7SENG010W Data Structures and Algorithms
Week 4 Tutorial Exercises: Stacks, Queues and Algorithm Analysis

These exercises cover: Stacks and Queues, and the .NET versions Stack<T> and Queue<T>, and the *empirical approach* to analysing algorithms.

You will also make use of David Galles' web based *Data Structure Visualizations* tools. During its use it may help to slow the animation right down so that you can follow the steps. You should also take screen shots of your built data structures and/or record its use.

You should **reuse** any code involving arrays and lists that you have previously produced for previous tutorials exercises.

Exercise 1.

(a) Implement a **static** fixed sized (15) stack of integers by completing the given C# class ArrayStack.cs, or by defining your own class. The following operations and enquires should be provided:

Push(int), Pop(), Clear(), Print()

Top(), Size(), isEmpty(), isFull()

(b) Test your implementation by defining a test program with code to test your stack using suitable test data, that performs all the operations and enquires on your stack. How does your implementation deal with attempting to push onto a full stack or pop from an empty stack?

(c) As part of the testing use David Galles' stack visualisation tool at:

https://www.cs.usfca.edu/~galles/visualization/StackArray.html

Use the same test data as in (b) to visualise your stack by perform the same sequence of test operations on it as in your test program. For the values of the enquiry operations determine by inspection of the stack visualisation. If your test program's output does not match those produced in the tool, make sure you understand what your mistake is and correct your code. You should take screen shots of your testing.

Exercise 2.

(a) Implement a **dynamic** queue of zoo animals by completing the given C# lass ListQueue.cs, or by defining your own class. The following operations and enquires should be provided:

Enqueue(string), Dequeue(), Clear(), Length(), isEmpty(), Print()

Front() - returns value of the first item in the queue, but does not remove from queue.

Last() - return value of the last item in the queue, but does not remove from queue.

(b) Test your implementation by adding code to your test program to test your queue, ensure that it tests all the operations and enquires on your queue. How does your implementation deal with attempting to dequeue from an empty queue?

(c) As part of the testing use David Galles' visualisation tool for queues at:

https://www.cs.usfca.edu/~galles/visualization/QueueLL.html

Use the same test data as in (b) to visualise your queue by perform the same sequence of test operations on it as in your test program. For the values of the enquiry operations determine by inspection of the queue visualisation. If your test program's output does not match those produced in the tool, make sure you understand what your mistake is and correct your code. You should take screen shots of your testing.

Exercise 3.

You are given the following table of execution times T(N) for a range of inputs for 4 algorithms:

Input Size N	**Execution Time T(N) values for Algorithms**			
	Alg-1	**Alg-2**	**Alg-3**	**Alg-4**
1000	15	7	23	37
2000	63	58	31	92
4000	248	453	39	214
8000	1020	3713	47	465
16000	4050	29503	55	919
32000	16501	237836	62	1918

Analyse these execution times using the empirical approach by applying the *"Doubling Hypothesis", i.e. doubling the size of the input (N) and determine the effect on the rate of growth on the execution times T(N).* From this infer the Big-O complexities of the 4 algorithms.

OPTIONAL Exercise 4.

(a) Using the .NET Framework Library Stack<T> class from System.Collections.Generics namespace. Add code to your test program to create the same stack as in **Exercise 1**, but using the Stack<T> class.

(b) Using the .NET Framework Library Queue<T> class from System.Collections.Generics namespace. Add code to your test program to create the same queue as in **Exercise 1**, but using the Queue<T> class.

(c) Do the Stack<T> and Queue<T> versions of the same stack and a queue produce the same results as those of your own hand coded versions? If they do not which is most likely to be correct and what should you do about it, email Microsoft that there is a bug in their code (not unheard of) or check your own versions for bugs?

(d) As an open question to think about -
What would you do if they disagree and both produce results that are not what you expected and/or also disagree with David Galles' visualisation tools?