**Objective:** To understand FIFO (First-In-First-Out) queue implementations in Python including being able to determine the big-oh of each operation.

**To start the lab:** Download and unzip the lab3.zip file from eLearning

# **Part A:** The textbook’s QueueText implementation in lab3/queue\_text.py uses a Python list

# 

a) Complete the big-oh notation for the above QueueText implementation: ("n" is the # items)

|  | \_\_init\_\_ | enqueue(item) | dequeue( ) | peek( ) | size( ) | isEmpty( ) | \_\_str\_\_ |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Big-oh | O(1) | O(n) | O(1) | O(1) | O(1) | O(1) | O(n) |

b) Explain your big-oh answer for enqueue(item).

When adding a new item, we put it at the back of the line, or at index 0. Because of this, we need to shift all n items up by one index. Because of this, O(n) is suitable.

c) Explain your big-oh answer for dequeue( )

When we remove an item, we are removing it from the highest index. Because of this, no others need to move, only the one item needs to move. Therefore, O(1) fits here.

d) Run the timeQueue.py file which times 100,000 enqueues followed by 100,000 dequeues.

Time for 100,000 enqueues: Time for 100,000 dequeues:

1.582 seconds 0.006 seconds

e) Why do the enqueues take so much more time?

They are shifting all the items every time, hence the O(n), while the dequeues are only removing from the end, which is constant (O(1))

**After answering the above questions, raise you hand and explain your answers.**

**Part B:**

**a) Complete the QueueAlt implementation in lab3/queue\_alt.py uses a Python list**

Alternate queue using Python list with front item at index 0.

b) Complete the big-oh notation for the above QueueAlt implementation: ("n" is the # items)

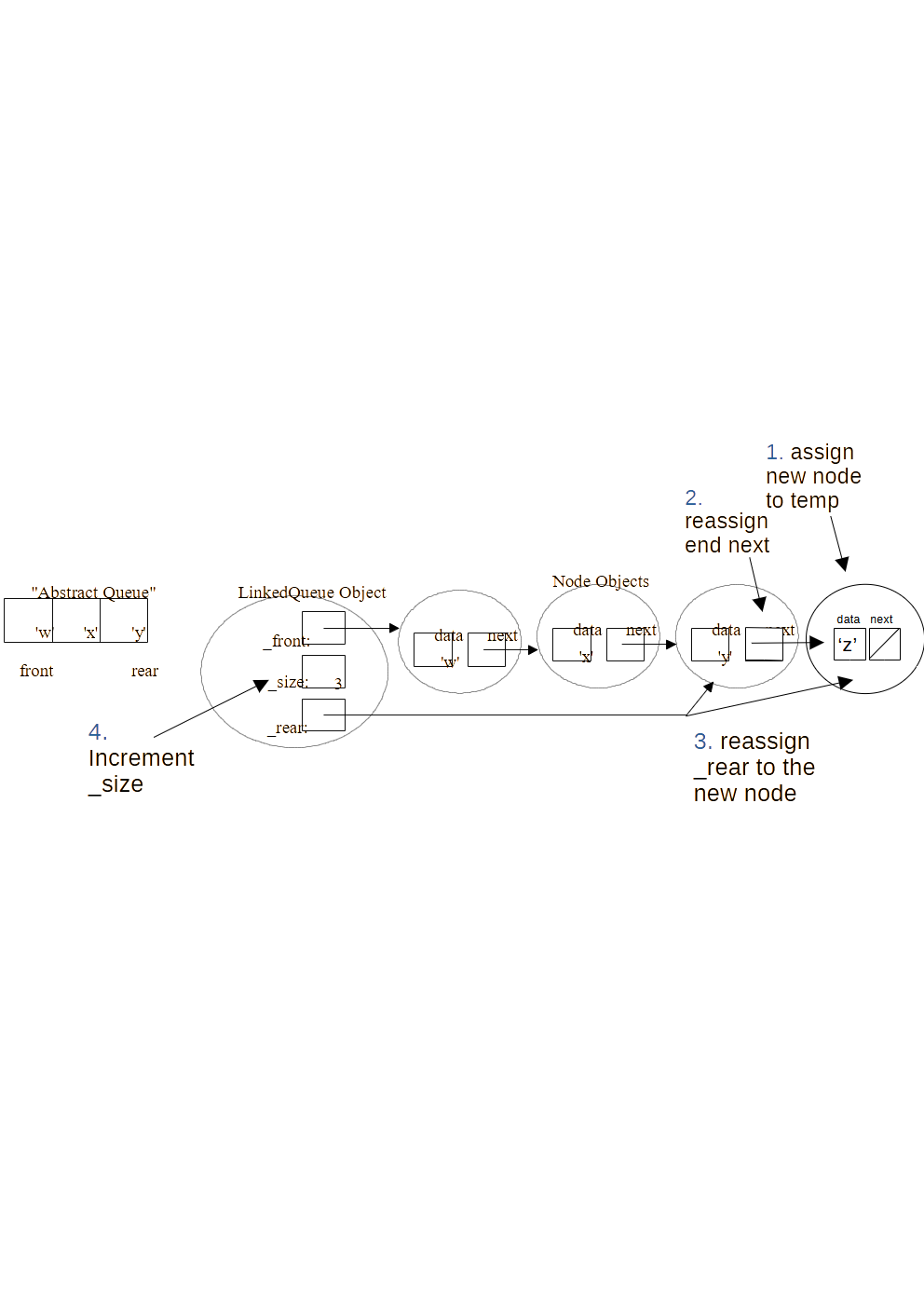
|  | \_\_init\_\_ | enqueue(item) | dequeue( ) | peek( ) | size( ) | isEmpty( ) | \_\_str\_\_ |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Big-oh | O(1) | O(1) | O(n) | O(1) | O(1) | O(1) | O(n) |

c) Run the timeQueueAlt.py file which times 100,000 enqueues followed by 100,000 dequeues.

Time for 100,000 enqueues: Time for 100,000 dequeues:

0.011 seconds 0.645 seconds

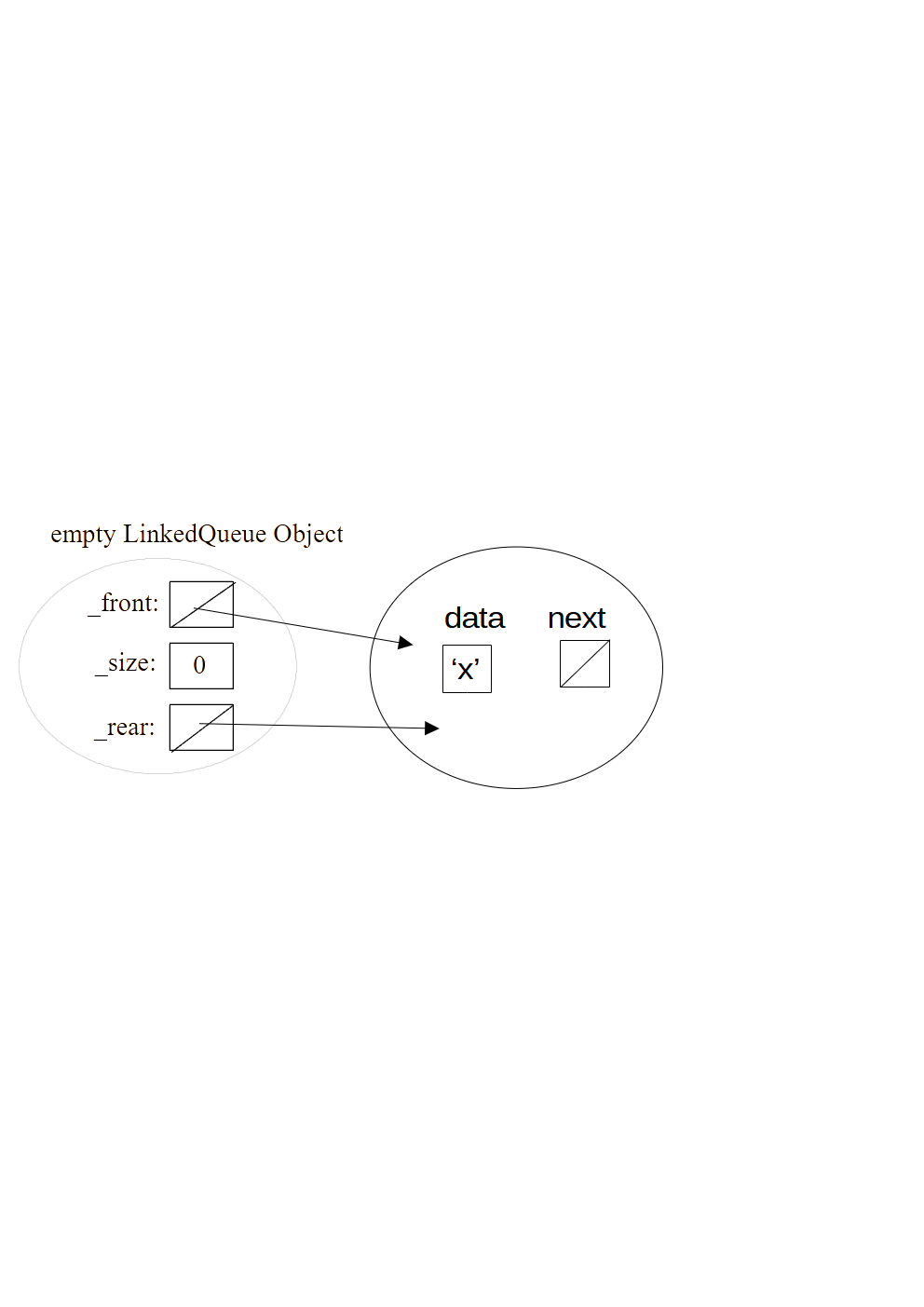
# **Part C:** Consider the LinkedQueue implementation in lab3/linked\_queue.py which uses a linked structure that looks like:

a) Modify the above picture and number the steps for the enqueue method’s “normal” case (non-empty queue)

b) Write the “normal” case code for the enqueue method below.

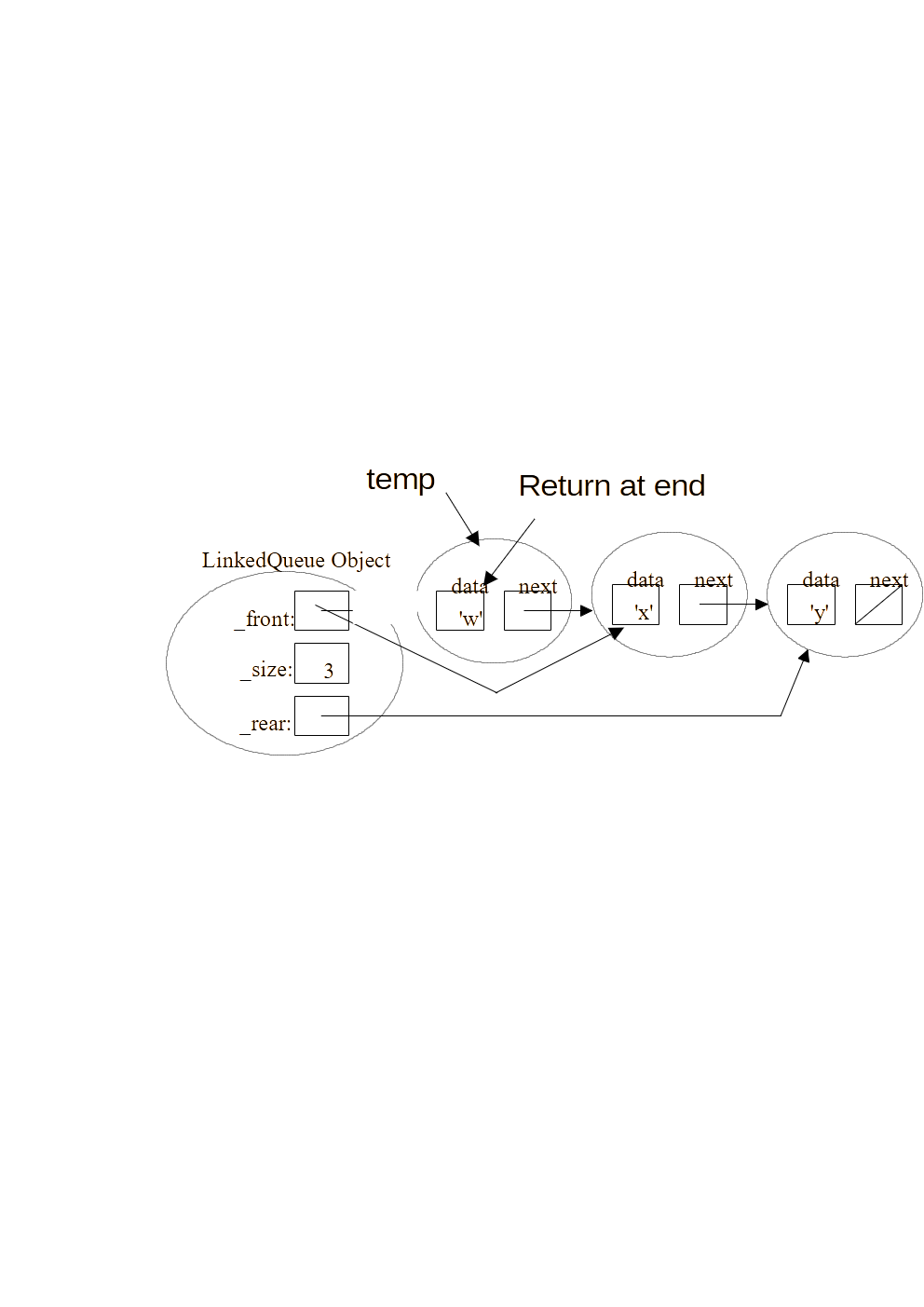
1. Make new item called ‘temp’ to store the new item
2. Reassign the ‘next’ of \_rear to the temp item
3. reassign \_rear to the temp item
4. Increment \_size by one

c) Starting with the empty queue below, draw the resulting picture after your “normal” case code executes.



d) **Complete the enqueue method code for the “normal”and special case(s) in the lab3/linked\_queue.py file**

Consider dequeuing from the below “normal” case picture (i.e., it should remove and return ‘w’):

e) Modify the above picture and number the steps for the dequeue method’s “normal” case (non-empty queue)

f) Write the “normal” case code for the dequeue method below.

1. Create temp as a copy of \_front
2. set \_front to \_front.next
3. decrement \_size
4. Return temp.getValue()

g) What “special case(s)” does the dequeue method code need to handle?

If the queue is already empty there’s nothing to dequeue, raise an attribute error.

h) **Complete the dequeue method code for the “normal” case and special case(s) in the lab3/linked\_queue.py file.**

i) **Complete the peek method code for the “normal” case and special case(s) in the lab3/linked\_queue.py file.**

j) Complete the big-oh notation for the LinkedQueue methods: ("n" is the # items)

|  | \_\_init\_\_ | enqueue(item) | dequeue( ) | peek( ) | size( ) | isEmpty( ) | \_\_str\_\_ |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Big-oh | O(1) | O(1) | O(1) | O(1) | O(1) | O(1) | O(n) |

k) Run the timeLinkedQueue.py file which times 100,000 enqueues followed by 100,000 dequeues.

Time for 100,000 enqueues: Time for 100,000 dequeues:

**After you have working code, zip the lab3 folder and submit it on eLearning. (You should save a copy too.)**

**If you have extra time, this would be a good chance to work on Homework #2!**