6-Arrays

Author: Vincent Lau

Learning Objectives

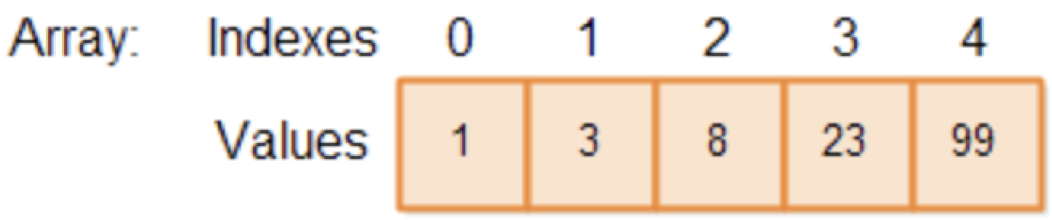
> What is Array?
>
> How are arrays being stored in memory?
>
> Why do we need Array?
>
> Able to create and access an array

Arrays

What's an Array?

- An **array** is an object **containing a fixed number of values of the same type**

- The elements of an array are indexed, which means we can access them with index. **The index of an array starts at zero**.
- **Arrays can be primitive type or object type**, as long as all the items are of the same type.



Creating an Array

```java
// Declaration of array
int[] arr;
// When initializing an Array, the size must be specified
// When creating an Array this way, we initialize each element to its default
value, 0
arr = new int[10]; // index 0 - 9

// Another way of initializing an Array
int[] arr2 = new int[] {1, 2, 6, 4, 5};

// Initializing a 2-dimensional Array
int[][] arr3 = new int[3][2];
String[][] arr4 = new String[][]{{"John", "Lau"}, {"Sally", "Wong"}, {"Eric",
"Cheung"}};
```

Get the length of array

```java
int[] arr = new int[] {1, 2, 200, 4, 5, 100, 7};
System.out.println(arr.length); // prints 7
```

Accessing Elements

```java
int[] arr = new int[] {1, 200, 3, 4, 5, 100, 7};
arr[2] = 99; // revise the value of 3th position of the array
System.out.println(arr[2]); // prints 99

int firstElement = arr[0];
int fourthElement = arr[3];

System.out.println(firstElement); // Output: 1
System.out.println(fourthElement); // Output: 4
```

Iterating Over Array Elements

```java
int[] numbers = { 1, 2, 3, 4, 5 };

// Iterating over array elements using a for loop
for (int i = 0; i < numbers.length; i++) {
    System.out.println(numbers[i]);
}
// Iterating over an array reversely
for (int i = numbers.length - 1; i >= 0 ; i--) {
    System.out.println(numbers[i]);
}

// Iterating over 2D-array elements using outer loop and inner loop
int[][] arr2d = new int[][] {{1,2,3}, {7,8,9}, {10,1,2}};
for (int i = 0; i < arr2d.length; i++) {
    for (int j = 0; j < arr2d[i].length; j++) {
        System.out.println("row=" + i + ", col=" + j + ", val=" + arr2d[i][j]);
    }
}
```

This example demonstrates how to iterate over array elements using a `for` loop. The loop variable `i` starts from 0 and goes up to `numbers.length - 1`. Each element of the `numbers` array is accessed using the loop variable `i`.

Adding an element into Array

```java
String[] colorsArray = new String[5];

// initial array values
colorsArray[0] = "Red";
colorsArray[1] = "Green";
colorsArray[2] = "Blue";
System.out.println("Original Array:" + Arrays.toString(colorsArray));

int numberOfItems = 3;
// try to add new value at the next position of the array
```

```java
colorsArray[numberOfItems] = "Yellow"; // colorsArray[3] = "Yellow"
System.out.println("Array after adding one element:" +
Arrays.toString(colorsArray));
```

Why do we need Array

Grouping Related Data

Arrays allow us to **group related data items of the same type under a single variable name**. For example, if we want to store a list of integers or strings, an array provides a convenient way to organize and access these elements.

Sequential Access

Arrays provide sequential access to elements based on their index. Each element in the array is assigned a unique index, starting from 0. This enables **efficient access and manipulation of individual elements based on their position within the array**.

Storage Efficiency

Arrays provide **efficient memory allocation and storage for elements of the same type**. Elements in an array are stored in contiguous memory locations, allowing for easy access and efficient memory management.

Search for an element

Searching is the most common algorithm that a developer uses every day.

The easiest way is **search it one by one** until you get the target element. We call it **linear search**.

In linear search, each element of the array is sequentially compared with the target element until a match is found or the entire array is traversed. When a match is found, we can do something and break the loop.

```java
// Searching for an element in an Array
arr = new int[]{1, 9, 8, 4, 6, 5, 2, 3, 7};
int target = 8;
for (int i = 0; i < arr.length; i++) {
    if (arr[i] == target) {
        System.out.println("Found " + target + " at index " + i);
        break;
    }
} // prints "Found 8 at index 2"

```

String to Array

Method `toCharArray()` is a kind of tool (just treat it as tool, we do not know method yet), which is able to **convert String Data Type to char Array Data Type, by creating a new array and copying the characters of the string into it.**

The `toCharArray()` method is a built-in method in Java that belongs to the `String` class. Each character in the resulting array corresponds to a character in the original string, preserving the order.

```java
public class StringSwapExample {
    public static void main(String[] args) {
        String str = "Hello";

        char[] chars = str.toCharArray();

        for (int i = 0; i < chars.length; i++) {
            System.out.println("i=" + i + ", char[" + i + "]=" + chars[i]);
        }
        // what is the output?
    }
}
```

In the example above, we have a `String` object called `text` with the value "Hello". We use the `toCharArray()` method to convert the string into a `char` array. Then, we iterate over the elements of the `char` array using a simple for loop and print each character to the console.

Basic Algorithms

Find Max

```java
int[] arr = new int[] {1, 2, 6, 4, 5};
int max = Integer.MIN_VALUE;

for (int i = 0; i < arr.length; ++i) {
    if (arr[i] > max) {
        max = arr[i];
    }
}
System.out.println(max); // print 6
```

Find Min

```
int[] arr = new int[] {1, 2, 6, 4, 5};
int min = Integer.MAX_VALUE;

for (int i = 0; i < arr.length; ++i) {
    if (arr[i] < min) {
        min = arr[i];
    }
}
System.out.println(min); // print 1
```

Swap Elements

Swapping two elements in an array is an important programming skill, which we always use it to solve technical questions, especially sorting algorithms. Let's start.

Example 1: Swap elements at index 1 and index 2

```
public class SwapElement {
    public static void main(String[] args) {
        int[] arr = {5, 10, 15, 20};

        // Swap elements at index 1 and index 2
        int temp = arr[1];
        arr[1] = arr[2];
        arr[2] = temp;

        // Print the array after swapping
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        // 5 15 10 20
    }
}
```

Example 2: Swap Characters in a String

```
public class StringSwapExample {
    public static void main(String[] args) {
```

```java
        String text = "Hello";

        // Swap characters at index 0 and index 4
        char[] chars = text.toCharArray();
        char temp = chars[0];
        chars[0] = chars[4];
        chars[4] = temp;

        // Print the string after swapping
        String swappedText = new String(chars);
        System.out.println(swappedText);
    }
}

```

Learning tips for Swapping

Understand Array Indexing

Familiarize yourself with how arrays are indexed in Java. Array elements are accessed using an index, starting from 0 for the first element.

Identify Swapping Elements

Determine the elements you want to swap within the array. You'll need to know the indices of the elements you want to exchange.

Use a Temporary Variable

To perform the swap, create a temporary variable to store one of the elements. This prevents data loss during the swapping process.

Swap Elements

Assign the value of one element to the other, and then assign the value of the temporary variable to the remaining element. This completes the swap.

Move the max number to tail

After learning the skill of swapping the elements, you should be able to figure out the way of moving the max number to the tail of array. Make sure you can understand and program this algorithm before moving to the next section.

Swapping can do lots of operations within the array. Lots of famous & useful java libraries & methods are achieved by simple swapping in arrays.

Example 1

```java
// Move the max number to the tail
int[] nums = new int[] {8, 3, -10, 30, 100, -19};

// Step1: Find the index of the max number
int maxIndex = 0;
for (int i = 0; i < nums.length; i++) {
    if (nums[i] > nums[maxIndex]) {
        maxIndex = i;
    }
 }
// Step2: Swap the max number to tail
int temp = nums[maxIndex];
nums[maxIndex] = nums[nums.length - 1];
nums[nums.length - 1] = temp;

// After the loop, do you know the sequence of all elements in the array now?
```

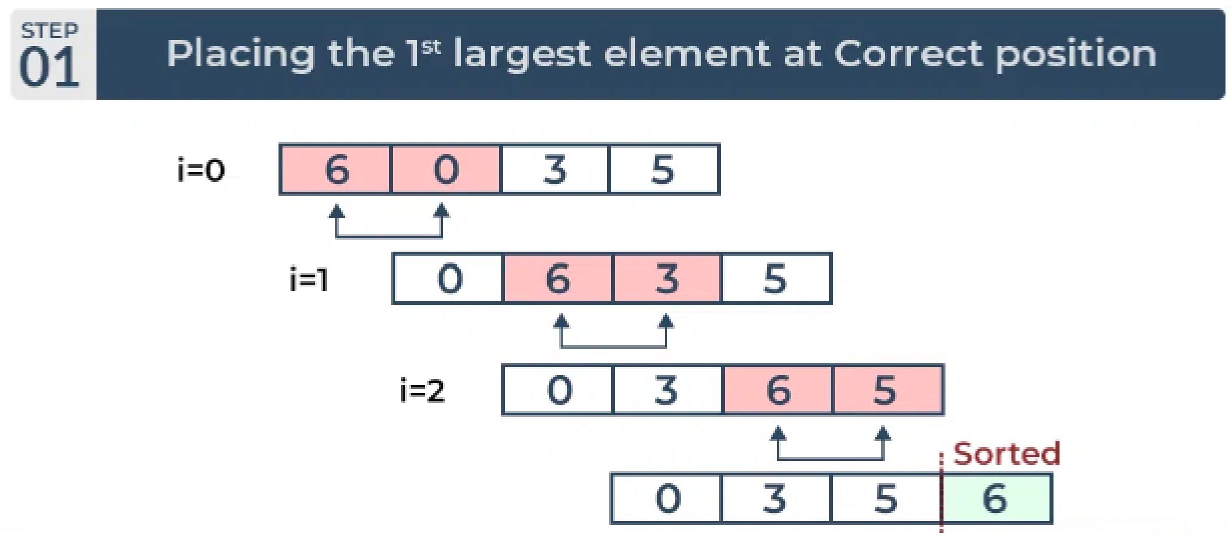
Exercise

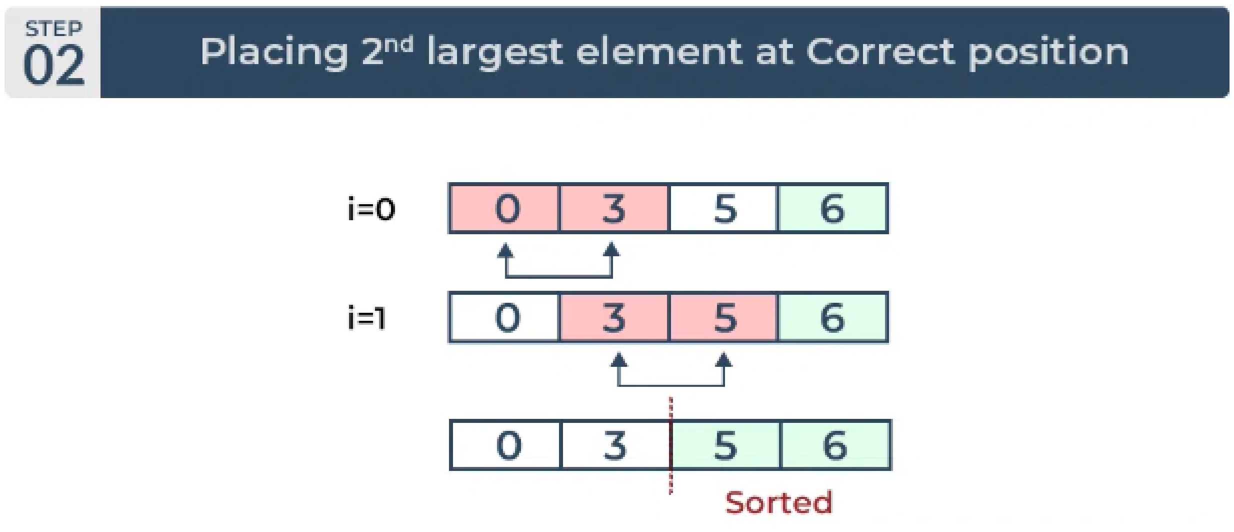
```java
// Move the min. number to the head of array
int[] nums2 = new int[] {1, 101, -1000, -3, 4};
// Your turn. Please try.
```

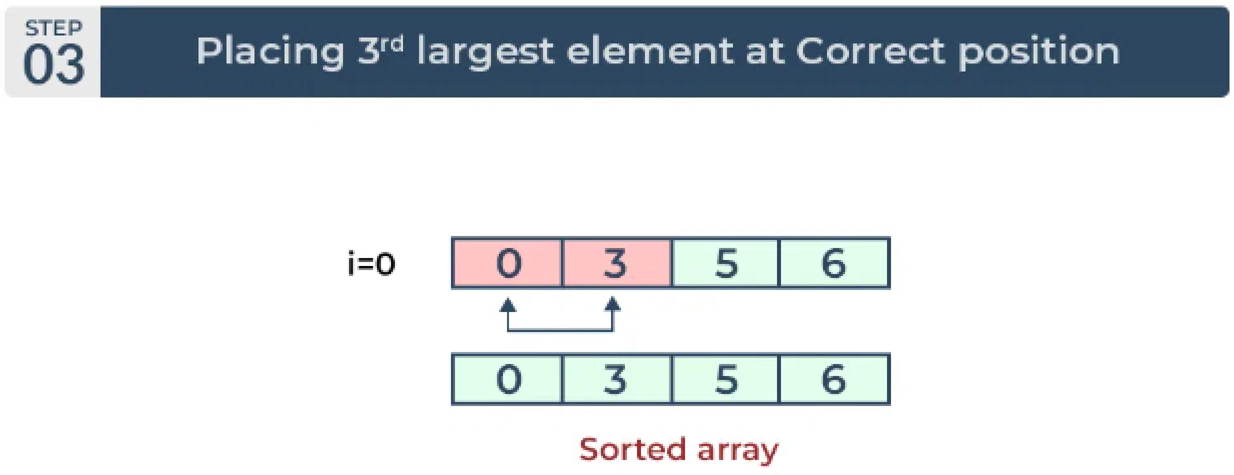
Sorting Algorithms

Sorting Array in Ascending Order

Bubble Sort



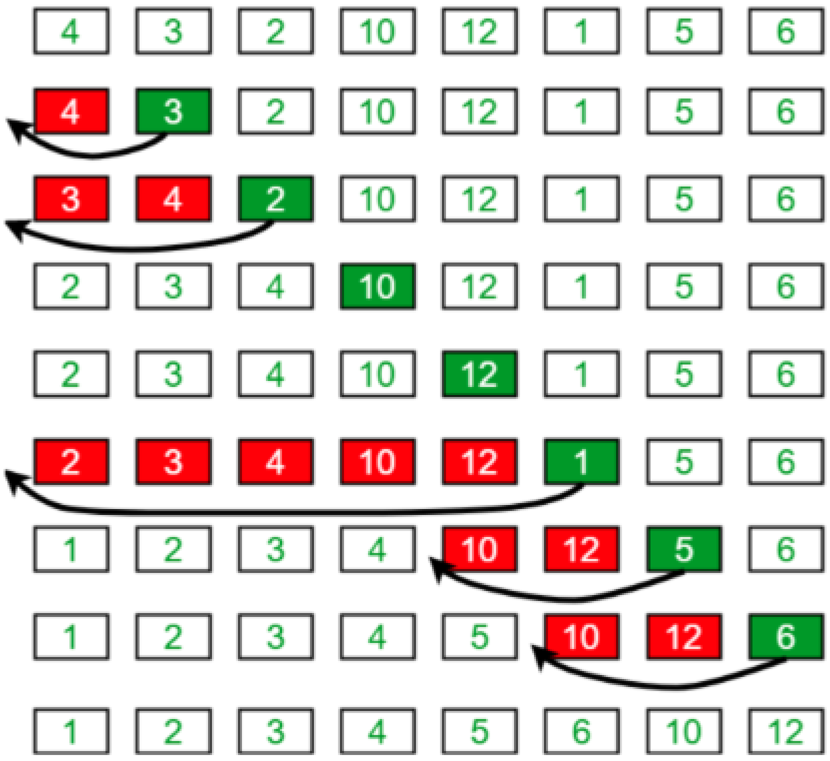




```java
int[] arr = {5, 1, 4, 2, 8};

// Perform bubble sort
int temp = 0;
for (int i = 0; i < arr.length - 1; i++) {
    for (int j = 0; j < arr.length - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
            // Swap elements
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }
    }
}
```

Insertion Sort



```
int[] array = {90, 12, 34, 27, 78};

// Perform insertion sort
int key = 0;
int idx = 0;
for (int i = 1; i < arr.length; i++) { // start from the second element
    key = arr[i]; // let the current element be "key"
    idx = i - 1; // the index of the first element to compare (from right to left)
    while (idx >= 0 && arr[idx] > key) { // check if the element > key
        arr[idx + 1] = arr[idx]; // move the element to right
        idx--;
    } // exit if the key > the left element, and < the right element, then it is the right position to insert
    arr[++idx] = key; // insert key
}
```

Sorting Array in Descending Order

```
// Your trun, try it by bubble sort.
```

Bonus: Counting Algorithms

Sometimes we need to count the number of elements in an array, and find out the max number of an element.

We can achieve it by an array directly.

Example 1: int[]

```java
int[] numbers = {1, 2, 3, 2, 1, 3, 1, 4, 5, 4};

int[] countArray = new int[numbers.length];

// Count occurrences of each number
for (int i = 0; i < numbers.length; i++) {
    countArray[numbers[i]] += 1;
}

// Find the maximum count
int maxCount = 0;
for (int count : countArray) {
    if (count > maxCount) {
        maxCount = count;
    }
}

// Print the maximum count
System.out.println("Maximum count: " + maxCount);
```

Example 2: char[]

- Assume the array contains the lowercase letter only.

```java
char[] chars = new char[] { 'a', 'b', 'z', 'a', 'z', 'c', 'a', 'z', 'z' };
int[] counts = new int[26];
// store the alphabet as position of the array.
```

```java
for (char c : chars) {
    counts[c - 'a']++;
}
// find the alphabet with the max count in counts array
int max = 0;
char alphabet = ' ';
for (int x = 0; x < counts.length; ++x) {
    if (counts[x] > max) {
        max = counts[x];
        alphabet = (char) (x + 97);
    }
}
System.out.println(alphabet); // z
```

Other array methods

```java
int[] arr = new int[10];

// Filling an Array
Arrays.fill(arr, 1);
System.out.println(Arrays.toString(arr)); // prints [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

arr = new int[]{1, 9, 8, 4, 6, 5, 2, 3, 7};

// Copying an Array
int[] arr2 = Arrays.copyOf(arr, arr.length);
System.out.println(Arrays.toString(arr2)); // prints [1, 9, 8, 4, 6, 5, 2, 3, 7]

// Comparison for equality
System.out.println(arr == arr2); // false
System.out.println(Arrays.equals(arr, arr2)); // true

// Sorting an Array
Arrays.sort(arr);
System.out.println(Arrays.toString(arr)); // prints [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Challenge

```java
int[] arr = { 32, 87, 3, 589, 12, 1076, 2000, 8, 622, 127 };
```

```java
int target = 12;

int i;
boolean foundIt = false;

for (i = 0; i < arr.length; i++) {
    if (arr[i] == target) {
        foundIt = true;
        break;
    }
}

if (foundIt) { // if foundIt == true
    System.out.println("Found " + target + " at index " + i);
} else {
    System.out.println(target + " not in the array");
}
// what is the print out?
```

Questions

- Read all examples, again and again.