Data Structures & Algorithms 2
Tutorial 3
Lists, Stacks, and Queues

OBJECTIVES

- Usage and implementation of basic data structures.

Exercise 1

Given two sorted lists, L1 and L2, write a procedure to compute L1 ∪ L2 using only the basic list operations. What is the running time of your procedure.

Exercise 2

Suppose that a singly linked list is implemented with both a header and a tail node. Describe constant-time algorithms to

a). insert item x before position p (given by an iterator)

b). remove the item stored at position p (given by an iterator)

Exercise 3

A deque is a data structure consisting of a list of items on which the following operations are possible:

push(x): Insert item x on the front end of the deque.

pop(): Remove the front item from the deque and return it.

inject(x): Insert item x on the rear end of the deque.

eject(): Remove the rear item from the deque and return it.

Write routines to support the deque that take O(1) time per operation.

Exercise 4

Propose a data structure that supports the stack push and pop operations and a third operation findMin, which returns the smallest element in the data structure, all in O(1) worst-case time.

Exercise 5

Write routines to implement two stacks using only one array. Your stack routines should not declare an overflow unless every slot in the array is used.

Exercise 6

An alternative to the deletion strategy is to use lazy deletion. To delete an element, we merely mark it deleted (using an extra bit field). The number of deleted and non-deleted elements in the list is kept as part of the data structure. If there are as many deleted elements as non-deleted elements, we traverse the entire list, performing the standard deletion algorithm on all marked nodes.

1. List the advantages and disadvantages of lazy deletion.
2. Write routines to implement the standard linked list operations using lazy deletion.

Exercise 7

The Circular Queue is similar to a Linear Queue in the sense that it follows the FIFO (First In First Out) principle but differs in the fact that the last position is connected to the first position, replicating a circle.

Implement in pseudo code a circular queue data structure using a 1D array. Once the implementation is complete, analyze the time complexity of various queue operations in terms of Big O notation. The following operations should be included in your analysis:

- **Front()** : Get the starting element of the Circular Queue.
- **Rear() :** Get the end element of the Circular Queue.
- **enQueue(value) :** Insert a new value in the Circular Queue. This operation takes place from the end of the Queue.
- **deQueue()** : Used to remove a value from the Circular Queue. This operation takes place from the front of the Queue.
- **empty ()** : check if the queue is empty
- **full()** : check if the queue is full
- **serach(value)** : search a value in the queue.
- **size()** : Find the size of the queue
- **Get(index) :** Access an element at a given index