Problem 1:

Implement Stack using Queues

Using two queues implement stacks. Queue is a FIFO data structure. To implement Stack (LIFO) by queues, we need two queues.

Examples:

For example,

Insert 2, insert 3, pop

1. Insert 2 to pingQ,
2. Insert 3 to pangQ
3. Remove 2 from ping q and insert to pangQ
4. pangQ now contains two elements 3 and 2
5. Remove first element, 3 from the pangQ

Insert 2, pop

1. Insert 2 to pingQ
2. Remove the first element, 2 from the pingQ

Insert2, insert3, insert4, pop

1. Insert 2 to pingQ
2. Insert 3 to pangQ
3. Remove 2 from pingQ and insert to pangQ
4. PangQ contains 3 and 2
5. Insert 4 to pingQ
6. Remove 3 and 2 from pangQ to pingQ
7. pingQ contains 4, 3, 2
8. remove 4 from pingQ

Explanation:

I use two queues to implement the stack

For push operation, insert element into an empty Queue, a ping queue. if another Queue, a pang queue is not empty, then remove all elements from pang Q and insert them to ping queue

This way would let the last element become the first element in the ping queue.

Time complexity:

For push operation, time complexity would be O(n)
For pop, top and empty operation, time complexity would be O(1)
Space complexity is O(n)

Problem 2:
Given the head of a singly linked list, reverse the list, and return *the reversed list*.

Examples:
1,2,3,4,5

1. head is 1, use next=head.next which is 2, prev = head which is 1
2. head = next which is 2, then next is 3, let head.next = prev, and then prev=2
3. now there are two list node, 2->1, 3->4->5
4. head = next which is 3, next is 4, let head.next = prev, and then prev=3
5. 3->2->1, 4->5
6. head = next which is 4, next is 5, let head.next=prev, and then prev=4
7. 4->3->2->1, 5
8. head=next which is 5, next is null, let head.next=prev, and then prev=5
9. 5->4->3->2->1

Explanation:
Use 3 linklist variables, next, prev and head. Using a loop which detect if next is not null, in the loop

head = next;

next = head.next;

head.next = prev;

prev = head;

Time complexity:
Time complexity is O(n)
Space complexity is O(4)

Problem 3:
Given two strings s and t, return true if t is an anagram of s, and false otherwise.

Examples:

anagram and nagaram

1. Loop ‘anagram’, there is 3 ‘a’, 1 ‘n’, 1 ‘g’, 1 ‘r’ and 1 ‘m’
2. Loop ‘nagaram’, there is 3 ‘a’, 1 ‘n’, 1 ‘g’, 1 ‘r’ and 1 ‘m’
3. They are same, they are anagram

rat and car

1. Loop ‘rat’, there is 1 ‘r’, 1 ‘a’, and 1‘t’
2. Loop ‘car’, there is 1 ‘c’, 1’a’ and 1 ‘r’
3. They are different, the are not anagram.

Explanation:

Use an int array to represent 26 alphabets. If there is an alphabet exists in s string, increase the number of the correspond position in the int array by 1. If there is an alphabet exists in t string, decrease the number of the correspond position in the int array by 1.
If s and t are anagram, eventually the value of all of the position in the int array should be zero.

Time complexity:
Time complexity is O(2n)
Space complexity is O(26)