CSE 12 Homework 2

Name: Chih-Hsuan Kao

PID: A92092372

Login: cs12sgh

Date: April 24, 2018

Email: c4kao@ucsd.edu

File: README.txt

In this file, I will briefly describe what each of the tests in MyLinkedList.java are attempting to validate, as well as any known problems or assymptions made in my classes or program.

The following tests are in MyLinkedListTester.java

testGetSize()

firstly check if the size of MyLinkedList 'a' has the size of 100 as I assumed.

testGetSize2()

firstly check if the size of MyLinkedList 'b' has the size of 100 as I assumed.

testGetFibonacci()

provided in the starter code

testSmall3Add()

small3 has size of 3 and has element 0 to 2. This test verifies its exception on IndexOutOfBounds by adding a new element out of index.

testGetException()

list 'a' has 100 index. This test verifies its exception caught when trying to access its 101 element.

testContent()

list 'b' has 100 elements from 0 to 99 on index 0 to 99. This test randomly choosing some values at specific indexes to verify its content is correct.

testClear()

verifying if clear() method is functioning correctly, and then verifies nothing can be removed anymore by calling remove()

testListSet()

setting new elements to small3 list and see if it sets correctly by comparing what I expect and what it actually has.

testSetNullExcep()

see if passing a null element to a list generates NullPointerException as expected.

testEmp()

check emptiness by checking size, without calling isEmpty() method.

testGetHead()

testing if the heads of the lists are correct. Provided in the starter code.

testListSize()

Provided in the starter code. Test if size of lists are correct.

testSet()

Provided in the starter code. Test setting a specific entry.

testEmpty()

Provided in the starter code. Test isEmpty method.

testGetException()

Provided in the starter code. Test out of bounds exception on get.

testIterAddException()

Test if the add method of iterator throws a nullpointerexception when passing in a null element.

testIterNextException()

Test when the iterator reaches the end, call next would throw nosuchelementexception.

testIterRemoveException()

Test when calling remove without calling next and previous, an IllegalStateException would be thrown.

testIterRemoveAfterSet()

Test if remove method still works after calling set(). It is supposed to perform what I expected by using assertion.

testSetAfterAdd()

Test if set method works without calling previous or next. An IllegalStateException should be thrown.

testIterPrevIndexAtStart()

Test if next and previous methods work properly by accessing the index the iterator returns.

testIterAdd()

Creates a new empty linkedlist and inserts new elements step by step using iterator. Also, traverse throught the linked list using iterator and see if adding and traversing are functioning correctly as expected.

testIterRemove()

Traverse through the list and reach the end, remove every element. See if remove() is functioning correctly by checking the list size after removal.

testIterRemoveReverse()

Traverse throught the list reversely from the back. Remove from the back to the very beginning. See if removal is performed correctly. Check list size at the end.

testSetIte()

Use iterator on the 'several' list and see if the set method is functioning correctly.

testIterator()

Provided in the starter code. Test iterator on empty list and several list.

testIteratorFibonacci()

Provided in the starter code. Test iterator on Fibonacci list.

RUNTIME

1.

Running Time: O(n)

Explanation:

There is a single loop that runs [int((n+1)/2)] times. Each time the loop runs, it executes 1 instruction. So the total number of instruction executed is 1*[int((n+1)/2)] = O(n).

2.

Running Time: O(log n)

Explanation:

Note that i value grows logarithmically. There is a single loop that runs [int(log(n-1)) + 1] with log base of 2. Each time the loop runs, it executes 1 instruction. So the total number of instruction executed is 1*[int(log(n-1) + 1] = O(log n) w/log base of 2.

3.

Running Time: O(n^2)

Explanation:

The outer loop is executed for n values; and the inner loop is also executed for n values. For each value that the outer loop runs, the inner loop is executed for another n times. So the total number of instruction executed is n*n = n^2 = O(n^2).

4.

Running Time: O(n)

Explanation:

Note that the two loops are independent to each other, so needs to be considered separately.

For the first loop, it runs n times to execute one instruction (sum++); similarly, for the second loop, it also runs n times to execute the only instruction (sum++). They both have time complexity of O(n).

The total number of instruction executed is 2*O(n) = O(n).

5.

Running Time: O(n)

Explanation:

There is a single loop that runs 2*n times. Each time the loop runs, it executes 1 instruction. So the total number of instruction executed is 2*n = O(n).

6.

Running Time: O(n^2)

Explanation:

There is a single loop that runs n*n times. Each time the loop runs, it executes 1 instruction. So the total number of instruction executed n^2 times = O(n^2).

7.

Running Time: O(n^3)

Explanation:

The outer loop is executed for n values; and the inner loop is executed for n^2 values. There is one instruction to be executed every time going inside the inner loop. For each value that the outer loop runs, the inner loop is executed for another n*n times. So the total number of instruction executed is n*n*n = n^3 = O(n^3).

8.

Running Time: O(n)

Explanation:

The outer loop is executed for n values; and the inner loop is executed for 10000 times. There is one instruction to be executed every time going inside the inner loop. For each value that the outer loop runs (there are n values), the inner loop is executed for another 10000 times. So the total number of instruction is 10000*n = O(10000*n) = O(n).