**Assignment 4 – Queues and Lists**

*Write pseudo-code not Java for problems requiring code. You are responsible for the appropriate level of detail.*

## 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 values with a reference to the first (front) and last (rear) items.

Methods

PQEmpty

**Input**: None

**Precondition**: None

**Process**: Check if the PQueue contains any data items.

**Postcondition**: None

**Output**: Return 1 or True if PQueue is empty, or 0/false otherwise.

PQDelete

**Input**: Queue and its head node

**Precondition**: Queue contains meaningful data values

**Process**: Remove an item from the front of the queue

**Postcondition**: The queue contains one less data item, and head is either null or points to the next node.

**Output**: Return the deleted value.

PQInsert

**Input**: A data item to be stored in the queue, along with its priority

**Precondition**: None

**Process**: Place an item based on its priority. Will be placed in front of items with lower priority, but behind earlier items of similar priority. If priority is 0, will be placed at end of queue. If priority is negative, will throw an error.

**Postcondition**: Queue contains one additional data item, and priority order is maintained

**Output**: None

PQPeek

**Input**: None

**Precondition**: Queue contains meaningful data values

**Process**: Retrieve the value of the data item at the front of the quee

**Postcondition**: The queue is unchanged

**Output**: None

1. **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.**

//Note, this first part takes place in the linked list class

Class LinkedList{

…

Class Reverse(Node current, Node previous){

If(current == Null){

Return error “Can’t use empty list”;

}

If(current.next == Null){

Head = current;

Current.next = previous;

}

//Create a temp node that saves the current next pointer, and uses it to continue the recursion

Node Temp = current.next;

Current.next = previous;

//Recursion on the former next pointer

Reverse(Temp, current);

Return head;

}

…

}

Class main(){

Node reverseList = originalList.Reverse(head,Null);

}

1. **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.**

Unordered list: Could be in the front or the back, so average cost is n/2.

Ordered list: Unless you have a reference in the list, you will search from the beginning. So cost is same as unordered list, n/2.

Unordered array: Same as unordered list, will have to search from beginning. Cost is n/2.

Ordered array: Search will be performed using binary search, so cost will be lg(n).

1. **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.**

//This would be a method within a linked list class

Class Swap(int m, int n){

Node Current\_M = head;

Node Current\_N = head;

Node Previous\_M = Null;

Node Previous\_N = Null;

If(Current\_M == Null){

Throw Error “empty list”;

}

//Finding nodes at position M and N

For( I = 1;I < m; I++){

Previous\_M = Current\_M;

Current\_M = Current\_M.next;

If(Current\_M == Null){

Throw error;

}

}

For( I = 1;I < N; I++){

Previous\_N = Current\_N;

Current\_N = Current\_N.next;

If(Current\_N == Null){

Throw error;

}

}

//Swap previous next pointers

If (previous\_M !=Null){

Previous\_M.next = Current\_N;

} else {

Head = Current\_N

}

If (previous\_N != Null){

Previous\_N.next = Current\_M;

} else {

Head = Current\_M;

}

//swap current next pointers

Node temp = Current\_M.next;

Current\_M.next = Current\_N.next;

Current\_N.next = temp;