Data Structures and Algorithms

Assignment 1

1.

(a) Priority Queue with Singly-Linked List:

A Queue is considered a linear data structure that takes a first in first out order of adding and removing or deleting data from its collection.

The method add(x) adds a piece of data to the priority queue while maintaining the priority order.

deleteMin() will take the lowest valued minimum element based on the priority from the priority queue remove it, and return it ultimately.

The size() method returns the total number of elements in the priority queue.

Using java I created a singly-linked list to maintain the elements of the priority queue. Each node in the list holds a value and its corresponding priority. Each node in a singly linked is is comprised of a piece of data and its pointer going to the next place in this case the queue. The add(x)traverses the linked list linearly to insert the new element based on its priority. The deleteMin() method removes the head of the lowest priority element in the list. Which corresponds is the minimum element. Whereas The size() method counts each piece of the linked list and returns the smallest piece of data or number in this case.

In the worst-case scenario running time of add(x) would be O(n) because we have to traverse the entire linked list to get to the last node which is where the correct position to insert the node of information is, where n is the number of elements in the priority queue. DeleteMin() has a run time of O(1) since we directly remove the minimum element from the head of the linked list. The running time of size() is O(n) since we need to traverse the entire linked list to count the elements.

B) Stack with Two Queues:

Two queues create a stack data structure. In stacks the terminology is different as the mechanism for adding and removing data is fundamentally different also. Stacks are a linear data structure that use first in last out order. The push(x) method adds x to one of the queues. When pop() is called, we can transfer all elements except the last one from one queue to the next interchangeably. The last element is then removed and returned as a popped element. This approach maintains the stack behavior while using two queues.

The running time of push(x) is O(1) as its directly appended to the element to one of the queues. The worst-case running time of pop() is O(n) because we need to move all elements from one queue to another to retrieve the last element, where n is the number of elements in the stack. The running time of size() is O(1) as it only returns the length of one of the queues.

1. A) Swap Adjacent Elements in Singly-Linked List:

Swapping adjacent elements in a singly-linked list can be done by adjusting the links between nodes. We use two pointers, prev and current, to keep track of the nodes that need to be swapped. We also keep a reference to the new head of the list, new\_head, which will be the second node after swapping.

We first check if the linked list is empty or has only one element. In these cases, no swapping is required, so we return the original head as the new head.

We set prev to None and current to the head of the linked list. Then, we set new\_head to the second node (if it exists) after swapping the first two nodes.

We iterate through the linked list using the while loop. For each pair of adjacent nodes, we swap their positions by adjusting the links: current.next becomes temp.next, and temp.next becomes current. Additionally, we update the link from prev to temp to ensure proper connections after swapping.

Finally, we return the new head of the modified linked list.

(b) Swap Adjacent Elements in Doubly-Linked List:

The Doubly linked is similar to the linked list however has a different element to take into consideration as there is a forward and backward pointer to the previous and next node. The approach is still similar to the singly-linked list, but with the addition of handling the previous pointers.

First the linked list is checked for whether it is empty or has only one element. In these cases, no swapping is required, so we return the original head as the new head.

We set prev to None and current to the head of the linked list. After that the new\_head is set to the second node , if there is one at all, after the first two nodes switch places. The linked list is then traversed iteratively . For each pair of adjacent nodes, we swap their positions by adjusting the links: current.next becomes temp.next, and temp.next becomes current. Additionally, we update the previous pointer of temp to point to prev, and the previous pointer of current to point to temp after swapping.

Finally, we return the new head of the modified linked list.

4.Implementing a RandomQueue:

Queues use a first in first out order of adding data and removing it from the queue. The terminology used for add is enqueue, whereas to remove data is to dequeue while also noting the enqueue part of the list is the at the rear and the dequeue in front.

The RandomQueue class is an implementation of the Queue interface, but with a additional property. When calling remove(), it will remove an element chosen at random from all the elements currently in the queue, hence the random feature. Each element in the queue has the same chance of being removed.

The RandomQueue uses a list to store the elements. When add(x) is called, the element x is added to the end of the list. When remove() is called, a random index is chosen, and the element at that index is removed and returned. This ensures that the elements are removed randomly from the queue.

1. Reversing the order of elements in a DLList:

The DLList class represents a doubly-linked list that allows elements to be Pushed and popped in a stack. Using this technique we can easily reverse the string as the way the first in last out nature is of a stack if using scanner for input can be done by adding the reversed strings to the secondary temp list. This effectively reverses the direction of the links and, hence, the order of elements. The time Complexity is O(N), for traversing over the string.