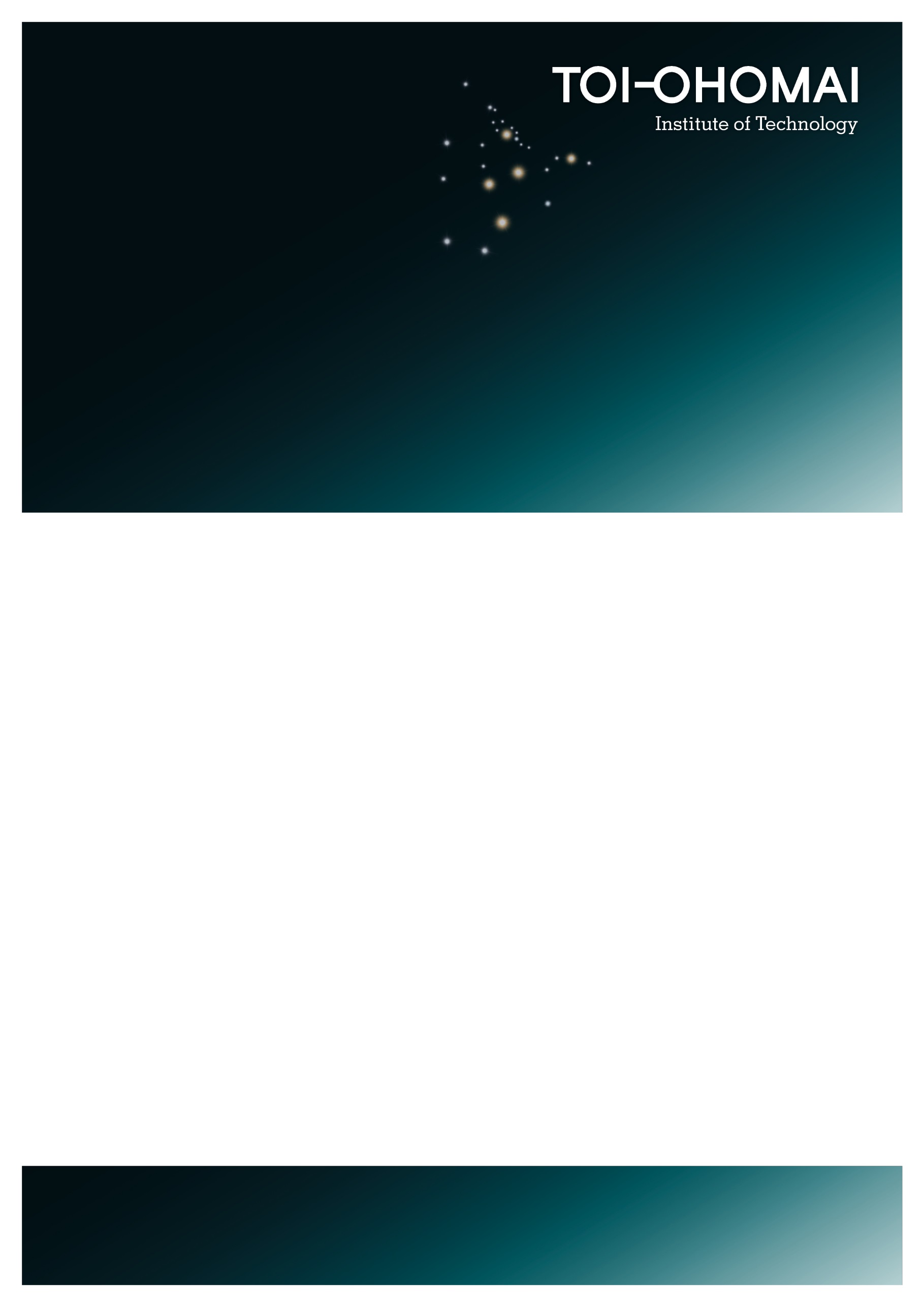
**

Assignment 2: Sorting and Searching

Date due: 31 May 2019

Weighting: 30%

Semester 1, 2019

COMP.6211

**Algorithms and Data Structures**

Instructions

**NOTE:** This assignment is an **individual** assignment. **Plagiarism will not be tolerated**. It is your responsibility to clarify any aspect of the assignment that you are unsure of, with your lecturer.

You are required to up a zip file with your solutions to Moodle before due date. The assignment will not be accepted using any other method.

Learning Outcomes Covered

1. Implement common data structures and algorithms.

Marks allocation

|  |  |
| --- | --- |
| **Question No.** | **Marks** |
| **1** | **18** |
| **2** | **16** |
| **3** | **14** |
| **4** | **17** |
| **Total** | **65** |

# Question 1 (18 marks)

**SETUP - for this entire assignment;**

You are required to write a console application with an **Algorithm** class

This class should contain a **Numbers** array which contains 50 integer elements.

These elements are to be generated at random.

You are to write a **Display** method, which will display the contents of this array to the screen.

Add the appropriate code to Main, so the numbers are displayed.

The particular challenge for this question is to then add a *findMaximum* method to your program that will receive two parameters:

* an array
* an integer (n = number of maximum elements to be found)

This method has the job of displaying the n largest elements in the array it receives and displaying it to the screen as shown in the sample program below.

**NOTE:** Do not use the array sort method to complete this challenge.  
 **Demo showing how program and method should work:  
*(NOTE: the demo is using a much smaller array than you are required to make)***

## Marking Criteria

|  |  |  |
| --- | --- | --- |
| **Description** | | **Marks** |
| Algorithm class created | | 1 |
| Numbers array populated with 50 random elements | | 2 |
| Display method - displays the array contents - called | | 3 |
| Prompts for and obtains input from user | | 1 |
| *findMaximum* method; created | 1 | 8 |
| *Has correct parameters* | 2 |  |
| *Algorithm to find n max numbers* | 5 |  |
| Display n largest numbers on screen |  | 1 |
| Comments |  | 2 |
| **TOTAL** |  | **18** |

# Question 2 (16 marks)

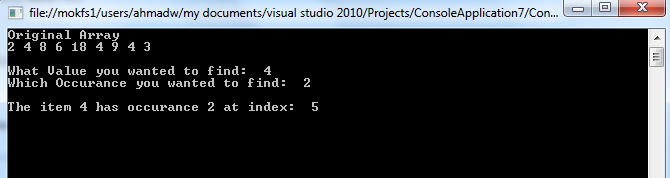
When using a sequential search algorithm (such as linear) to find a value, it will always find the first occurrence of that item in the array.   
  
*For example consider an array intArray with values 2, 4, 8, 6, 18, 4, 9, 4, 3.   
If the user wanted to search for the number 4 in the array, a sequential search algorithm will return the first occurrence of 4 (that is at index 1 in the intArray).*

For this question you are required to extend the program and class created in question 1 by adding a *NumOccuranceSearch* method, which is a modified sequential search that receives three parameters:

* an array
* an integer (the value the user wants to search for i.e. 4)
* and a second integer (the occurrence of the item they want to search for i.e. 2nd).

The method is to display the index of the occurrence of that number to the screen as shown in the sample program below.

**Demo showing how the method should work:  
*(NOTE: the demo is using a much smaller array than you are required to make)***



## Marking Criteria

|  |  |  |
| --- | --- | --- |
| **Description** | | **Marks** |
| Prompts for and obtains input from user | | 2 |
| *NumOccuranceSearch* method; created | 1 | 9 |
| *Has correct parameters* | 3 |  |
| *Algorithm to find an occurrence of a number* | 5 |  |
| Display details on screen |  | 3 |
| Comments |  | 2 |
| **TOTAL** |  | **16** |

Question 3 (14 marks)

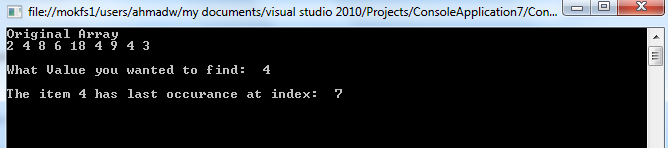
For this question you are required to extend the program and class even further by adding a *LastOccuranceSearch* method, which finds the **last** occurrence of the searched item.

The method should receive two parameters:

* an array
* an integer (the value the user wants to search for i.e. 4)

And return the index of the last occurrence to Main, so it can be displayed.

**Demo showing how the method should work:  
*(NOTE: the demo is using a much smaller array than you are required to make)***



## Marking Criteria

|  |  |  |
| --- | --- | --- |
| **Description** | | **Marks** |
| Prompts for and obtains input | | 1 |
| *LastOccuranceSearch* method; created | 1 | 9 |
| *Has correct parameters and return* | 3 |  |
| *Algorithm to find last occurance* | 5 |  |
| Display details on screen |  | 2 |
| Comments |  | 2 |
| **TOTAL** |  | **14** |

# Question 4 (17 marks)

As you know the bubble sort algorithm isn’t the fastest method for sorting data, but it could be improved.

The challenge for this question is to look at improving the bubble sort. Consider the following:

*You will notice after the first pass of the algorithm the highest number is “in place”, after the second pass the two highest numbers are “in place” and so on. Consider as the array elements are moved so the highest numbers in place, do we need to continue to make 49 passes of a 50 element array? Or can this be reduced?*

For this question you are required to extend your program and class again by adding the following two methods:

* ***bubbleSort ()***
* ***improvedBubbleSort ()***

The ***bubbleSort ()*** method will implement the standard bubble sort algorithm and time how long the takes to sort the values in your array.

The ***improvedBubbleSort ()*** method will implement the following two improvements and time how long it takes to sort the same data:

1. Modify the algorithm so instead of making 49 comparisons on every pass of a 50 element array, it will now make 48 comparisons on the second pass, 47 on the third pass and so on.
2. Also modify the algorithm to check at the end of each pass whether any swaps have been made. If none have been made, the data must already be in the proper order, so the algorithm should terminate.

Both methods should output the resulting times to the screen to see if any improvement in times has been achieved.

### Marking Criteria

|  |  |  |
| --- | --- | --- |
| **Description** | | **Marks** |
| ***bubbleSort***  method | | 4 |
| ***improvedBubbleSort*** method |  | 8 |
| Improvement 1 implemented | 4 |  |
| Improvement 2 implemented | 4 |  |
| Timer implemented and displayed | | 3 |
| Comments | | 2 |
| **TOTAL** | | **17** |

----------------------------------------------------------------The End--------------------------------------------------------