Jeffrey Wan Module 4 Homework

Assignment 4 – Queues and Lists Write pseudo-code not Java for problems requiring code. You are responsible for the appropriate level of detail.

1. **Develop an ADT specification for a priority queue. A priority queue is like a FIFO queue except that items are ordered by some priority setting instead of time. In fact, you may think of a FIFO queue as a priority queue in which the time stamp is used to define priority.**

Priority queue is that the highest priority item in the queue is the first that comes out. When you insert, you don't know where it's going to go since you have to compare priorities for the new items with the items inside the queue already. In a priority queue, when an item is added to the queue, it is assigned a priority to indicate the relative priority of the item versus other items.

Methods:

Constructor:
inputs: None
precondition: None
process: Creates a new empty priority queue.
postcondition: None
output: The new empty priorty queue.

isEmpty:
inputs: None
precondition: a queue needs to have been created
process: Checks length of items of queue.
postcondition: None
output: An integer value representing the size of the queue.

enqueue:
inputs: an item and a priority value
precondition: integer needs to be positive or 0. throws error if priority is negative.
process: inserts item into the queue and places the item in front of all elements with lesser priority but behind all others.
postcondition: None
output: None

dequeue:
inputs: None.
precondition: queue needs to have items in it or else throws exception.
process: removes an item from the front of the queue which has the highest priority.
postcondition: None
output: Returns the removed item with the highest priority.

Insert(item)
inputs: None
precondition:
process: Insert item into the queue in priority order.
postcondition:
output: None

2. **Write an algorithm to reverse a singly linked list, so that the last element become the first and so on. Do NOT use Deletion - rearrange the pointers.**

```
# Initialize three pointers: prev = NULL, curr = Head and next = NULL.
prev = null
curr = list.head
next = null

# Iterate through the linked list. In loop, do following.
      while curr.next != null:
         // Before changing next of current, store next node
         next = curr.next

         // This is where actual reversing happens. On first pass, the head points to null so this
makes sense as it should now be the tail.
               curr.next = prev
               // Move prev and curr one step forward
               prev = curr
               curr = next
```

3. **What is the average number of nodes accessed in search for a particular element in an unordered list? In an ordered list? In an unordered array? In an ordered array? Note that a list could be implemented as a linked structure or within an array.**

When you search for a particular element x in an array of size n, that element may be located at the first or last position. When we search, we check each element in the array and compare it with x, and so when we reach kth element of the array we have already done k comparisons.

In order to average it you sum the total number of comparisons $1+2+\cdots+n = (n+1)n/2$
then you can divide it by n (size of the array) resulting in $n+1/2$.

unordered list: n/2
ordered list:n/2
unordered array: n/2
ordered array: (log n)

For ordered arrays, performing a binary search algorithm would come to log(n).

Question for grader: What is the difference between an ordered list and an ordered array? I guess I'm slightly confused by the subtle difference between a list and an array. A list can be implemented using an array, that I know… but what else? Why can't binary search be used on an ordered list?

4. **Write a routine to interchange the mth and nth elements of a singly-linked list. You may assume that the ranks m and n are passed in as parameters. Allow for all the ways that m and n can occur. You must rearrange the pointers, not simply swap the contents**

Edge cases:
1) mth and nth node may or may not be adjacent.
2) Either mth node or nth node may be the head.
3) Either mth or nth node may be the tail.
4) mth node and/or nth node may not be present in linked list.
5) they might be equal (so we can just write a guard clause)

```
def swapNodes(self, m)
        # Nothing to do if m and n are the same
        if m == n:
                return

        if m > list.length or n > list.length:
                return


        m_node = find(m)

        if m == 1:
                left_of_m = null
        else:
                left_of_m = find(m - 1)

        if m == list.length()
                right_of_m = null
        else:
                right_of_m = find(m+1)


        n_node = find(n)

        if n == 1
                left_of_n = null
        else
                left_of_n = find(n-1)

        if n == list.length()
                right_of_n = null
        else:
                right_of_n = find(n+1)


       # swap nodes
       if left_of_m !=  null
                left_of_m.next = n_node
       else:
```

```
            list.head = n_node

# if right_of_m is null or another node, this line works
n_node.next = right_of_m

# if n_node is the tail, which was set right above, set list tail to n_node
if n_node.next == null
    list.tail = n_node

if left_of_n != null
            left_of_n.next = m_node
else:
            list.head = m_node

# if right_of_n is null or another node, this line works
m_node.next = right_of_n


# if m_node is the tail, which was set right above, set list tail to m_node
if m_node.next == null
    list.tail = m_node
```