Ron Cox

HW4
605.202.81
Data Structures

1. ADT PriorityQueue

Data: A collection of elements each associated with a priority.
Operations:

- `insert(item, priority)`: Inserts an item with the given priority into the priority queue.
- `extractMax()`: Removes and returns the item with the highest priority.
- `isEmpty()`: Returns true if the priority queue is empty, false otherwise.
- `peekMax()`: Returns the item with the highest priority without removing it.
- `changePriority(item, newPriority)`: Changes the priority of the given item to the new priority.

```
Operation insert(item, priority)
 Input: item - the element to be inserted
   priority - the priority of the item
 Output: none
 Effect: The item is inserted into the priority queue with the specified
priority

Operation extractMax()
 Input: none
 Output: the item with the highest priority
 Effect: The item with the highest priority is removed from the priority
queue

Operation isEmpty()
 Input: none
 Output: boolean - true if the priority queue is empty, false otherwise

Operation peekMax()
 Input: none
 Output: the item with the highest priority
 Effect: The item with the highest priority is returned without being
removed from the priority queue

Operation changePriority(item, newPriority)
```

```
Input: item - the element whose priority needs to be changed
    newPriority - the new priority of the item
Output: none
Effect: The priority of the specified item is updated to the new priority
```

2. Algorithm to Reverse a Singly Linked List

Algorithm ReverseLinkedList(head)

```
Input: head - the head of the singly linked list
Output: the new head of the reversed list

1. Initialize previous to null
2. Initialize current to head
3. While current is not null
   a. next <- current.next
   b. current.next <- previous
   c. previous <- current
   d. current <- next
4. Return previous as the new head of the reversed list
```

3. Average Number of Nodes Accessed in Search

Unordered List (Linked Structure):

- Average case: $\frac{n+1}{2}$
- Justification: On average, the element is located halfway through the list.

Ordered List (Linked Structure):

- Average case: $\frac{n+1}{2}$
- Justification: Similar to an unordered list, on average, the element is located halfway through the list.

Unordered Array:

- Average case: $\frac{n+1}{2}$
- Justification: Similar to an unordered list, on average, the element is located halfway through the array.

Ordered Array:

- Average case: $log_2(n)$
- Justification: Binary search can be used, which halves the search space each time.

4. Routine to Interchange the mth and nth Elements of a Singly-Linked List

Algorithm InterchangeNodes(head, m, n)

```
Input: head - the head of the singly linked list
      m - the position of the first node to be interchanged
      n - the position of the second node to be interchanged
Output: none

1. If m equals n, return (no changes needed)

2. Initialize current to head
3. Initialize prevM, nodeM, prevN, and nodeN to null

4. Traverse the list to find the mth node:
   a. For i from 1 to m-1
      i. prevM <- current
      ii. current <- current.next
   b. nodeM <- current

5. Traverse the list to find the nth node:
   a. current <- head
   b. For i from 1 to n-1
      i. prevN <- current
      ii. current <- current.next
   c. nodeN <- current

6. If prevM is not null
   a. prevM.next <- nodeN
   Else
   b. head <- nodeN

7. If prevN is not null
   a. prevN.next <- nodeM
   Else
   b. head <- nodeM

8. Swap the next pointers of nodeM and nodeN:
   a. temp <- nodeM.next
   b. nodeM.next <- nodeN.next
   c. nodeN.next <- temp
```