Name: An Le

Account: masc0369

Class: CS 310

Prof. Alan Riggins

**Project 1 Report**

**1) addFirst:**

**public** **void** addFirst(E obj){

Node<E> newNode = **new** Node<E>(obj);

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

head = tail = newNode;

**else** {

newNode.next = head;

head = newNode;

}

currentSize++;

}

The method always does the same amount of work regardless of the input size. Therefore:

F = O(1)

|  |  |
| --- | --- |
| **Elements** | **Time (ms)** |
| 1000 | 152 |
| 2000 | 91 |
| 3000 | 9 |
| 4000 | 6 |
| 5000 | 5 |
| 6000 | 6 |
| 7000 | 7 |
| 8000 | 8 |
| 9000 | 8 |
| 10000 | 11 |

**2) addLast:**

**public** **void** addLast(E obj){

Node<E> newNode = **new** Node<E>(obj);

**if** (tail == **null**)

head = tail = newNode;

**else** {

tail.next = newNode;

tail = newNode;

}

currentSize++;

}

The method always does the same amount of work regardless of the input size. Therefore:

F = O(1)

|  |  |
| --- | --- |
| **Elements** | **Time (ms)** |
| 1000 | 9 |
| 2000 | 4 |
| 3000 | 3 |
| 4000 | 4 |
| 5000 | 3 |
| 6000 | 3 |
| 7000 | 3 |
| 8000 | 3 |
| 9000 | 4 |
| 10000 | 8 |

**3) removeFirst :**

**public** E removeFirst(){

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

**return** **null**;

E tmp = head.data;

**if** (head == tail)

head = tail = **null**;

**else**

head = head.next;

currentSize--;

**return** tmp;

}

The method always does the same amount of work regardless of the input size. Therefore:

F = O(1)

|  |  |
| --- | --- |
| **Elements** | **Time (ms)** |
| 1000 | 52 |
| 2000 | 36 |
| 3000 | 20 |
| 4000 | 18 |
| 5000 | 15 |
| 6000 | 18 |
| 7000 | 20 |
| 8000 | 12 |
| 9000 | 15 |
| 10000 | 13 |

**4) removeLast** :

**public** E removeLast(){

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

**return** **null**;

E tmp = tail.data;

Node<E> previous = **null**, current = head;

**if** (current == tail) {

head = tail = **null**;

currentSize--;

**return** tmp;

}

**while** (current != tail){

previous = current;

current = current.next;

}

**if** (previous == **null**)

head = tail = **null**;

**else**{

previous.next = **null**;

tail = previous;

}

currentSize--;

**return** **tmp**;

}

Since the comparisons and assignments have O(1), we discard them and only focus on the loop. The while loop starts with current = head, and it loops until current = tail. Therefore, the loop always takes n comparisons for an n-element-list.

F = O(n)

|  |  |
| --- | --- |
| **Elements** | **Time (ms)** |
| 10 | 14 |
| 20 | 7 |
| 40 | 23 |
| 80 | 20 |
| 160 | 32 |
| 320 | 63 |
| 640 | 127 |
| 1280 | 302 |
| 2560 | 601 |
| 5120 | 1205 |

**5) find:**

**public** E find(E obj){

Node<E> tmp = head;

**while** (tmp != **null**){

**if** (((Comparable<E>)obj).compareTo(tmp.data) == 0) **return** tmp.data;

tmp = tmp.next;

}

**return** **null**;

}

The while loop stops when “obj” is found or the next node is null. In the worst case, it will loop n times; in other words, it will loop from head to tail. Therefore

F = O(n)

|  |  |
| --- | --- |
| **Elements** | **Time (ms)** |
| 1000 | 7 |
| 2000 | 4 |
| 4000 | 6 |
| 8000 | 14 |
| 16000 | 28 |
| 32000 | 52 |
| 64000 | 118 |
| 128000 | 264 |
| 256000 | 534 |
| 512000 | 1066 |

**6) contains:**

This method has the same algorithm to “find” method except it returns Boolean value instead of the found element. And so, its complexity is:

F = O(n)

**7) remove:**

**public** **boolean** remove(E obj){

Node<E> previous = **null**, current = head;

**while** (current != **null** && ((Comparable<E>)obj).compareTo(current.data) != 0){

previous = current;

current = current.next;

}

**if**(current == **null**)

**return** **false**;

**if** (current == head) {

head = head.next;

currentSize--;

}

**else** **if**(current == tail){

previous.next = **null**;

tail = previous;

currentSize--;

}

**else** {

previous.next = current.next;

currentSize--;

}

**return** **true**;

}

The while loop of this method also loops from head node until a null node is hit or a matched node is found. So, in the worst case, the list may not contain the element we are looking for, or the element is the tail of the list. In both cases, the loop takes n comparisons. So, we have

F = O(n)

|  |  |
| --- | --- |
| **Elements** | **Time (ms)** |
| 1000 | 8 |
| 2000 | 4 |
| 4000 | 7 |
| 8000 | 14 |
| 16000 | 28 |
| 32000 | 51 |
| 64000 | 121 |
| 128000 | 247 |
| 256000 | 520 |
| 512000 | 1052 |

**8)**

|  |  |
| --- | --- |
| * peekFirst * peekLast * makeEmpty * isEmpty * isFull * size * iterator | The method always does the same amount of work regardless of the input size. Therefore:  F = O(1) |