Working With Arrays

PART II

In Part I of Working with Arrays the algorithms for outputting an array to the console or returning it as a String as well as adding and removing an element from an array we presented. Part II will present the algorithms and pseudo code for working with arrays to accomplish the remaining tasks:

- Searching for a value in the array
- Searching an array for a maximum or minimum value
- Moving elements within the array, the most common operation involves "swapping" two elements
- Merging two arrays

In each case, the goal is to write a method for the algorithm.

Searching an Array

Before considering the algorithm for searching an array, consider the result of such a search. Using the example `names` array, if the value to be searched for was "`Dennis`" then the result of such a search could have several different forms.

```
String[] names={"Adam","Becky","Charlie","Dennis","Ethan","Fred","Gary","Heath"};
```

The value searched for, "`Dennis`," could be returned, but that wouldn't be terribly useful since the value was already known, after all it was the search value. All we would know in that case is that the value existed. A simple `boolean` result could be returned, the value was either in the array or it wasn't.

Instead, a more useful piece of information would be the index of the array element where the value was found! If the value isn't in the array, then a value that would not normally be returned can be returned instead. This is how the `String` method `.indexOf()` works. When the value can't be found in the string, -1 is returned because it isn't a possible index.

Below is a summary of the information about the method to be written:

Method Attribute	Description	JAVA
Return Type	Returning an array index	`int`
Method Name	This method will search an array for a value	`searchLinear`
1st Parameter	The value to search for, or find	`String s`
2nd Parameter	The array to search, using a generic name	`String[] arr`
Return value(s)	If the string to be found is in array, return the index	`0 to array.length-1`
	If the string to be found isn't in the array, return -1	`-1`

With this information the method signiature can be written:

```
public static int searchLinear(String s, String[] arr)
```

Recall that the proper way to compare two `String` values for equality is the `.equals()` method instead of the == operator. However, this will introduce possible complications. Recall as well that a `String` may have the special value **`null`**, in which case trying to use the `.equals()` method will generate a "`NullPointerException`" so it is important to first make sure that each `String` is not **`null`** before trying to use the `.equals()` method.

Such a test can be written as a simple "**`if`**" statement:

```
if (find==null)
```

If the value to find, stored in `s` variable is **`null`**, then a separate kind of equality test can be performed to see if the array value is also **`null`**!

Comparing `s` to each value in the array inside a "for loop" will complete the LINEAR SEARCH algorithm.

Here is the complete pseudo-code:

1. Get the value to find and the array as parameters.
2. If the array is null return -1.
3. Loop over each value in the array.
4. If the array element is the same as the value to find, return the current array index.
5. If the end of the loop is reached, then the value isn't in the array, so return -1.

The method signature is:

```
int searchLinear(String s, String[] arr)
```

Assume an array: `letters={"A","B","C","D"};`

And a call to the method `index=searchLinear("C", letters)`:

Use Chart D to follow this process step by step for the sample method call:

Chart D

`String[] arr`

index	Value
0	"A"
1	"B"
2	"C"
3	"D"

Step 1: Get the array value

```
arr={"A","B","C","D"};
s="C";
```

Step 2: `arr` is not null so do nothing.

Step 3: Set up the iterator loop.

Step 4: Return the current iterator index if `s` is equal to the array element at the iterator index. In this example, when the iterator reaches 2, `arr[iterator]` will equal `s` so the value 2 will be returned.

Step 5: Return the value negative one if the end of the loop is reached. This does not occur in this example.

Now follow the same algorithm for three examples:

Step #	Searching for a value that isn't in the array	Searching for a null value	Searching for a null array
1	`arr={"A","B","C","D"};` `s="E";`	`arr={"A",null,"C","D"};` `s=null;`	`arr=null;` `s="D";`
2	`arr` is not null, do nothing	`arr` is not null, do nothing	`arr` is `null`, return -1
3	Set up the iterator loop	Set up the iterator loop	Not Reached
4	`s` never equals an array value	`s` is null, so use the == operator to compare it to each element of the array, and return the index of 1 because that array element is `null`	Not Reached
5	Return -1	Not Reached	Not Reached

Checking for equality when the variable might be null is tricky because you have to make sure the variable isn't null as your first step.

The code to check for equality when a variable might be null looks like this (using the same variable names from the example):

```
if(s==null && arr[iterator]==null) {
      return iterator;
} else if (s.equals(arr[iterator])) {
      return iterator;
}
```

When null values are not possible the code is much simpler:

```
if (s==arr[iterator]) return iterator;
```

Searching an Array for a Minimum or Maximum Value

It will be easier to consider searching for a min or max value considering numbers, but the process can be applied to String arrays as well.

`int[] numbers`

index	Value
0	12
1	15
2	11
3	9
4	10

Now examine a new data set for this example, an `int` array called `numbers`:
`int[] numbers={12,15,11,9,10};`

A quick glance at the table reveals that the smallest value, "9," is stored at index 3.

However, if this array contained a list of a million numbers, it would take much more than a glance to establish what the smallest value was in the list.

In such a case, an algorithm would make more sense!

Use the Chart E to follow this process step by step:

Chart E: Finding a Minimum

index	Value	Low Index	Low Value	Compare	Next Low Index	Next Low Value
0	12	0	12	Assume	0	12
1	15	0	12	12<15	0	12
2	11	0	12	11<12	2	11
3	9	2	11	9<11	3	9
4	10	3	9	10<9	3	9

Begin by assuming the first value in the list, the one at index 0, is the smallest value.

Now compare the value stored at the current low index of 0 with the one stored at index 1. Since the value stored at index 0 is less than the value at index 1, index 0 remains the low index.

Next compare the value stored at the current low index of 0 with the one stored at index 2. Since the value stored at index 0 is greater than the value at index 2, index 2 becomes the new low index.

Then compare the value stored at the current low index of 2 with the one stored at index 3. Since the value stored at index 2 is greater than the value at index 3, index 3 becomes the new low index.

Finally, compare the value stored at the current low index of 3 with the one stored at index 4. Since the value stored at index 3 is less than the value at index 4, index 3 remains the new low index.

Notice that each of the comparison steps followed the same procedure, only the indexes to be compared each time changed. The first index, the current low index could easily be considered a variable and the second the second index is simply a loop iterator that starts at 1 and ends at end of the array!

This is how a LINEAR SEARCH algorithm works. Why record the index instead of the actual number?

The answer once again is a question of getting the maximum information possible. If you have the index you can get the value, but if you only have the value it is impossible to know the index too.

Now that the algorithm has been analyzed consider the method header, also known as a METHOD SIGNATURE. Just like searching for a matching value, we can construct a table to help analyze the method:

Below is a summary of the information about the method to be written:

Method Attribute	Description	JAVA
Return Type	Returning an array index	`int`
Method Name	This method will return the index of the lowest value in the array	`indexMin`
Parameter	The array to search, using a generic name	`int[] arr`
Return value	Index of the element with the lowest value	`0 to arr.length-1`

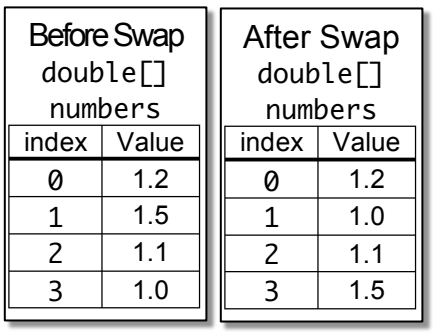
Notice that there are a few differences from the `searchLinear` method. The return type will still be an `int` because an array index value will be returned, but there is only one parameter, the `int` array.

Some index value will always be returned there is always a smallest value!

The method signature is: `int indexMin(int[] arr)`

Swapping Values in an Array

Consider the data set of the `double` array called `numbers`:
`double[] numbers={1.2,1.5,1.1,1.0};`



To swap two elements of the array a simple algorithm can be followed:

1. Store the value of the first element in a temporary variable.
2. Assign the value of the second element to the first element.
3. Assign the value of the temporary variable to the second element.

A method to accomplish this with any two element indexes would require just a bit more work because it should work with any array and it should check to make sure it is legal to swap the desired elements so an "`ArrayIndexOutOfBoundsException`" exception is not thrown.

Below is a summary of the information about the method to be written:

Method Attribute	Description	JAVA
Return Type	A `boolean` value indicating that the swap was performed successfully.	`boolean`
Method Name	This method will swap the values of two elements.	`swap`
Parameter(s)	The array in which the swap will take place.	`int[] array`
	Index of the first element	`int indexA`
	Index of the second element	`int indexB`
Return value	`true` only if both `indexA` and `indexB` fall in the range `0 to array.length-1`	`true` or `false`

The method signature is:

```
boolean swap(int indexA, int indexB, int[] arr)
```

Creating an Array by combining two Arrays

The algorithm for combing two arrays is actually quite simple:

1. Create a new array with a size equal the combined length of the two arrays.
2. Use a loop to assign elements from the first array to the new array using the same indexes until the end of the first array.
3. Use a second loop to assign elements from the second array to the new array offsetting the index in the new array by the length of the first array.
4. Return the new array.

Below is a summary of the information about the method to be written:

Method Attribute	Description	JAVA
Return Type	The new `String` array made up of all the combined elements from both parameter arrays.	`String[]`
Method Name	This method will combine the values of two arrays into a new array.	`add`
Parameter(s)	The first array.	`String[] arr1`
	The second array.	`String[] arr2`
Return value	The combined array.	`String[]`

The method signature is:

```
String[] add(String[] arr1, String[] arr2)
```