**Lab W1D1**

Problem: <https://leetcode.com/problems/implement-stack-using-queues/>

1. Explanation of the problem with examples (minimum three examples):

The problem asks us to implement stack using queue. Recall queue is FIFO while stack is LIFO. Stack only accepts Integer value and has functions to push, pop, top & empty. Example:

* Push 1 => Push 2 => Get Top will be 2.
* Push 1 => Push 2 => Get Empty will be False (Stack is not empty).
* Push 1 => Push 2 => Pop => Get Top will be 1.

1. Explanation of possible solution(s):

When analyzing the problem, because we must use queue to implement the stack, I realized that if every time we push an item into the queue, we can push it normally and then pop the remaining one by one (except the newly added item) items from the queue and push it back to the queue, this way can make sure the newly pushed item will stay in the front of the queue.

1. Implementation and testing of your solution.

In LC225.java file.

1. Time and Space complexity of your algorithm.

* Space complexity: O (n) – n is the total number of items
* Time complexity:
  + Pop: O (1)
  + Top: O (1)
  + Empty: O (1)
  + Push: O (n) – n is the total number of items

Problem: <https://leetcode.com/problems/reverse-linked-list/>

1. Explanation of the problem with examples (minimum three examples).

The problem asks us to reverse the linked list. Here are the examples:

* 1 => 2 => 3 will be reverse to 3 => 2 => 1
* 1 will be reverse to 1
* Null head will return null

1. Explanation of possible solution(s):

Because this is a singly linked list which means one node only has 1 connection with another node or none (none when it is the last node). The strategy here is to iterate the node one by one, break the connection with the next node and connect to the previous node, but what is the initial value of the first previous node? After observing the examples, we can see that the beginning of the linked list after reverse always connects to none (because eventually the first node will be the last node, and last node will connect to none), so the initialized value of previous node can always be none.

1. Implementation and testing of your solution.
2. Time and Space complexity of your algorithm.

* Space complexity: O (1)
* Time complexity: O (n) – n is the total number of nodes

Problem: <https://leetcode.com/problems/valid-anagram/>

1. Explanation of the problem with examples (minimum three examples).

The problem asks us to check whether the given 2 strings are anagram or not. An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, using all the original letters exactly once.

* “anagram” and “nagaram” are True.
* “rat” and “car” are False.
* “b” and “c” are False.

1. Explanation of possible solution(s)

There 2 approaches we can use:

* Iterate through the first string, each character will be added to hash table, the value of each key will store the number of appearances of that character. Then continue to iterate the second string, now check whether the character exists in the table, if not then return False, else deduce the value => if the value is lesser than 0 => return false. In the end, we will loop through the table, if any keys is greater than 0 then return False else return True.
* Because the anagram requires the other string to use characters from the original string only, which means that they have to be at the same length, also because character can be sorted so if we sorted both string’s characters, they both have to be equal.

1. Implementation and testing of your solution.
2. Time and Space complexity of your algorithm.

* Space complexity: O (n)
* Time complexity: O (n log n) – n is the total number of characters. Because I use Arrays.sort which is a Java built-in, the algorithm is Timsort, it will be O (n log n) on average case and O (n^2) on worst case.