Bubble Sort

Bubble sort is a sorting algorithm that works by repeatedly iterating through a list and comparing adjacent elements. If the elements are in the wrong order, they are swapped. This process is continued until no swaps are needed, indicating that the list is sorted.

Here's a detailed explanation of the bubble sort algorithm:

**How it Works:**

1. **Iterate Through the List:** The algorithm starts by iterating through the list. In each iteration, it compares adjacent elements.
2. **Compare Adjacent Elements:** For each pair of adjacent elements, the algorithm compares them. If the first element is greater than the second element, they are swapped.
3. **Swapping Elements:** If a swap is needed, the larger element is moved to the right position in the list.
4. **Repeat Until Sorted:** This process of iterating, comparing, and swapping is repeated for all adjacent elements in the list. After one complete pass through the list, the largest element will be "bubbled" to the end of the list.
5. **Multiple Iterations:** The entire process (iterations through the list) is repeated multiple times until no swaps are needed in a complete pass. This indicates that the list is sorted.

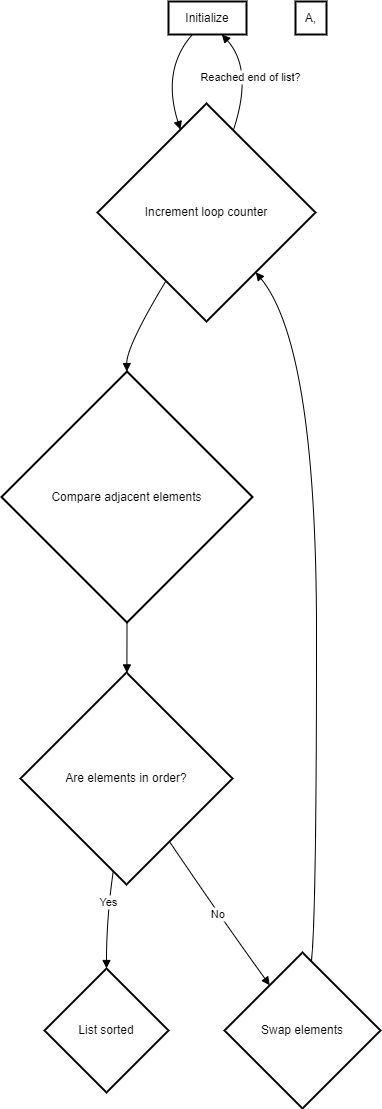
**Visualizing Bubble Sort:**

Imagine the elements in the list like bubbles in water. The larger elements, like bigger bubbles, tend to rise to the surface (end of the list) during each iteration.

**Why Bubble Sort?**

Bubble sort is a simple and easy-to-understand sorting algorithm. It's often used as a first introduction to sorting algorithms due to its simplicity.

However, bubble sort is not very efficient for large datasets. Its time complexity is O(n^2), which means the sorting time increases significantly as the number of elements in the list grows. There are more efficient sorting algorithms available for practical use cases.



Binary Search:

Binary search is a search algorithm that excels at finding the position of a target value within a **sorted array**. It works by repeatedly dividing the search interval in half and comparing the target value with the middle element. This "divide and conquer" approach significantly reduces the time complexity compared to linear search.

Here's a breakdown of the binary search algorithm:

**Core Idea:**

1. Start with the entire sorted array as the search interval.
2. Identify the middle element of the interval.
3. Compare the target value with the middle element:
   * If they match, the target element is found at the middle index.
   * If the target value is less than the middle element, eliminate the right half of the interval and repeat steps 2 and 3 on the left half.
   * If the target value is greater than the middle element, eliminate the left half of the interval and repeat steps 2 and 3 on the right half.

