until everything is sorted.

Instructions

1. Remove the bars from the container and place them back in a randomly unsorted order.
2. Start with the **first two bars on the left**.
3. Compare them:
    - If the **left bar is taller**, lift it up (your **temporary box**), slide the shorter bar to the left, and place the taller bar on the right.
    - Otherwise, if the **left bar is shorter**, do **not swap** — just hold it in place and move on.
4. **After each comparison, continue with the bar that ended up on the right**:
    - If a **swap happened**, compare the **new right-side bar** with the next bar.
    - If **no swap happened**, compare the **same left bar** with the next bar.
5. You are always moving through the row by comparing the current bar with the one directly next to it, until you reach the end.
6. At the end of the pass, the **tallest bar should be at the far right**. Meaning it's now sorted.
7. Go back to the start and repeat the process with the remaining unsorted bars.
8. Keep going until the bars are completely sorted from **shortest to tallest**.

Quick Reminders

- Always compare **side-by-side bars**, left to right.
- Only **swap** if the left bar is taller than the right.
  Use a **temporary hold** (your hands) when swapping bars.
- Each pass places one more bar into its final position on the right.
- It's okay if it takes a few rounds. This is about understanding the process!