Question 1:

A) A single linked list L containing characters. Write an algorithm as efficient as possible, in pseudo-code, that accepts a pointer to the head of the list, and checks if the letters in the list comprise a palindrome. Your algorithm must use a stack.

```
function isPalindrome(head):
   if head is NULL:
      return true

   stack = new Stack()
   slow = head
   fast = head

   // Push first half elements onto stack
//condition1
   while fast is not NULL and fast.next is not NULL:
      stack.push(slow.data)
      slow = slow.next
      fast = fast.next.next

   // If the list has an odd number of elements, skip the middle element
   if fast is not NULL:
      slow = slow.next

   // Compare second half with stack
   //condition2
   while slow is not NULL:
      top = stack.pop()
      if top != slow.data:
         return false
      slow = slow.next

   return true
```

B) What is the run time complexity of your algorithm? Explain.

First while: O(n/2) as it traverses first half of elements and as in big O we ignore constants so simps to O(n)

Second while: O(n/2) as it traverses second half of elements and as in big O we ignore constants so simps to O(n)

Third (within loops): All are just matter fact operations is O(1)

{Space complexity: The stack stores data from the first half of the linked list. In the worst case, this means storing (n/2) elements. Thus, the space complexity for the stack is O(n/2), which simplifies to O(n) as stated above}

SO: Full Run time is O(n)

C) Prove that the run time complexity of your algorithm is the most efficient for the given problem

Since we need to compare every element from the first half with the respective element in the second half the best case scenario is big O of n

Question 2

Given: Two single linked lists each L1 and L2. L1 list contains n elements and L2 contains m elements and the key in each element is a natural number. The list L1 contains the elements a1,a2,…,an And the list L2 contains the elements b1,b2,…,bn We want to build a new list L3 with n elements c1,c2,…,cn where the content(info, key) of each element is:

$$k = 1,2,3,\ldots,n \quad , c_k = \sum_{i=1}^{k} a_i b_{k-i+1}$$

a) Write a function that builds the list L3 given that the two lists L1 and L2 are double-linked lists. What is the run time complexity of the function that you wrote? O(n^2) because each k from 1 to n, it performs a nested loop up to k iterations, resulting in time complexity quadratic

```
Node:
  int key
  Node* next
  Node* prev
buildL3(L1, L2, n):
  L3 = null
  L3_tail = null
  For k from 1 to n:
    sum = 0
    temp1 = L1
    temp2 = L2
     // Compute the sum for c_k
    For i from 1 to k:
      sum += temp1.key * temp2.key
      temp1 = temp1.next
      temp2 = temp2.prev
    // Create a new node with the computed sum
    newNode = new Node(sum)
    // Append the new node to L3
    If L3 is null:
      L3 = newNode
      L3_tail = newNode
    Else:
      L3_tail.next = newNode
      newNode.prev = L3_tail
      L3_tail = newNode
    Return L3
```

b) Write a function that builds the list L3 given that the two lists L1 and L2 are singly linked lists. You can not reverse the direction of the lists but you must build L3 directly from the original lists, L1 and L2. What is the run time complexity of the function that you wrote? **O(n^2)**

```
buildL3(L1, L2, n):
    if n <= 0:
        return null

    // Initialize dummy head for L3
    L3_head = new ListNode(0)
    L3_current = L3_head
   // Create arrays to store values of L1 and L2
    a = new array[n + 1]
    b = new array[n + 1]

    // Populate array a with values from L1
    current = L1
    for i from 1 to n:
        a[i] = current.val
        current = current.next

    // Populate array b with values from L2
    current = L2
    for i from 1 to n:
        b[i] = current.val
        current = current.next

    // Calculate values for L3
    for k from 1 to n:
        sum = 0
        for i from 1 to k:
            sum += a[i] * b[k - i + 1]
        // Add the calculated value to L3
        L3_current.next = new ListNode(sum)
        L3_current = L3_current.next

    return L3_head.next
```

Question 3

The following function, f(L,x) accepts a single linked-list containing whole numbers and an additional whole number, x.

```
f( L, x)
if ( head(L) == null )
return false
 if (key(head(L) == x) return true
p = head (L)
while ( next(p) != null)
{ if (key (next (p) ) == x)
{ q = next ( next (p) )
next ( next (p) ) = head
head = next (p)
next (p) = q return true
}
else p = next(p)
}
return false
```

1,2,3,2
Explain what f does. Note: Do not explain how, but what it do.

Answer: It finds the X and moves it to the front of the list and moves everything else down one and delete where it previously was

Question 4

a. Implement a stack efficiently using 2 queues. Compute the run time of the stack basic functions that you wrote.

```
class Stack:
  queue1 = new Queue()
  queue2 = new Queue()
function Push(x):
    queue2.enqueue(x) //o(1)
    while not queue1.isEmpty(): //O(n) bec of n elements in que
      queue2.enqueue(queue1.dequeue())
    swap(queue1, queue2) //O(1)
function Pop():
    if queue1.isEmpty(): //O(1) no n just matter of fact
      return "Stack empty"
    return queue1.dequeue()
function Top():
    if queue1.isEmpty(): //O(1) no n just matter of fact
      return "Stack empty"
    return queue1.front()
 function IsEmpty(): //O(1) no n just matter of fact
    return queue1.isEmpty()
```

Computing: TTC: push = O(n), POP = O(1), TOP = O(1), isEmpty = O(1)

Answer: O(mn) because it will rely on efficacy of push which is O(n) and how many times it is called

b. Can you implement a stack using only 1 queue with O(1) additional memory? If yes, implement the basic functions and determine their run-time complexity, and if not, explain why.

```
class Stack:
    queue = new Queue()
    function Push(x):
        queue.enqueue(x)
        size = queue.size()
        while size > 1:
            queue.enqueue(queue.dequeue())
            size = size - 1
    function Pop():
        if queue.isEmpty():
            return "Stack is empty"
        return queue.dequeue()
    function Top():
        if queue.isEmpty():
            return "Stack is empty"
        top_element = queue.tail()
        // queue.enqueue(queue.dequeue())
        return top_element
    function IsEmpty():
        return queue.isEmpty()
```

TTC: O(n^2)

Saves memory with 1 queue but has inefficiency with using push too much

Question 5

Write an algorithm that reads n numbers from the input, and puts them into a stack so that they are sorted every moment inside the stack. One additional auxiliary stack may be used, but without additional variables. What is the run time complexity of the algorithm you wrote? Note: You may assume that the input is finished when the value (-1) is inputted.

```
SortStackedNumbers(n):
Input: n numbers from the input, to terminate enter -1
Output: Sorted numbers in the main stack
main_stack = Stack()
aux_stack = Stack()

for i from 1 to n:
    push input number onto aux_stack

    while main_stack is not empty and input number < top element of main_stack:
        move top element of main_stack to aux_stack
    push input number to main_stack

    while aux_stack is not empty:
        move top element of aux_stack to main_stack

while main_stack is not empty:
    output top element of main_stack
    pop top element of main_stack
```

TTC: $O(n^2)$

Question 6

Explain how two stacks can be implemented by one array A[1..n] so that overflow occurs only when the number of elements in both stacks together reaches n.

Divide the array into two equal parts: one for each stack.
Each stack grows towards the center of the array. Left$\rightarrow$ N$\leftarrow$ Right
Overflow occurs only when both stacks together have filled the entire array, ensuring that overflow is only reached when both stacks are full.
Similar to Merge sort