7SENG010W Data Structures & Algorithms
Week 1 Tutorial Exercises

These exercises cover: Abstract Data Types (ADTs), Big-O Complexity, Timing an Algorithm

Exercise 1.

(a) Using C# & following the guidelines for implementing an ADT, create an ADT - `TubeStation` class that represents a London tube station. It should hold the following information about a single tube station:

`StationID`	– a unique ID number for the station
`Name`	– `the` station's name
`TubeLine`	– name of the tube line the station is on, if more than one then just pick one.
`StationAccess`	– how passengers get to the platforms, i.e. stairs, lift or escalator.
`TravelZone`	– one of the possible travel zones 1 to 6.
`StationStatus`	– whether the station is open or closed to passengers.

Define constructors, operations to access & set the above values, & a `ToString()` method.

For information about the London Underground see https://tfl.gov.uk/maps.

(b) Create a `Testing` class with a `Main` method that creates 2 instances of the `TubeStation` class for 2 different tube stations & prints out their information.

Exercise 2.
Complete the table for the quadratic growth rate running time equation $T(N) = 2N^2 + 3N + 4$. This gives an idea why when calculating the Big-O for a T(N) that only the value of the *dominant term*, i.e. N^2, determines its corder of complexity.

Values of N	$T(N) = 2N^2 + 3N + 4$			
	$2N^2$	$3N$	4	T(N)
1000				
2000				
4000				
8,000				
16,000				

Exercise 3.
Complete the table for the following common growth rates for Big-O for the values of N. To help you calculate these values see the list of online tools given in the module's *Software Requirements* document. Note that it may not be possible to calculate the value of some of these expressions for large N, in those cases experiment to find large values of N that they can be evaluated for.

Big-O	Values of N				
	20	**50**	**100**	**1000**	**100,000**
O(1)					
O(N)					
$O(N^2)$					
$O(N^3)$					
$O(\log_2(N))$					
$O(N \log_2(N))$					
$O(2^N)$					
O(N!)					

Note that when we deal with order of complexity expressions that involve $\log_2(N)$ the result is very rarely a whole number, so the standard practice is to take the smallest whole number that is greater than the fractional value, this is called the "ceiling" in maths e.g. ceiling(4.3219) = 5.

For comparison, the estimated age of the Universe is approximately 13.5 billion years, & in seconds that is: $(13.5 \times 10^9) \times (365 \times 24 \times 3600) = 4.25736 \times 10^{17}$.

Exercise 4.

(a) Read the documentation for the `StopWatch` class & the `TimeSpan` struct, see:

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

(b) Modify the array searching program from Lecture 1 & using the `StopWatch` class & `TimeSpan` struct, add suitable code to the Main method that records the execution time of calls of the `Search` method & prints out the elapsed time. (Hint: you should only need to consider seconds & milliseconds.) Does 1 call of `Search` take any time?

(c) Experiment by increasing the size of the array & search for a number not in it, or put the call of `Search` inside a for-loop & repeating the search a large number of times. How many for-loop iterations does it take to get close to 1 second?