Thinh Le  
Lab 6  
04/10/2019

(1) (25 pts) The add method in IntArrayBag that we discussed in our class.

1) public void add(int element)

2) {

3) if (manyItems == data.length)

4) {

5) int biggerArray[ ];

6) biggerArray = new int[manyItems\*2 + 1];

7) for(int i=0;i < manyItems;i++){

8) biggerArray [i] = data[i];

9) }

10) data = biggerArray;

11) }

12) data[manyItems] = element;

13) manyItems++;

14) }

Analysis:

Let n = manyItems. We want to represent the total number of basic operations of this method as a function of n.

This method contains 2 parts.

The first part is from line 3 to 11.

* It contains an ‘if’ statement. In the worst case, which means when the ‘if’ condition is true, the statement is executed 1 time.
* It also contains 3 operations (line 5, 6 and 10) and a loop.
* In the worst case the loop iterates n times and in each loop, it needs approximately 3 operations: 1 updates i, 1 compares i to manyItems, 1 compares biggerArray[i] to data[i].
* The worst case running time for line 3 – 11 is 3n+4

The second part is from line 12- 13 and it contains 2 operations. In the worst case, these operations each operates once

The overall operations from 2 parts is 3n+6. In Big-O, the time complexity is O(n).

(2) (25 pts) A method to count the number of occurrences of a particular element target. This method is

implemented in the IntArrayBag class that we discussed in class.

1) public int countOccurrences(int target)

2) {

3) int answer = 0 ;

4) int index;

5) answer = 0;

6) for (index = 0; index < manyItems; index++)

7) if (target == data[index])

8) answer++;

9) return answer;

10) }

Analysis:

Let n = manyItems. We want to represent the total number of basic operations of this method as a function of n.

This method contains 3 parts.

The first part (line 3-5) contains 3 operations that execute in any case.

The second part (line 6-8) contains a ‘for’ loop. In the worst case, the ‘for’ loop iterates n times. And in each loop, it needs approximately 4 operations: 1 updates index, 1 compares index to manyItems, and then 1 ‘if’ to compare target with data [index], and 1 updates answer. That’s in the worst case.

The third part (line 9) only contains 1 operation: return.

Overall, the method operations from 3 parts is: 4n+4. So the time complexity Big-O is O(n).

(3) (25 pts) A method to find a node at a specified position in a linked list starting from the given head.

This method is implemented in the IntNode class that we discussed in class.

1) public static IntNode listPosition(IntNode head, int position)

2) {

3) IntNode cursor;

4) int i;

5) if (position <= 0)

6) throw new IllegalArgumentException("position is not positive");

7) cursor = head;

8) for (i = 1; (i < position) && (cursor != null); i++)

9) cursor = cursor.link;

10) return cursor;

11) }

Analysis:

Let n = position. We want to represent the total number of basic operations of this method as a function of n.

This method contains 5 parts.

The first part (line 3-4) is the 2 operations that execute in any case.

The second part (line 5-6) is the if statement and throw statement. In the worst case, the condition in the if statement is executed once and the throw operation doesn’t get executed.

The third part (line 7) is just 1 operation that set cursor.

The fourth part (line 8-9) contains a loop. In the worst case, this loop iterates n times. And in each loop, it needs approximately 5 operations:

* 1 updates i
* 1 compares i to position
* 1 checks if cursor is not null
* 1 gets the link to the next cursor
* 1 moves the cursor to the next node.

Finally, we have 1 operation at the end.

Totally, the method adds up to 5n + 4 operations. So Big-O is O(n)

(4) (25 pts) A method to compute the number of nodes in a linked list starting from the given head. This

method is implemented in the IntNode class that we discussed in class.

1) public static int listLength(IntNode head)

2) {

3) IntNode cursor = null;

4) int answer = 0;

5) for (cursor = head; cursor != null; cursor = cursor.link)

6) answer++;

7) return answer;

8) }

Analysis:

Let n = the total number of nodes in the linked list. We want to represent the total number of basic operations of this method as a function of n.

This method contains 3 parts.

The first part (line 3-4) contains 2 operations that execute in any case.

The second part (line 5-6) contains a ‘for’ loop that executes n times. It means that the head given is the real head of the linked list. In each loops it needs approximately 3 operations:

* 1 updates cursor
* 1 checks if cursor is different from null
* 1 increases answer

The third part contains 1 operation: return

Overall, the methods add up to 3n + 3 operations. Thus, the time compexity Big-O is O(n).