FIT1008 – Intro to Computer Science
Solutions for Tutorial 8

Semester 1, 2018

Exercise 1

One possible answer is:

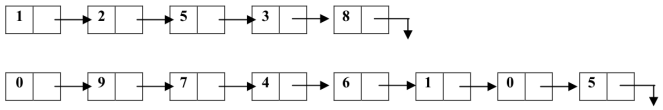
```
def delete_negative(self):
    # skip all negative elements at the head
    while self.head != None and self.head.item < 0:
        self.head = self.head.next

    if self.head != None:
        # skip the remaining negatives if any
        previous = self.head
        current = self.head.next
        while current != None:
            if current.item < 0:
                previous.next = current.next     # either skip it
            else:
                previous = current               # or move previous along

            current = current.next               # anyway, move current along
```

Common mistakes:

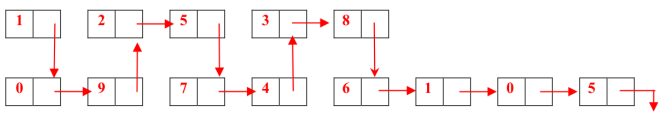
- forgetting to set the head appropriately;
- moving `previous` along when an element has been eliminated;
- checking for `current != None` and then use `current.next.item`;
- forgetting to check whether the list is empty (`head == None`).

Exercise 2

We were considering a call to `mystery(l1, l2)` where the `l1` and `l2` are, respectively:



This is the result after the function has been executed.



What the function does is interleave the two lists, taking an element from each of them, while both lists have elements. When the function has consumed all the elements from the shorter list, it appends all the elements left in the longer one:
The function's best and worst Big O time complexity are the same, since it will traverse all elements in the two lists until one of them is finished. Therefore, there is no property of any two large N1 and N2 inputs that could cause a change in complexity. Since the amount of work at each step is constant (comparisons and copies), and the method must go through every element in each list up to the length of the shorter list, the complexity is O(min(N1,N2)).

Exercise 3

```
class  CircularQueueIter:

      def __init__(self, queue):
          self.array = queue.array
          self.size_of_q = queue.count
          self.count = 0
          self.current = queue.front

     def __iter__(self):
          return self

     def __next__(self):
          if self.count >= self.size_of_q:
             raise StopIteration

          item = self.array[self.current]
          self.current = (self.current + 1) % len(self.array)
          self.count += 1
          return item
```

Exercise 4

(a)

```
def appears(a_list, item, k):
       if k > len(a_list):
           return False

       kfound = 0
       for elem in a_list:
            if elem == item:
               kfound += 1
            if kfound == k:
               return True

```

```
12          return False
```

(b) Best case: O(k) where k< N and the first k elements are the item we are looking for, since it will finish as soon as we find the first k. Worst case: O(N) where the item does not appear k times, since we will traverse all elements trying to find item k times.