Applying the Algorithm

Let's look at a practical example of this algorithm. The `insert()` function at the top of the program has not been completed, so we're going to go ahead and do that.

1. **Find the position** where the new element **should be** inserted:

   ```
   size_t pos = 0;
   while (pos < size && a[pos] < value)
   {
       ++pos;
   }
   ```

   The variable (`pos`) is initialized to `0`. After the loop, it will contain the location where the new element should be inserted.

   If there are no elements larger than the number you are inserting, `pos` will contain the same value as `size`, and the number will then be added at the end of the array, which is what you want.

2. **Move the existing elements**. Before you can store value in the array, you must move the existing elements out of the way (to the right), to "open up a hole" for the new value.

   ```
   for (size_t i = size; i > pos; --i)
   {
       a[i] = a[i - 1];
   }
   ```

   You must start **at the end of the array**, traversing to the left, until you reach the location where you intend to insert the new element, so you don't overwrite data.

3. **Insert the new element**. After moving, copy the new number into position, and **update the `size`**.

   ```
   a[pos] = value;
   ++size;
   ```

This course content is offered under a CC Attribution Non-Commercial license. Content in this course can be considered under this license unless otherwise noted.