UNIVERSITY OF THE FRASER VALLEY
SCHOOL OF COMPUTING

Assignment 1 — Algorithm Analysis
COMP 251 – AB1 : Data Structures & Algorithms

(100 points)

Instructor's Name: Opeyemi Adesina, Ph.D.
Instructor's UFV Email: opeyemi.adesina@ufv.ca
Office: ABB-C2401

When Due: October 8, 2025 – 23:59:00 (PDT) [Submission via Blackboard]

A. Background

The goal of this assignment is to assess your skills in developing implementations for the List abstract data type while understanding their complexities through executions. This assignment amounts to **10%** of the entire course grade. This is required to be done **in groups of 2-3 students**. Whatever you obtain as a score will be scaled to this value for final grade computation. Specifically, for final computation your earned points x is converted using this: $[x \text{ (points)} = 10x/100]$ %. No late submission will be permitted (see deadline above), unless approved or granted by the course instructor.

In the real world, there are several solutions to most problems. This is equally true for the computational world. For example, in the Java programming language, the development of List has been realized with array-based (aka ArrayList) and linked-based (aka LinkedList) implementations. To avoid conflicts, we shall be referring to these implementations as **ArrayBasedList** (self-expanding or auto-growing) and **SinglyLinkedList** in our work. This leaves us with analyzing these implementations to determine which is best for the problem at hand (particularly, understanding the implications of our design for significant operations). Of course, we understand there is no one-size-fits-all solution for problems of this nature, notwithstanding - understanding the implications of our designs could help us make informed decisions.

B. List Abstract Data Type (ADT)

The Java API is a detailed list of permissible operations (Javadoc) on ArrayList and LinkedList. However, in Table 1 we present a list of operations (and their descriptions) that are most interesting to us (which is a proper subset of the list referenced for the original Java implementations for both cases) for the purpose of this assignment. Similarly, in Figure 1, we present a class model of the internal structure of the expected implementation. You are required to base your implementation on generics so that your implementations will be type-independent.

Table 1: Required Application Programming Interfaces

Operations	**ArrayBasedList**	**SinglyLinkedList**
Constructors	**ArrayBasedList()** Constructs an empty list with an initial capacity of ten. **ArrayList(int initialCapacity)** Constructs an empty list with the specified initial capacity.	**SinglyLinkedList()** Constructs an empty list.

[1] https://cruise.umple.org/umpleonline/umple.php?model=23101038cnvt3me08r

Operations	ArrayBasedList	SinglyLinkedList
	Modifiers	
	void add(T element) Appends the specified element to the end of this list. **void add(int index, T element)** Inserts the specified element at the specified position in this list. **void set(int index, T element)** Replaces the element at the specified position in this list with the specified element. **T remove(int index)** Returns and removes the element at the specified position in this list. **T remove(T element)** Returns and removes the first occurrence of the specified element from this list, if it is present.	**void add(T element)** Appends the specified element to the end of this list. **void add(int index, T element)** Inserts the specified element at the specified position in this list. **void set(int index, T element)** Replaces the element at the specified position in this list with the specified element. **T remove(int index)** Returns and removes the element at the specified position in this list. **T remove(T element)** Returns and removes the first occurrence of the specified element from this list, if it is present.
	Accessors	
	boolean contains(T element) Returns true if this list contains the specified element. **T get(int index)** Returns the element at the specified position in this list. **int indexOf(T element)** Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. **ADTList< T > subList(int fromIndex, int endIndex)** Returns a view of the portion of this list between the specified $fromIndex$, inclusive, and $endIndex$, exclusive.	**boolean contains(T element)** Returns true if this list contains the specified element. **T get(int index)** Returns the element at the specified position in this list. **int indexOf(T element)** Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. **ADTList< T > subList(int fromIndex, int endIndex)** Returns a view of the portion of this list between the specified $fromIndex$, inclusive, and $endIndex$, exclusive.

C. Tasks To Be Completed

You are required to complete the following tasks (and make a video of yourself, describing the solution you have provided and upload a zipped file of your project and a detailed report with your names clearly indicated):

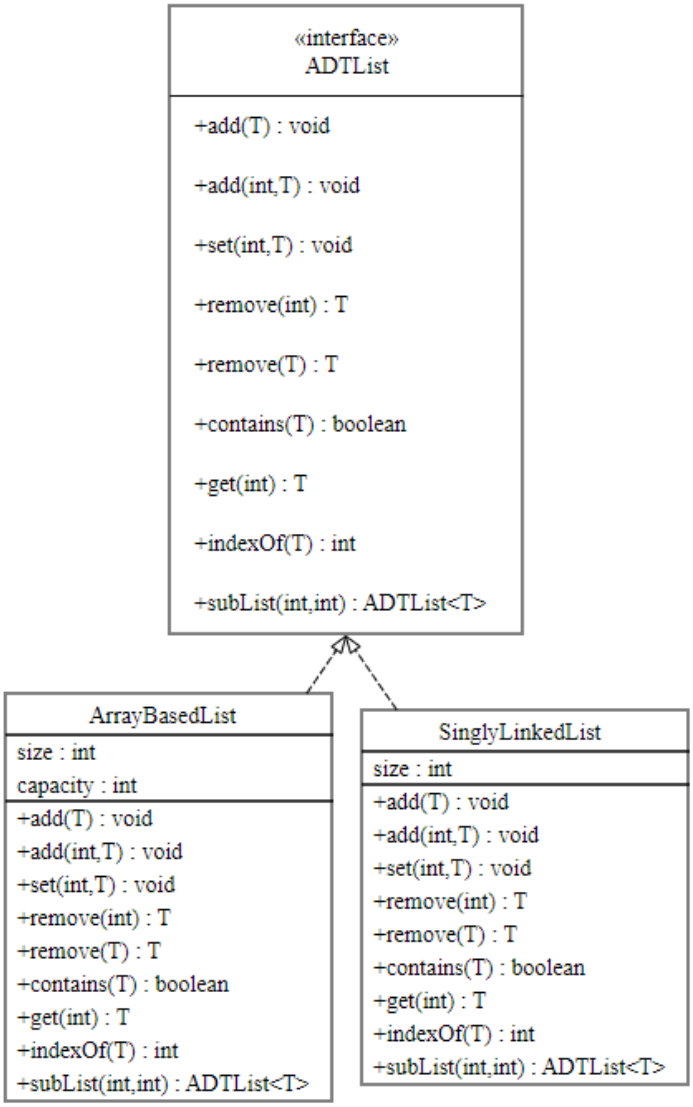

Figure 1: ADTList Specification Model. Available on UmpleOnline[1]

50 points Implement ArrayBasedList and the SinglyLinkedList with the specification (or requirements) presented in Table 1. Implementation of other relevant or helper methods should have private accessibility specifications while all APIs shall have public access modes. I will be developing test cases to assess the correctness of your implementations. Make sure your implementations depend on the $ADTList < T >$ interface.

30 points For each pair of accessor and modifier methods, conduct a performance study (using different input sizes like 100, 1,000, 10,000, 100,000, 250,000).

(a) For each input size, generate at least two test cases.
(b) For each test case, run the program at least three times (preferably in milliseconds) collect the time value, and take the average of all the runs.
(c) Plot the results (average execution time against the input size) on the same graph. You may use a log scale for better/clearer visualizations of the results.

10 points In tabular form, compare the best-case and worst-case time and space complexities of each pair of algorithms you have developed for the methods in Table 1.

10 points Discuss your findings and empirical results. Particularly, which of the implementations do you

think would work best for any particular situations you could think of assuming you are a Solution Architect?

D. Academic Integrity Statement

By submitting this assignment, you pledge to abide by the statements of Policy 70 of University of the Fraser Valley (UFV) including every other policy of the University that might be relevant to this subject matter. You agree that this submission is entirely yours but not a solution copied from the internet, written by a tutor (either paid or unpaid), or written by a friend or a senior student. You acknowledge to seek help from the instructor as often as may be required (either through office hours or intermittent drop-by or a scheduled appointment).

With the advent of generative AI, it is possible to employ AI-based systems to generate code. The use of AI is not prohibited as long as it doesn't obstruct learning. If you have used AI for any component of this assignment, you are required to document how you have used it and what you have learnt in relation the activities of this assignment. If you are not sure how you can use this technology, please contact the instructor on the do's and dont's with generative AI.