- This lab will cover <u>Linked Lists</u>.
- It is assumed that you have reviewed chapter 6 & 7 of the textbook. You may want to refer to the text and your lecture notes during the lab as you solve the problems.
- When approaching the problems, <u>think before you code</u>. Doing so is good practice and can help you lay out possible solutions.
- <u>Think of any possible test cases</u> that can potentially cause your solution to fail!
- You must stay for the duration of the lab. If you finish early, you may help other students. If you don't finish by the end of the lab, we recommend you complete it on your own time. <u>Ideally you should not spend more time than suggested for each problem.</u>
- Your TAs are available to answer questions in lab, during office hours, and on Piazza.

Vitamins (maximum 20 minutes)

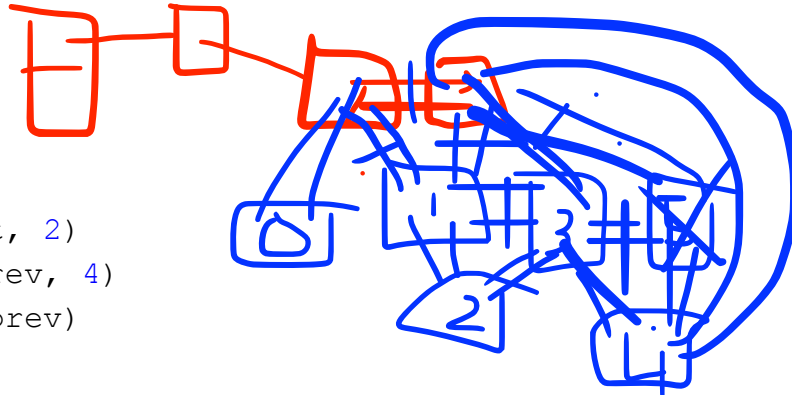1. Draw the memory image of the linked list object as the following code executes: (5 minutes)

```
from DoublyLinkedList import DoublyLinkedList


dll = DoublyLinkedList()
dll.add_first(1)
dll.add_last(3)
dll.add_last(5)
dll.add_after(dll.header.next, 2)
dll.add_before(dll.trailer.prev, 4)
dll.delete_node(dll.trailer.prev)
dll.add_first(0)

print(dll)
```

What is the output of the code?

0,1,2,3,4

2. During lecture you learned about the different methods of a doubly linked list.

   Provide the following worst-case runtime for those methods:

   a. `def __len__(self):`

      1

   b. `def is_empty(self):`

      1

   c. `def add_after(self, node, data):`

      1

   d. `def add_first(self, data):`

   e. `def add_last(self, data):`

   f. `def add_before(self, node, data):`

   g. `def delete_node(self, node):`

   h. `def delete_first(self):`

   i. `def delete_last(self):`

3. Trace the following function. What is the output of the following code using the doubly
    linked list from question 3? Give mystery an appropriate name.

    remove_second_to_last

```
#dll = Doubly Linked List
def mystery(dll):

    if len(dll) >= 2:
        node = dll.trailer.prev.prev    takes the 2nd to end
        node.prev.next = node.next    sets next of 3rd to end to last
        node.next.prev = node.prev

        node.next = None
        node.prev = None
        return node

    else:
        raise Exception("dll must have length of 2 of
        greater")

print(mystery(dll))
```

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code.

> Download the **DoublyLinkedList.py** file attached under content on Brightspace

1. In class, we defined the `stack` ADT using a dynamic array as the underlying data structure. Because of the resizing, the ArrayStack run-time for its operations is not exactly in Θ(1). Instead, the cost is Θ(1) amortized. (10 minutes)

   Define the stack ADT that guarantees each method to always run in Θ(1) **worst case**.

   ```python
   class LinkedStack:

       def __init__(self):
           ...

       def __len__(self):
           ''' Returns the number of elements in the stack. '''


       def is_empty(self):
           ''' Returns true if the stack is empty,false otherwise.
           '''


       def push(self, e):
           ''' Adds an element, e, to the top of the stack. '''


       def top(self):
           ''' Returns the element at the top of the stack.
               An exception is raised if the stack is empty. '''


       def pop(self):
           ''' Removes and returns the element at the top of the
           stack.
               An exception is raised if the stack is empty. '''
   ```

2. Implement the following method for the `DoublyLinkedList` class. The get item operator takes in an index, i, and returns the value at the ith node of the Doubly Linked List.

   You only need to support non-negative indices. Your solution should try to optimize the get item method. This means that you should decide whether to iterate from either the header or trailer based on whichever is closer to i. Index should start at 0. (30 minutes)

   ```
   def __getitem__(self, i):
       '''Return the value at the ith node. If i is out of range,
       an IndexError is raised'''
   ```

   For example, if your Doubly Linked List looks like this:
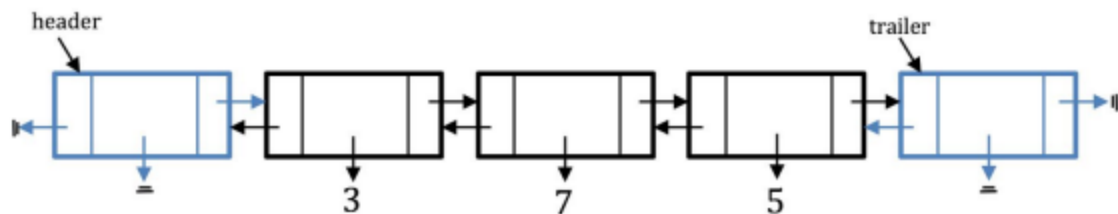

   ```
   print(dll[0]) # 3 (should iterate from the header)

   print(dll[1]) # 7 (either way works)

   print(dll[2]) # 5 (should iterate from the trailer)

   print(dll[3]) # IndexError
   ```

   What is the worst-case run-time of the get item operator? Can we do better?

   theta(n) if it needs to go through half the list, this is the best we can do if we only have access to the ends

3. In <u>homework 5</u>, you were asked to implement a `MidStack` ADT using an ArrayStack and an ArrayDeque. This time, you will create the MidStack using a **Doubly Linked List** with Θ(1) extra space. All methods of this MidStack should have a Θ(1) run-time.

   The middle is defined as the $(n + 1)//2\,th$ element, where n is the number of elements in the stack.

   <u>**Hint:**</u> To access the middle of the stack in constant time, you may want to define an additional data member to reference the middle of the Doubly Linked List. (40 minutes)

```python
class MidStack:

    def __init__(self):
        self.data = DoublyLinkedList()
        ...

    def __len__(self):
    ''' Returns the number of elements in the stack. '''


    def is_empty(self):
    ''' Returns true if stack is empty and false otherwise.
    '''


    def push(self, e):
    ''' Adds an element, e, to the top of the stack. '''


    def top(self):
    ''' Returns the element at the top of the stack.
        An exception is raised if the stack is empty. '''


    def pop(self):
    ''' Removes and returns the element at the top of the
    stack.
        An exception is raised if the stack is empty. '''


    def mid_push(self, e):
    ''' Adds an element, e, to the middle of the stack.
```
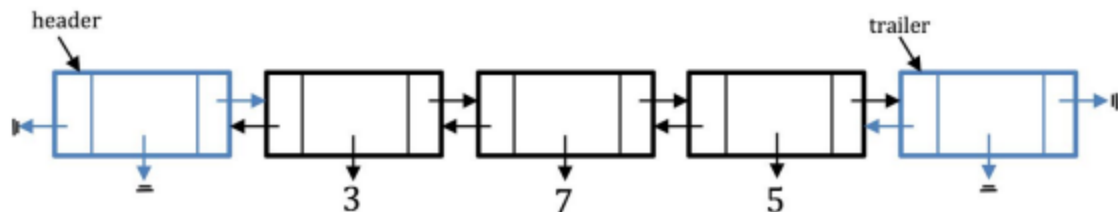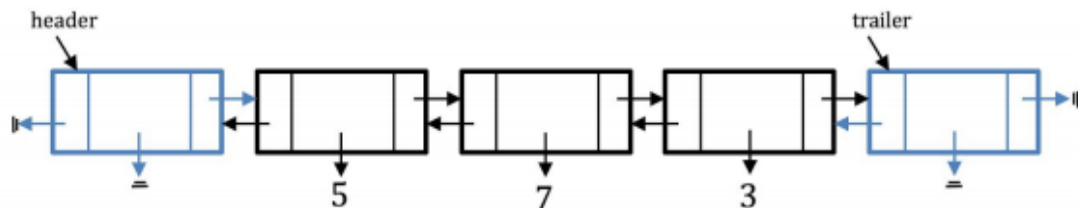
```
            An exception is raised if the stack is empty. '''
```

4. Implement a method to reverse a doubly linked list. This method should be non-recursive and done **in-place** (do not return a new linked list).

For example if your list looks like:

After calling the method on it, it will look like:

You will implement the reversal in two ways:

    a. First implement a function which reverses the data in the list, but does not move any nodes. Instead, change the value at each node so that the order is reversed.

```python
def reverse_dll_by_data(dll):
''' Reverses the linked list '''
```

    b. Next, implement a function which reverses the order of the nodes in the list. That is, you should move the nodes objects around, without changing their data value and without creating any new node objects.

```python
def reverse_dll_by_node(dll):
''' Reverses the linked list '''
```

OPTIONAL

5. Implement the `SinglyLinkedList`. The SinglyLinkedList differs from the DoublyLinkedList in that there is only a header and no trailer. In addition, each node only references the node after it. The last node in the linked list will reference its next Node as `None` (this would've been self.trailer for a DoublyLinkedList).

> **You may add additional data members that are O(1) extra space.** Analyze the run-time of each method after completing the implementation. Is it possible to make add_last and or delete_last work in O(1) constant run-time? (40 minutes)

```python
class SinglyLinkedList:
    class Node:
        def __init__(self, data=None, next=None):
            self.data = data
            self.next = next

        def disconnect(self):
            self.data = None
            self.next = None


    def __init__(self):
        self.header = SinglyLinkedList.Node()
        self.size = 0

    def __len__(self):
        return self.size

    def is_empty(self):
        return (len(self) == 0)

    def add_after(self, node, val):
        ''' Creates a new node containing val as its data and adds
        it after an existing node in the SinglyLinkedList'''

    def add_before(self, node, val):
        ''' Creates a new node containing val as its data and adds
        it before an existing node in the SinglyLinkedList'''

    def add_first(self, val):
        ''' Creates a new node containing val as its data and adds
        it to the front of the SinglyLinkedList'''
```

```python
    def add_last(self, val):
      ''' Creates a new node containing val as its data and adds
      it to the back of the SinglyLinkedList'''


    def delete_node(self, node):
      ''' Removes an existing node from the SinglyLinkedList and
      returns its value'''

    def delete_first(self):
      ''' Removes an existing node from the front of the
      SinglyLinkedList and returns its value'''

    def delete_last(self):
      ''' Removes an existing node from the back of the
      SinglyLinkedList and returns its value'''

    def __iter__(self):
        cursor = self.header.next
        while(cursor is not None):
            yield cursor.data
            cursor = cursor.next

    def __repr__(self):
        return "[" + " -> ".join([str(elem) for elem in self])
+ "]"
```