Assignment 4 – Queues and Lists

1.

PriorityQueue(): initialize the queue

Methods

PriorityQueueEmpty()

Input: none

Precondition: none

PostCondition: if the priority queue length == 0 then queue is empty.

Output: return priorityqueueempty true if no values.

PriorityQueueRemove()

Input: None

Precondition: if PriorityQueueEmpty() = false

Process: PriortyQueueRemove item at the head which is of highest priority

Postcondition: Returns at value at the head of the element.

Output: returns the removed value.

PriorityQueueInsert()

Precondition:none

Process: Places value at the rear of the queue which is behind all others that are in queue which makes it of lesser priority.

Postcondition: value inserted at tail end of queue

Output: value inserted

2.

Best way to do it would be as below the while loop will repeat its self until the head of the node is at the end of the single linked list were it wont processed to the next value because it is a null.

Method for reversing node:

Node prev = null;

Node current = node;

Node next = null;

while (current != null)

Then next = current.next

Current.next = prev

Prev=current

Current = next

Then node = prev

3.

In an unordered list since it would probably be a sequential search in this case we would have a average number of nodes be n/2.

With an ordered list we have a relative position to each of the items within that list. The ordered list would start at the beginning of the list and search through each item causing it to be the same average number of searches which is n/2. Even though it is ordered this is a linked list we have to traverse through each element in order to find the one we are searching for unlike an array with random access.

In an unordered array it would the same thing but the values contained within that array are able to be accessed directly. The array would be searched sequentially just do to it being a better time complexity to search an unordered array. So number of searches would be n/2.

With an ordered array we have the elements being sorted so this would go with a binary search were it would start at the middle element and compare with the value we are searching for is less than or greater than that value. If the value in the middle value is greater then the value we are searching for that means the value would most likely be towards the left half of the array which is the first portion of the array. So the average number for the search of this area would be $(\log_2 n) - 1$

4.

```
Initializing a node through class
InterchangingMandN(int M, int N)
If(M == N) // if both the mth and nth elements are equal
{
Return;
}
Node prevM = null
Node currM = head
Node nextM = null
While (currM != null && currM.value != I)
prevM = currM
currM = currM.next

Node prevN = null
Node nextN = null
Node currN = head
While(curr != N && currN.value != N)
PrevN= currN
CurrN = currN.next

If(currM == null || currN == null)
Then return; // returns nothing
Else If(prev N != null)
Then
prevN.next = currM
else
head = currM

If (prevM != null)
Then
prevM.next = currN
else
head = currN

Node swap = currM.next
currM.next = currN.next;
currN.next = swap
```

5.

Initialize the node
object data;
Node previous
Node next
Node currNode
insertLeft(object item)
if the currNode.previous cannot equal null
currNode = currNode.previous
currNode.previous = new Node()
currNode.previous.data = item
currNode.previous.previous = null
currNode.previous.next = currNode
else if the currNode == null
currNode = new Node();
currNode.data = item
currNode.previous = null
currNode.next = null

DeleteRight()
If(currNode.next != null)
CurrNode = currNode.next
currNode.data = item
currNode = currNode.previous
currNode.next.previous = null
currNod.next = null
else if the currNode == null
then return null

6.

InsertRight
If head is null
Head = currNode;
tail = currNode
currNode.previous = currNode
currNode.next = currNode;
capacity = 1
else
currNode.previous = tail
currNode.next = head
tail = currNode
capacity++

deleteLeft
if the head equal to null
return null;
else
data = head.item

head.previous = null
tail.next = head.next
head.next = null
head = tail.next
head.previous = tail