Fundamental and Brute Force Algorithms

Note: Many of these exercises require that you create a program to manipulate an array of 10 grades. In this instance, you can do all these actions in a single program, rather than creating a new one for each challenge.

One of the most fundamental algorithm techniques is "brute force". Algorithms employing this approach check every possible option, go through every possible slot in an array and essentially try every possible version of the answer to find the right one. In these exercises, you are practising applying that technique across a variety of core actions.

Before you start:

Create a Java class called `ArrayUtils.java`. It should be in a package called `utils`

Displaying all elements in an array

This algorithm is extremely straightforward and is probably the algorithm you'll use most in all of the ones you develop for this course.

Exercise 1.1) Displaying all elements in a numeric array

Write a static method called displayArray() in ArrayUtils.java that:

- Takes in an **`integer`** array
- Prints out all the elements in the array (and their position in the array). This should not use the Arrays class!

Write a JavaDoc for this method. It should:

- Explain what the method does (one line description)
- Explain what the parameter is used for (@param tag)
- There should be no return tag, as it's a void method (i.e. the method does not return anything)

Commit and push your code:

- When your method is complete (your commit message for this should use the "feat" label)
- When you have added your Javadoc (your commit message for this should use the "docs" label)

Exercise 1.2) Displaying all elements in a String array

Write a static method called displayArray() in ArrayUtils.java that:

- Takes in a **`String`** array
- Prints out all the elements in the array (and their position in the array). This should not use the Arrays class!

Write a JavaDoc for this method. It should:

- Explain what the method does (one line description)
- Explain what the parameter is used for (@param tag)
- There should be no return tag, as it's a void method (i.e. the method does not return anything)

Commit and push your code:

- When your method is complete (your commit message for this should use the "feat" label)
- When you have added your Javadoc (your commit message for this should use the "docs" label)

Calculating the average of an array

Calculating the average of an array of numbers is a useful ability (and a simple algorithm to start with). You will have done this in first year, this is just to refresh your memory and so the instructions included are less detailed. Note: You cannot use any library method to achieve this.

Exercise 2.1) Calculating the average of an array

Write a static method called calcAverage() in ArrayUtils.java that:

- Takes in an **`integer`** array
- Calculates the average of the array
- Returns that number

Write a JavaDoc for this method. It should:

- Explain what the method does (one line description)
- Explain what the parameter is used for (@param tag)
- Explain what the method returns (@return tag)

Commit and push your code:

- When your method is complete (your commit message for this should use the "feat" label)
- When you have added your Javadoc (your commit message for this should use the "docs" label)

Exercise 2.2) Using Your Method

Write a new program (file) called `ArrayManipulation.java`. It should be in a package called `apps`:

- Asks the user to enter 10 grades and stores them in a numeric array (use your `getValidInteger()` method from **`InputUtility.java`** sheet 1 to do this reliably)
- Uses your method to calculate the average of these numbers
- Displays the GPA (grade point average)

Commit and push your code:

- When your program is storing all ten numbers in the array (your commit message for this should use the "feat" label)
- When your program is complete (your commit message for this should use the "feat" label)

Brute Force Approaches

*Brute force approaches involve checking **all** potential solutions and identifying the most suitable one. Searching for an element in an array and identifying the maximum or minimum element of an array are fundamentally suitable for a brute force approach.*

Finding the maximum element in an array

Finding the max and the min in a data set are extremely useful abilities. You will have done this in first year, this is just to refresh your memory and so the instructions included are less detailed. Note: You cannot use any sorting method to achieve this.

Exercise 3.1) Finding the maximum number in an array

Write a static method called findMax() in ArrayUtils.java that:

- Takes in an **`integer`** array
- Finds the highest number in the array
- Returns that number

Write a JavaDoc for this method. It should:

- Explain what the method does (one line description)
- Explain what the parameter is used for (@param tag)
- Explain what the method returns (@return tag)

Commit and push your code:

- When your method is complete (your commit message for this should use the "feat" label)
- When you have added your Javadoc (your commit message for this should use the "docs" label)

Exercise 3.2) Finding the maximum element in an array of Strings

Note: The compareTo (or compareToIgnoreCase) method from the String class will be useful here

Syntax: word1.compareTo(word2)

Returns: **< 0** if they're in ascending alphabetical order (i.e. word1 would come before word2 in the dictionary)
> 0 if they're in descending alphabetical order (i.e. word1 would come after word2 in the dictionary)
= 0 if they are the same.

Write a static method called findMax() in ArrayUtils.java that:

- Takes in a **`String`** array
- Finds the element that would appear **last** in an array of text sorted in alphabetical order
- Returns that element

Write a JavaDoc for this method. It should:

- Explain what the method does (one line description)
- Explain what the parameter is used for (@param tag)
- Explain what the method returns (@return tag)

Commit and push your code:

- When your method is complete (your commit message for this should use the "feat" label)
- When you have added your Javadoc (your commit message for this should use the "docs" label)

Exercise 3.3) Using Your Maximum Number Method

Amend your `ArrayManipulation` program to add the following:

- Take in 10 grades from the user (use your `getValidInteger`() method from **`InputUtility.java`** sheet 1 to do this reliably)
- Finds the highest grade the user achieved and displays it

Commit and push your code:

- When your program successfully takes in ten numbers and finds the highest value (your commit message for this should use the "feat" label)

Exercise 3.4) Using Your Maximum Text Method

Amend your `ArrayManipulation` program to add the following:

- Takes in 10 pieces of text from the user
- Finds the piece of text that would appear last in the dictionary and displays it.

Commit and push your code:

- When your program successfully takes in ten lines of text and finds the one that will appear last in the dictionary (your commit message for this should use the "feat" label)

Finding the minimum element in an array

As with finding the maximum element, you will have done finding the minimum in first year. This is just to refresh your memory and so the instructions included are less detailed. Note: You cannot use any sorting method to achieve this.

Exercise 4.1) Finding the minimum number in an array

Write a static method called findMin() in ArrayUtils.java that:

- Takes in an **`integer`** array
- Finds the lowest number in the array
- Returns that number

Write a JavaDoc for this method. It should:

- Explain what the method does (one line description)
- Explain what the parameter is used for (@param tag)
- Explain what the method returns (@return tag)

Commit and push your code:

- When your method is complete (your commit message for this should use the "feat" label)
- When you have added your Javadoc (your commit message for this should use the "docs" label)

Exercise 4.2) Finding the minimum element in an array of Strings

Note: As with identifying the maximum of two Strings, compareTo (or compareToIgnoreCase) is useful to locate the minimum. A.compareTo(B) will return < 0 if A is the minimum of the two.

Write a static method called findMax() in ArrayUtils.java that:

- Takes in a **`String`** array
- Finds the element that would appear **first** in an array of text sorted in alphabetical order
- Returns that element

Write a JavaDoc for this method. It should:

- Explain what the method does (one line description)
- Explain what the parameter is used for (@param tag)
- Explain what the method returns (@return tag)

Commit and push your code:

- When your method is complete (your commit message for this should use the "feat" label)
- When you have added your Javadoc (your commit message for this should use the "docs" label)

Exercise 4.3) Using Your Minimum Number Method

Amend your `ArrayManipulation` program to add the following:

- Takes in 10 grades from the user (use your `getValidInteger`() method from **`InputUtility.java`** in sheet 1 to do this reliably)
- Finds the lowest grade the user achieved and displays it

Commit and push your code:

- When your program successfully takes in ten numbers and displays the smallest value entered (your commit message for this should use the "feat" label)

Exercise 4.4) Using Your Minimum Text Method

Amend your `ArrayManipulation` program to add the following:

- Takes in 10 pieces of text from the user
- Finds the piece of text that would appear first in the dictionary and displays it.

Commit and push your code:

- When your program successfully takes in ten lines of text and finds the one that will appear first in the dictionary (your commit message for this should use the "feat" label)

Counting the Frequency of a Number in an Array

Finding out how many times something appears in an array can be very useful (e.g. for voting!) This is essentially a search for a specific element, tracking how many times the match appears – another example of a brute force approach.

5.1) Counting the number of times a number appears in an array

Write a static method called count() in ArrayUtils.java that:

- Takes in an **`integer`** array (nums) and an **`integer`** to count the frequency of (value)
- Counts how many times value appears in the array, and returns it

5.2) Using Your Methods

Amend your `ArrayManipulation` program to add the following:

- Asks the user to enter 10 grades and stores them in a numeric array
- Uses an appropriate method you have written to count in how many subjects the user scored 70.
- Displays the count.

Commit and push your code:

- When your program successfully displays the number of times 70 appears in the entered array (your commit message for this should use the "feat" label)

Finding the Most Frequent Element

Code reuse is an extremely good practice to get into. Not copy and paste, but using methods you've already built to solve the same problem in other places. Here you need to find out the frequency of each element in an array, and decide which one has the highest. You have a method to calculate the frequency of an element in an array, so use it here!

6.1) Finding the most frequent element in an array

Write a static method called getMostFrequent() in ArrayUtils.java that:

- Takes in an **`integer`** array (nums)
- Tracks the element in the array with the highest frequency by doing the following for each element:
  - Calculating the frequency for the element
  - If it's higher than the current highest, save it as the current highest
  - If not, discard it
- Return the number with the highest frequency at the end of the loop

Write a JavaDoc for this method. It should:

- Explain what the method does (one line description)
- Explain what the parameter is used for (@param tag)
- Explain what the method returns (@return tag) Remember to explain what happens if the array is empty, or if there are multiple elements with the same frequency.

Commit and push your code:

- When your method calculates the frequency of each element (your commit message for this should use the "feat" label)
- When your method correctly identifies the most frequent element (your commit message for this should use the "feat" label)
- When you have added your Javadoc (your commit message for this should use the "docs" label)

7.3) Using Your Methods

Amend your `ArrayManipulation` program to add the following:

- Asks the user to enter 10 grades and stores them in a numeric array
- Uses an appropriate method you have written to find the most frequently occurring grade in the array.
- Displays that grade to the user.

Commit and push your code:

- When your program successfully displays the most frequently appearing grade in the entered array (your commit message for this should use the "feat" label)

Calculating How Many Elements Are Greater Than the Average

As with the previous challenge, code reuse is helpful here. You need to find how many elements are greater than the average value. You have a method to calculate that, so use it as part of your countGreaterThanAverage method!

Another useful concept is that of "wrapper methods" – a method that calls another one with a fixed parameter. Here, you make a general countGreater method that works with any value, then create a wrapper method that calls the countGreater method but only ever supplies it with the numeric array and the average for that array

7.1) Counting How Many Are Greater Than a Value

Write a static method called `countGreater()` in ArrayUtils.java that:

- Takes in an **`integer`** array (nums) and an **`integer`** (value)
- Counts how many elements in the nums array are greater than the supplied number (value)
- Returns the count

Write a JavaDoc for this method. It should:

- Explain what the method does (one line description)
- Explain what the parameter is used for (@param tag)

- Explain what the method returns (@return tag)

Commit and push your code:

- When your method functions correctly (your commit message for this should use the "feat" label)
- When you have added your Javadoc (your commit message for this should use the "docs" label)

7.2) Counting How Many Are Greater Than the Average

Reminder: You can use any of the methods you already wrote to get this working!

Write a static method called `countGreaterThanAverage()` in ArrayUtils.java that:

- Takes in an `integer` array (nums)
- Calculates the average of the array
- Counts how many elements in the nums array are greater than the average of that array
- Returns the count

Write a JavaDoc for this method. It should:

- Explain what the method does (one line description)
- Explain what the parameter is used for (@param tag)
- Explain what the method returns (@return tag)

Commit and push your code:

- When your method correctly identifies the average of the supplied array (your commit message for this should use the "feat" label)
- When your method correctly counts the number of elements greater than the average of the array and returns it (your commit message for this should use the "feat" label)
- When you have added your Javadoc (your commit message for this should use the "docs" label)

7.3) Using Your Methods

Amend your `ArrayManipulation` program to add the following:

- Asks the user to enter 10 grades and stores them in an integer array
- Uses an appropriate method you have written to count in how many subjects the user scored higher than their GPA.
- Displays the count.

Commit and push your code:

- When your program successfully takes in ten numbers and finds the highest value (your commit message for this should use the "feat" label)