**BUBBLE SORT**

In code

**What Happens in Each Step:**

1. **Outer Loop (for (int i = 0; i < length - 1; i++))**:
   * The outer loop runs for each "pass" through the array.
   * With each pass, the largest unsorted element bubbles to the correct position in the array, so the number of elements to compare (length - i - 1) decreases by 1 on each subsequent pass.
2. **Inner Loop (for (int j = 0; j < length - i - 1; j++))**:
   * The inner loop goes through the array and compares adjacent elements.
   * The comparison happens between arr[j] and arr[j + 1].
3. **Comparison (if (arr[j] < arr[j + 1]))**:
   * This condition checks if the element at index j is smaller than the element at index j + 1.
   * **Descending Order**: If arr[j] < arr[j + 1] is true, it means the elements are out of order (since you want to sort in descending order), so a swap is required.
4. **Swapping (long temp = arr[j]; arr[j] = arr[j + 1]; arr[j + 1] = temp;)**:
   * **Step 1**: Store arr[j] in a temporary variable temp. This ensures the current value at index j is saved before it's replaced.
   * **Step 2**: Assign the larger value arr[j + 1] to arr[j], effectively moving the larger element to the left.
   * **Step 3**: Assign the smaller value stored in temp to arr[j + 1], completing the swap. Now, the smaller element is moved to the right.

**Example Walkthrough:**

For an array {5, 1, 4, 2, 8}, after the first pass:

* **Before Comparison**:
  + Compare 5 and 1 → no swap needed, continue.
  + Compare 1 and 4 → swap them (array becomes {5, 4, 1, 2, 8}).
  + Compare 1 and 2 → swap them (array becomes {5, 4, 2, 1, 8}).
  + Compare 1 and 8 → swap them (array becomes {5, 4, 2, 8, 1}).

After the first pass, the largest value 8 is now correctly placed at the end of the array.

* **Second Pass**:
  + Now, we ignore the last element (since it's sorted) and go through the remaining part of the array.
  + Continue this process until the array is fully sorted in descending order.

**Summary:**

* The code implements **Bubble Sort** in **descending order**.
* The key operation happens in the **swap** section: elements are swapped if they're out of order based on the comparison condition (arr[j] < arr[j + 1] for descending order).
* After all the passes, the largest elements will be "bubbled up" to the beginning of the array (for descending sort), resulting in a fully sorted array.

**Selection Sort**

**What Happens in Each Part of the Code:**

1. **Outer Loop (for (int i = 0; i < length - 1; i++))**:
   * This loop tracks which part of the array is being sorted. At each iteration, the code will find the largest (or "maximum") unsorted element and place it at the i-th position, progressively sorting the array in **descending order**.
2. **Finding the Maximum (int maxIdx = i)**:
   * The variable maxIdx is used to track the index of the largest element found so far.
   * Initially, maxIdx is set to i, assuming that the element at index i is the largest in the unsorted portion of the array.
3. **Inner Loop (for (int j = i + 1; j < length; j++))**:
   * The inner loop looks through the unsorted portion of the array, starting at j = i + 1. It checks if any element after index i is larger than the element at maxIdx.
   * The condition if (arr[j] > arr[maxIdx]) ensures that the largest element in the remaining unsorted part is found. If a larger element is found, maxIdx is updated to the index j.
4. **Swapping the Largest Element**:
   * After the inner loop finishes, maxIdx holds the index of the largest element in the unsorted portion.
   * The code swaps the element at index i (which is where the next largest element should go) with the element at maxIdx (the actual largest element).
   * The swapping process involves temporarily storing the value of arr[maxIdx] in a temp variable, replacing arr[maxIdx] with arr[i], and then assigning the value of temp to arr[i].

**Example Walkthrough:**

Let's walk through the code with an example array: {5, 1, 4, 2, 8}.

* **First Iteration** (i = 0):
  + maxIdx starts at 0 (assuming 5 is the largest element).
  + Inner loop compares 5 with the rest of the elements: 1, 4, 2, 8.
  + 8 is found to be the largest, so maxIdx is updated to 4.
  + Swap arr[0] (which is 5) with arr[4] (which is 8).
  + Array becomes {8, 1, 4, 2, 5}.
* **Second Iteration** (i = 1):
  + maxIdx starts at 1 (assuming 1 is the largest).
  + Inner loop compares 1 with 4, 2, 5.
  + 5 is found to be the largest, so maxIdx is updated to 4.
  + Swap arr[1] (which is 1) with arr[4] (which is 5).
  + Array becomes {8, 5, 4, 2, 1}.
* **Third Iteration** (i = 2):
  + maxIdx starts at 2 (assuming 4 is the largest).
  + Inner loop compares 4 with 2, 1.
  + No element larger than 4 is found, so no swap is needed.
  + Array remains {8, 5, 4, 2, 1}.
* **Fourth Iteration** (i = 3):
  + maxIdx starts at 3 (assuming 2 is the largest).
  + Inner loop compares 2 with 1.
  + No element larger than 2 is found, so no swap is needed.
  + Array remains {8, 5, 4, 2, 1}.

After all iterations, the array is sorted in descending order: {8, 5, 4, 2, 1}.

**Summary of the Key Concepts:**

* **Selection Sort** is an algorithm that repeatedly selects the maximum (or minimum) element from the unsorted portion of the array and swaps it with the first unsorted element.
* The inner loop finds the largest element in the unsorted portion of the array.
* After the inner loop completes, the largest element is swapped with the first unsorted element to progressively sort the array.