Array Algorithms

by Sophia

WHAT'S COVERED

In this lesson, you will learn about how to process data using array algorithms in Java. This includes finding high and low values, search features, and copying code in arrays. Specifically, this lesson covers:

1. Finding the Highest and Lowest Values in an Array

In the previous lesson, you learned about the `Arrays.sort()` method. This method sorts the data in an array in numeric or alphabetic order. This functionality can be used to find the highest and lowest values in an array without looping through the array. While loops are used in this tutorial, they will be covered later in the course.

TRY IT

Directions: Start with an array of integers declared like this:

EXAMPLE

```
int[] scores = {77, 89, 100, 68, 95};
```

Using `Arrays.sort()` will then render the array in this order:

EXAMPLE

```
{68, 77, 89, 95, 100}
```

After the values have been sorted, you will see that the first value in the array is the lowest value. Keep in mind that the first element in the array has the index 0.

It can be seen as the lowest value and can be accessed like this:

EXAMPLE

```
int lowest = scores[0];
```

The last value in the array is the highest value in the array, after the data has been sorted. Calculating the last index is a bit more complex than finding the first index.

The **length (or size) of the array** can be determined when using the `length` properly, as seen below:

EXAMPLE

```
int arrayLength = scores.length;
```

Since the first index is 0, the highest index is always one less than the size of the array. It looks like this:

EXAMPLE

```
int highestIndex = scores.length - 1;
```

The highest value (in the last element in the array) can be accessed using highestIndex, which displays like this:

EXAMPLE

```
int highest = scores[highestIndex];
```

As you have learned, an expression can be used in the square brackets for the array index.

Therefore, the highest value in the array could also be accessed like this:

EXAMPLE

```
int highest = scores[scores.length - 1]
```

REFLECT

It is hard to overemphasize the point that the highest index in an array is always one less than the length of the array. Off-by-one errors are a common bug in code, so the programmer needs to be alert.

TRY IT

Directions: Try typing in the complete program in the IDE to see the results:

```
import java.util.Arrays;

class ArraySort {
  public static void main(String[] args) {
    // Declare and initialize array with values
    int[] scores = {77, 89, 100, 68, 95};
    //System.out.println("Incorrect way to print: " + scores);
    System.out.println("Scores: " + Arrays.toString(scores));
    Arrays.sort(scores);
    System.out.println("Sorted Scores: " + Arrays.toString(scores));
    // Use the length property (or attribute) to get size of array
    int size = scores.length;
    System.out.println("Array Size: " + size);
    // Element 0 contains the lowest value
    System.out.println("Lowest Score: " + scores[0]);
    // Last index is 1 less than the size
    int lastIndex = size - 1;
    System.out.println("Highest Score: " + scores[lastIndex]);
  }
}
```

REFLECT

The output from running this program should look like this:

```
Scores: [77, 89, 100, 68, 95]
Sorted Scores: [68, 77, 89, 95, 100]
Array Size: 5
Lowest Score: 68
Highest Score: 100
```

TERM TO KNOW

Length (or Size) of an Array
The length or size of an array is the number of elements in the array. The `.length` property is used to get the length or size of an array.

2. Searching an Array

You have explored how to use `Arrays.sort()` to find the highest and lowest values in an array. The `Arrays.sort()` method also plays a role in searching an array. The `Arrays` utility class includes a method called **Arrays.binarySearch()**. This method returns the index of the sought value, or -1 if the value is not found in the array.

There is an important stipulation, though. The array being searched must be sorted first. If it is not sorted, the behavior and result of the search will be undefined and unreliable.

KEY CONCEPT

In a small set of values like the example below, the binary search may seem to work, but it's important to remember that undefined behavior doesn't mean it won't work, just that it may fail—and perhaps at the worst possible moment.

After using `Arrays.sort()` to put the array elements in order, the `Arrays.binarySearch()` method can be called like this (to search for the value 100):

EXAMPLE

```
Arrays.sort(scores);
int location = Arrays.binarySearch(scores, 100);
```

HINT

The data type of the value being searched for must be the same as the data type of the array.

TRY IT

Directions: Type this code into the IDE to see how the process works:

```
import java.util.Arrays;

class BinarySearch {
  public static void main(String[] args) {
    int[] scores = {77, 89, 100, 68, 95};
    int searchValue = 100;
    Arrays.sort(scores);
    System.out.println("Sorted Array: " + Arrays.toString(scores));
    int location = Arrays.binarySearch(scores, searchValue);
    if(location > -1) {
      System.out.println(searchValue + " found.");
    }
    else {
      System.out.println(searchValue + " not found.");
    }
  }
}
```

The program's run should look like this:

```
Sorted Array: [68, 77, 89, 95, 100]
100 found.
```

REFLECT

When using a binary search, it may seem that it works correctly when the array hasn't been sorted first, but it's important to remember that this is not a guarantee that it will work in a real-world production environment or with a larger array. Sometimes things may seem to be okay when tested in a small, simple program, but best practices are best practices, and all of the official documentation insists on sorting the array before using Arrays.binarySearch().

TERM TO KNOW

Arrays.binarySearch()
This method is used to search for the occurrence of a specified value in a sorted array.

3. Copying All or Part of an Array

The **Arrays.copyOf()** method allows users to copy an array from a source array and paste it to a destination array. The `Arrays.copyOf()` method uses two parameters.

These include the source array and the length or size of the copied array, as seen below:

EXAMPLE

```
int[] scores = {77, 89, 100, 68, 95}; int[] scoresCopy = Arrays.copyOf(scores,
5);
```

Both arrays now contain the values {77, 89, 100, 68, 95}. If the value provided as the parameter for the size is larger than the original array, the new array will have the extra elements filled with 0, an empty `char`, or an empty `String`. Which one depends on the data type. This has the effect of allowing us to increase the size of an array, although it is not very efficient to do it this way.

Here is a sample program that increases the size of the scores array by copying a larger array onto the original:

```
import java.util.Arrays;

class ArrayCopy {
  public static void main(String[] args) {
    // Original array of 5 integers
    int[] scores = {77, 89, 100, 68, 95};
    System.out.println("Original: " + Arrays.toString(scores));
    // Copy larger array of 10 integers back to scores
    scores = Arrays.copyOf(scores, 10);
    System.out.println("Enlarged: " + Arrays.toString(scores));
  }
}
```

When run, this program produces the following output:

```
Original: [77, 89, 100, 68, 95]
Enlarged: [77, 89, 100, 68, 95, 0, 0, 0, 0, 0]
```

In addition to the `Arrays.copyOf()` method, there is also an **Arrays.copyOfRange()**. `Arrays.copyOfRange()` allows users to copy part of an array to a destination. This method is called with values passed for the source array, the starting index for the copy, and the ending index for the copy.

This method automatically handles the offset by 1:

EXAMPLE

```
int[] scores = {77, 89, 100, 68, 95};
int[] lastThree = Arrays.copyOfRange(scores, scores.length - 3,
scores.length);
```

The array lastThree will contain the values {100, 68, 95}.

If used with `Arrays.sort()`, `Arrays.copyOfRange()` allows for gathering the top or bottom values, as this example shows:

```
import java.util.Arrays;

class ArrayCopyOfRange {
  public static void main(String[] args) {
    // Original array of 5 integers
    int[] scores = {77, 89, 100, 68, 95};
    System.out.println("Original: " + Arrays.toString(scores));
    Arrays.sort(scores);
    System.out.println("Sorted: " + Arrays.toString(scores));
    // start is 3rd from end, runs until the end - 1
    int[] topThree = Arrays.copyOfRange(scores, scores.length - 3, scores.length);
    System.out.println("Top 3: " + Arrays.toString(topThree));
  }
}
```

The results from the program show the top 3 scores:

```
Original: [77, 89, 100, 68, 95]
Sorted: [68, 77, 89, 95, 100]
Top 3: [89, 95, 100]
```

TERMS TO KNOW

Arrays.copyOf()
The `Arrays.copyOf()` method allows users to copy an array from a source array and paste it to a destination array. The `Arrays.copyOf()` method uses two parameters. These include the source array and the length or size of the copied array.

Arrays.copyOfRange()
This method allows copying part of an array to a destination. This method is called with values passed for the source array, the starting index for the copy, and the ending index for the copy.

SUMMARY

In this lesson, you have learned how to use a couple useful array processes. You have determined how to find the **highest and lowest values** in an array by using Arrays.sort(). You have also seen how Arrays.binarySearch() can be used to **search for a value in an array**, provided that the array has been sorted first. You have learned how to **copy all or part of an array**. Finally, you learned how methods provided by the Arrays class can help enlarge an array to assist in finding the top set of values.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

Arrays.binarySearch()
This method is used to search for the occurrence of a specified value in a sorted array.

Arrays.copyOf()

The Arrays.copyOf() method allows users to copy an array from a source array and paste it to a destination array. The Arrays.copyOf() method uses two parameters. These include the source array and the length or size of the copied array.

Arrays.copyOfRange()

This method allows copying part of an array to a destination. This method is called with values passed for the source array, the starting index for the copy, and the ending index for the copy.

Length (or Size) of an Array

The length or size of an array is the number of elements in the array. The .length property is used to get the length or size of an array.