Sri Jay Adarsh Gogineni

RUID 171003252

Problem 2)

1.

This code achieves O(1) time for remove() because it always removes from the front of the queue. We get constant time because we always know where the head of the queue is. All we have to do for the remove function is save the data in a temp variable and make the next Node the new head. Since the head is always known, we can do this with O(1).

**public** **int** remove(){

**int** temp = head.data;

head = head.next;

**return** temp;

}

For getValue(index I) this particular code achieves O(n) time. This occurs because every time we want a certain value, we have to traverse through the queue to find it. So depending on the number of elements in the queue, we have to traverse that many times. This results in O(n) time.

**public** **int** getValue(**int** i){

**int** temp = 0;

**int** counter = 1;

Node n = head;

**while**(n!=**null** && counter <i){

n = n.next;

counter++;

}

temp = n.data;

**return** temp;

}

2.

This code achieves O(n) time to add because of a similar explanation as above. Since every time we want to add a Node, we have to traverse the entire queue until we find the last node which is pointing to null. This traversing aspect of the add function makes it so that depending on the number of elements in the queue, that is how many times we will have to traverse to find the position to add. This results in O(n) time complexity.

**public** **void** add(**int** a){

Node last = **new** Node();

last.data = a;

last.next = **null**;

**if**(head == **null**){

head = last;

}**else**{

Node n = head;

**while**(n.next!=**null**){

n = n.next;

}

n.next = last;

}

As shown, we first traverse until we find where the last node is, and then add.