Assignment 1

Answer question 1, question 2, and **any other 2** questions from questions 3 to 6 – maximum 100 marks. You must score at least 50 to pass the assignment.

1. (25 + 15 = 40 marks) You have learned some fundamental data structure concepts such as array, queue and priority queue, stack, list and linked list, sequence, and unordered set, and you understand the concept of interface or abstract data type that defines the set of operations supported by a data structure and the semantics, or meaning, of those operations. You can use the interface of one particular data structure to define or implement the operations of a different data structure.

    a. (25 marks total) Describe the meaning of the essential methods `add(x)`, `deleteMin()`, and `size()` that are supported by the priority queue interface (5 marks).
    Implement those methods using a singly-linked list (5 marks for each method).
    Analyze the running time of the `add(x)` and `deletMin()` operations based on this implementation (5 marks).

    b. (15 marks total) Implement the stack methods `push(x)` and `pop()` using two queues (5 marks for each method).
    Analyze the running time of the `push(x)` and `pop()` operations based on this implementation (5 marks).

2. (10 + 10 = 20 marks) Swap two adjacent elements in a list by adjusting only the links (and not the data) using

    a. singly-linked list (10 marks).
    b. doubly-linked list (10 marks).

3. (20 marks) Exercise 1.5. Using a `USet`, implement a `Bag`. A `Bag` is like a `USet`—it supports the `add(x)`, `remove(x)`, and `find(x)` methods—but it allows duplicate elements to be stored. The `find(x)` operation in a `Bag` returns some element (if any) that is equal to `x`. In addition, a `Bag` supports the `findAll(x)` operation that returns a list of all elements in the `Bag` that are equal to `x`.

4. (20 marks) Exercise 2.3. Design and implement a *`RandomQueue`*. This is an implementation of the `Queue` interface in which the `remove()` operation removes an element that is chosen uniformly at random among all the elements currently in the queue. (Think of a `RandomQueue` as a bag in which we can add elements or reach in and blindly remove some random element.) The `add(x)` and `remove()` operations in a `RandomQueue` should run in constant time per operation.

5. (20 marks) Exercise 3.12. Write a method, `reverse()`, that reverses the order of elements in a `DLList`.

6. (20 marks) Exercise 3.14. Design and implement a `MinStack` data structure that can store comparable elements and supports the stack operations `push(x)`, `pop()`, and `size()`, as well as the `min()` operation, which returns the minimum value currently stored in the data structure. All operations should run in constant time.