# 7SENG010W Data Structures & Algorithms
# Week 2 Tutorial Exercises

These exercises cover: Searching and Sorting Arrays

Any exercises not completed during the tutorial should be completed by the following week.

**Exercise 1.**
Using the online data structures and algorithm visualization tool developed by David Galles, University of San Francisco: https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

(a) Experiment doing binary searches using the Binary Search algorithm visualization tool at: https://www.cs.usfca.edu/~galles/visualization/BST.html

(b) Experiment by comparison different sorting algorithms, e.g. selection, bubble and merge, using the following tool: https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

**Exercise 2.**
(a) Implement a C# version of the Binary Search algorithm as presented in the Week 2 Lecture. Base your solution on the pseudo code for Binary Search algorithm given in the Lecture.

(b) Create a `Testing` class with a `Main` method that creates and runs your Binary Search algorithm method on several sample arrays, such as those used in the Lecture. Make sure that you test for different values, i.e. in and not in the array.

**Exercise 3.**
(a) Implement a C# version of the Bubble Sort algorithm as presented in the Week 2 Lecture. An explanation and example of the algorithm is given in the Lecture.

Hint: start from the Selection Sort algorithm implementation and modify the code to switch from “selection” sorting to “bubble” sorting.

(b) Add code to your `Testing` class's `Main` method of Exercise 2 to create and run your Bubble Sort on several sample arrays, e.g. those used in the Lecture. Make sure that you test for different types of arrays, e.g. random values, already sorted in ascending order and already sorted in descending order, all the same values, etc.

**Exercise 4.**
Read the documentation for the `Random` class, and review the `StopWatch` class & the `TimeSpan` struct, see:

https://docs.microsoft.com/en-us/dotnet/api/system.random?view=net-6.0#methods

https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch?view=net-6.0

https://docs.microsoft.com/en-us/dotnet/api/system.timespan?view=net-6.0

(a) Modify your Test program's `Main` method to use much larger arrays doubling the size of the array, e.g. 1000, 2000, 4000 & 8000 or even larger. Fill these arrays with random numbers using the methods of the `Random` class.

(b) Then run them and using the same code timing features from Tutorial 1, print out the execution times for the searching and sorting methods and record them in a table.

| Values of N (Array size) | T(N) - Execution times in seconds/milliseconds, | | |
|---|---|---|---|
| | Binary Search | Selection Sort | Bubble Sort |
| 1000 | | | |
| 2000 | | | |
| 4000 | | | |
| 8,000 | | | |
| ... | | | |

Can you draw any conclusions from the results?

**OPTIONAL Exercise 5.**

(a) Complete the code of the Merge Sorting algorithm as given in the Lecture by defining the MergeSort's `mergeRanges method:`

```
private static void mergeRanges( int[] values,
                                 int first, int mid, int last )
{
      // OPTIONAL TO DO EXERCISE
}
```

Hint: run the algorithm visualisation tool at:
https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html
on the Merge sorting algorithm by “stepping ” through it or slowing the “Animation Speed” to very slow and follow what it is doing.

(b) Time the algorithm as in Exercise 4. How does it compare to the other sorting algorithms?

(c) If you have not completed the above code then find an implementation online and try to understand how the “segment merging” process code works.