Destination sort code and explanation of operation

By Ray Houghton 10/5/25

```
void loop()
{
ptrInitialOrder = &initialOrder[0];
ptrIntermediateOrder = &intermediateOrder[0];
ptrDestination = &destination[0];
```

Assigns pointers to arrays

```
InitArray(ptrInitialOrder, ptrIntermediateOrder, elementNumber);
```

Writes elements in initialOrder to intermediateOrder

```
while(Array_cmp(ptrInitialOrder, ptrDestination, elementNumber) == false)
```

While loop that runs until the sorting is complete

```
for(element = elementNumber; element>=1; element --)
```

For loop that counts down from elementNumber to 1.
element refers to the position of the number being worked on in the destination array.

```
  {
  a = 0;
```

“a” is a value that is set to one when an element is in the correct position.

```
m = element;
```

Assigns the starting point to look for the current element in "Destination" in the "initialOrder" array

```
 while(a != 1)
```

while loop to find current "destination[]" element in the "initialOrder" array

```
 {

 if(initialOrder[m]!=destination[element])
```

find equivalent element

```
   {
    m--;
   }
   else
    {
     a = 1;   exit loop with "m" at the correct value
    }
 }
 int b = (m);
```

assigns equivalent element pos'n to variable "b"

```
for(int j = b; j <= (element-1); j++)
```

 loop to move selected "element" into correct position

```
{

SwapElement(ptrInitialOrder, ptrIntermediateOrder, j);
```

move misplaced element towards correct position

```
PrintArray(ptrInitialOrder, elementNumber);
```

print order of "initialOrder" after swap

```
}
}
}
}
```

Graphical representation

m = 7

initOrder	1	2	3	4	5	6	7	8

elementNumber = 7

Destination	1	3	5	7	2	4	6	8

m is in the correct position so no action is taken

m = 6

initOrder	1	2	3	4	5	6	7	8

elementNumber = 6

Destination	1	3	5	7	2	4	6	8

m is not adjacent to the correct element so m is decremented until it is positioned correctly.

						m = 5		
initOrder	1	2	3	4	5	6	7	8
						b = 5		
							elementNumber = 6	
Destination	1	3	5	7	2	4	6	8

m is now adjacent to the element indicated by elementNumber so the element at m is swapped so that it is now in the correct position. The difference between b and element indicates the number of swaps required to get m in the correct position.

				m = 3				
initOrder	1	2	3	4	5	7	6	8
				b = 3				
						elementNumber = 5		
Destination	1	3	5	7	2	4	6	8

The last two elements are now in the correct position in the initOrder array, so elementNumber is moved to the next element in the Destination array. The variable m has decremented to find the correct element. Now b is two less than element so there are two swaps to get the element at m in the correct position. This continues until all of the elements are in the correct position.