**Inserting Along the Traversal**

Another standard array algorithm is [**the insertion algorithm**](javascript:void(0);) **(an insertion is an algorithm in which elements are added to the array or ArrayList).** Similar to the others, the insertion algorithm involves a traversal with modifications.

This is like going through your three-ring binder to a particular spot and then opening the binder and inserting new pages. You are not replacing anything, but rather adding to the binder. This is the insertion algorithm.

For arrays, an insertion involves adding a new element to a particular part of the array. Unlike a replacement, you are not changing an existing element; you are adding a new one. When inserting items into an array, you first need to decide where you want to perform the insertion. This could be done by specifying a location, such as a particular index. Or you could search for an appropriate location based upon some criteria, such as adding relaxation kits after the coffee kits in an inventory list.

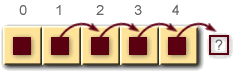
Once the location is decided upon, then all that remains is moving existing elements in the array make room for the new item. Since arrays do not automatically resize and if all positions in the array are occupied, that means the last element of the array gets lost. This is not normally desirable, so in practice you would use an array that is larger than you actually need. This would let you insert items without losing anything (until the array is full).

ArrayLists are dynamic and grow in size when adding elements, so they do not lose items like an array insertion might. Therefore, depending on the situation, many programmers prefer using ArrayLists in Java.

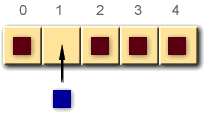
### Part 1

This lesson will highlight one of the big differences between managing arrays and ArrayLists. The eIMACS labs will allow you to explore the insertion algorithm in more detail.

Rather than replacing an element, another way to change the contents of an array is to *insert* an element. [Just now](https://www.eimacs.com/eimacs/mainpage?epid=E2365697720&cid=162149), we saw that replacing an array element results in the destruction of the element being replaced. Insertion, on the other hand, does not necessarily entail the destruction of an existing object. In fact, the first stage of performing an element insertion is to create a space for the new element by moving existing elements.



Specifically, starting with the element at the insertion index, each element must be moved one space to the right (see the figure on the right, which illustrates the preparation for inserting an element at index 1). Once the space has been created, the new item can be inserted into it:



There are two "gotchas":

* If elements are moved to the right, what happens to the last element of the array? It has nowhere to go.
* The elements must be moved to the right one at a time. In which order should they be moved?

Regarding the first of these, there are two possible strategies: First, we could allow the last array element to "drop off the edge of the cliff". That is to say, we could simply discard it. Second, we could preserve the element in some way, perhaps by extending the array (which is the strategy employed by ArrayLists). For our purposes here, we adopt the first strategy.

Concerning the second question, consider the following code fragment in which a new Item is inserted at index 1 as follows: First, the element currently at index 1 is copied to index 2. Then the element at index 2 is copied to index 3, and so on, and the element currently at the end of the array (at index 4) is discarded. Finally, the new element is placed into the array at index 1. Is this a correct algorithm for inserting the new item? Run the code to find out.

  public static void main( String[] args )   
  {   
    Item[] array = Item.makeItemArray( 5 );   
  
    System.out.println( "Before: " );   
    Item.displayArray( array );   
  
    // make a new Item   
    Item newItem = new Item( 99 );   
  
    // make space for the item   
    array[ 2 ] = array[ 1 ];   
    array[ 3 ] = array[ 2 ];   
    array[ 4 ] = array[ 3 ];   
  
    // insert new item   
    array[ 1 ] = newItem;   
  
    System.out.println( "\nAfter: " );   
    Item.displayArray( array );   
  }

No, it is not a correct algorithm. When we copy array[ 1 ] into array[ 2 ], we overwrite the Item currently stored in array[ 2 ], losing it forever. The mistake is then compounded by the subsequent copies, which in fact cause the original object at index 1 to overwrite *all* the elements that come after it.

To avoid this problem, we copy the elements in order from the right to left, as follows: First, we copy array[ 3 ] into array[ 4 ], thereby overwriting array[ 4 ] and effectively discarding it. Next, we copy array[ 2 ] into array[ 3 ]. Although this action overwrites array[ 3 ], that element is not lost because we just got through copying it into array[ 4 ]. Then array[ 1 ] is copied into array[ 2 ], and finally we can place the new item into array[ 1 ]. Run the following code to verify that the insertion is made correctly:

  public static void main( String[] args )   
  {   
    Item[] array = Item.makeItemArray( 5 );   
  
    System.out.println( "Before: " );   
    Item.displayArray( array );   
  
    // make a new Item   
    Item newItem = new Item( 99 );   
  
    // make space for the item   
    array[ 4 ] = array[ 3 ];   
    array[ 3 ] = array[ 2 ];   
    array[ 2 ] = array[ 1 ];   
  
    // insert new item   
    array[ 1 ] = newItem;   
  
    System.out.println( "\nAfter: " );   
    Item.displayArray( array );   
  }

### Part 2

The fact that arrays are static with regard to their length and ArrayLists are dynamic in size can make a big difference in the amount of coding required for some programming needs. Use the demonstration programs to solidify your understanding of the insertion algorithm with the two data structures.

* Open the 07.06 Virtual Lecture Notes.
* Create a new project called 07.06 Insertions in the Mod07 Lessons folder.
* Download the following Java files to the newly-created folder:
  + InventoryItem.java
  + TestInventory5.java
  + TestInventory6.java
* Carefully read the discussion and carry out the instructions.