Adam Rich
EN.605.202.87.SP18 Data Structures
Homework Assignment 4
February 27, 2018

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.

```
ADT PriorityQueue
        Data
                An empty list of items (values and and associated prioritites) with
                references to the first/front/head and last/end/tail items.  Items are
                sorted by descending priority.  Or in other words, "first" has highest
                priority, "last" has lowest.

        Methods
                isEmpty
                        Input: None
                        Precondition: None
                        Process: Check if the queue contains items
                        Postcondition: None
                        Output: Return 1 or true if queue is empty, 0 or false otherwise
                delete
                        Input: None
                        Precondition: Queue is not empty
                        Process: Remove item from front of queue, i.e. item with highest
                        priority
                        Postcondition: Queue contains one less item
                        Output: Return the deleted value
                insert
                        Input: A value and assiociated priority for storage in the queue
                        Precondition: None
                        Process: Compare the priority of the inserted value with those
                        already in the queue and position the new value so that the queue
                        is still sorted by descending priority.
                        Postcondition: The queue contains one additional data item, all
                        items are in the right order
                        Output: None
                delete
                        Input: None
                        Precondition: Queue is not empty
                        Process: Look at item with highest priority, get its value
                        Postcondition: Queue has not changed
                        Output: Return the value of the highest priority item
end ADT PriorityQueue
```

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.

```
// Following convention of zyBook that first element in list is L.head
method reverse_list(singly-linked-list L)

  if L has 0 or 1 elements then return L

  // L has 2 or more elements
  init temp as empty node
  init after as empty node

  // The lectures do not have L.tail, but zyBooks does
  // If implemented with a tail then
  L.tail = L.head

  temp = L.head
  L.head = null

  loop while temp is not null

    after = temp.next
    temp.next = L.head
    L.head = temp
    temp = after

  end-loop

  return L
end-method
```

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.

Assume that "unordered" means "not sorted" (becuase lists were defined as "ordered" in the lectures). Assume that "ordered" means "sorted". Therefore, we are searching by "value" not by position.

An "unordered list":
If the list does not allow duplicate values or if it does and we only need the first match this will require us to access N/2 nodes on average (sometimes the value will be at the front of the list, sometimes at the end and we expect that they will be uniformly distributed). If a list allows duplicates and we want to find all of them then we will access all N elements on every search.

An "ordered list":
We cannot do a binary search in an ordered list because we don't know anything about the implementation (it is an ADT afterall), so we still have to walk the links until we find something. So, knowing nothing about the implementation just the ADT definition this will be the same as the "unordered list". The list ADT does not specify that it has random access so we cannot assume that.

An "unordered array":
This will be the same as an unordered list, N/2.

An "ordered array":
With this one we will be able to use a variant on the binary search algorithm (depending on which implementation alternative we use: standard, bit-masked, or marked). If we are using the standard implemenation we will access log(size of list) on average. If we are using one of the other implementations where free space is not contiguous we will have to access log(last known used position in array) elements on average.

4. Write a routine to interchange the *m*th and *n*th elements of a singly-linked list. You must rearrange the pointers, not simply swap the contents.

```
// Following convention of zyBook that first element in list is L.head
// Per lecture, assuming that ptrTo is available

method eminem_swap(singly-linked-list L, int m, int n)
  make sure that m and n are valid positions in L
    if not throw error

  // trivial case
  if m == n then just return L

  // get space to keep track of everything
  init node_m as empty node
  init node_n as empty node
  init after_m as empty node
  init after_n as empty node
  init before_m as empty node
  init before_n as empty node

  // Point to m, n and neighbors
  if m == 1 then before_m = null
    else before_m = L.ptrTo(m - 1)
  node_m = L.ptrTo(m)
  after_m = node_m.next

  if n == 1 then before_n = null
    else before_n = L.ptrTo(n - 1)
  node_n = L.ptrTo(n)
  after_n = node_n.next

  // Repoint
  if before_m == null L.head = node_n
    else before_m.next = node_n
  node_n.next = after_m

  if before_n == null L.head = node_m
    else before_n.next = node_m
  node_m.next = after_n

end-method
```