Lab 6 – Time Complexity

1. The add method in IntArrayBag that we discussed in our class.

1 if (manyItems == data.length){

This operation is O(1) constant as it is only making one check of equivalency.

2 int biggerArray[];

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

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

5 biggerArray[i] = data[i];

6 }

7 data = biggerArray;

8 }

***Let n*** be defined as manyItems, how many items are currently in the bag, and the following is representing the total number of basic operations of the method as a function of n.

This method contains three parts.

The first part is lines 1-3.

* At line 1, the operation is O(1) constant as it is only making one check of equivalency.
* At line 2, it is declaring an array, so it is a constant O(1) operation.
* At line 3, it is initializing biggerArray[] with an instantiated array, and its size is being calculated with two mathematical operations, which in total is 4 operations, but is reduced to a constant at O(1).

The second part is lines 4-5

* The for loop assigns 0 to index which is O(1), and checks and increments n times, which is 2n.
* This is a for loop that iterates n times, as our n is manyItems, and since it is not a nested loop, and performs a constant number of operations inside (assignment), then the complexity is O(n) as it goes through n amount of times.

The third part is line 7, it involves one operation, which is setting the biggerArray’s pointer to data’s pointer, effectively changing the array pointer. Assignment is a constant operation, so it is O(1).

The overall number of operations adds up to the number of operations in each part:

6 + 1 + 2n + n + 1 = n + 7

In Big-O notation from the equation above, the time complexity is O(n).

1. 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 int answer = 0;

2 int index;

3 answer = 0;

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

5 if (target == data[index])

6 answer++;

7 return answer;

***Let n*** be defined as manyItems, how many items are currently in the bag, and the following is representing the total number of basic operations of the method as a function of n.

This method contains three parts.

The first part is lines 1-3

* Line 1 has one operation, which is assignment, which is constant at O(1).
* Line 2 has one operation, which is initializing an index reference, O(1).
* Line 3 is assigning 0 to the variable answer, which is constant at O(1).

The second part is lines 4-6

* Line 4 assigns 0 to index, which is O(1), and does an inequality check and increment n amount of times which is 2n.
* Line 5 does an equivalency check, which is constant at O(1).
* Line 6 increments which is constant at O(1).

The third part is line 7 which involves one return statement, which is constant at O(1).

The overall number of operations adds up to the number of operations in each part:

3 + 1 + 2n + 2n + 1 = 4n + 5

In Big-O notation from the equation above, the time complexity is O(n).

1. A method to find a node at a specified position in a linked list starting from the head. This method is implemented in the IntNode class that we discussed in class.

1 IntNode cursor;

2 int i;

3 if (position <= 0)

4 throw new IllegalArgumentException(“position is not

positive”);

5 cursor = head;

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

7 cursor = cursor.link;

8 return cursor;

***Let n*** be defined as the amount of nodes in the linked list including the head, and the following is representing the total number of basic operations of the method as a function of n.

This method contains three parts.

The first part is lines 1-5

* Line 1 initializes a reference to an IntNode object, which is constant at O(1).
* Line 2 initializes a reference to an int, which is constant at O(1).
* Line 3 does an inequality or equivalence check, which which constant at O(1).
* Line 4 throws a new IllegalArgumentException, which is both a throw and instantiation operation, which is 2 operations, but a constant at O(1).
* Line 5 sets the cursor memory address to the head memory address, which is a constant at O(1).

The second part is lines 6-7.

* Line 6 assigns 1 to variable i, which is one operation at constant O(1), and also does 3 inequality checks: a less than, not equal, and a logical and. The three checks are accompanied by an increment to i, which all run n amount of times, resulting in 4n operations.
* Line 7 is assigning the cursor reference the address to cursor.link, which is a constant O(1), but runs n amount of times inside the loop for O(n).

The third part is a return operation, returning the cursor, which is constant at O(1).

The overall number of operations adds up to the number of operations in each part:

6 + 4n + n + 1 = 5n + 7

In Big-O notation from the equation above, the time complexity is O(n).

1. 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 discuss in class.

1 IntNode cursor = null;

2 int answer = 0;

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

4 answer++;

5 return answer;

***Let n*** be defined as the amount of nodes in the linked list including the head, and the following is representing the total number of basic operations of the method as a function of n.

This method contains three parts.

The first part is lines 1-2

* Line 1 initializes a reference to an IntNode object and assigns null to it, 2 operations which is constant at O(1).
* Line 2 initializes a reference to an int and assigns it 0, 2 operations which is constant at O(1).

The second part is lines 3-4.

* Line 3 assigns the head address to variable cursor, which is one operation at constant O(1), and also does a logical inequality check. The check is accompanied by an assignment of the cursor.link address to cursor, which all run n amount of times, resulting in 2n operations.
* Line 4 is incrementing answer, which is a constant O(1), but runs n amount of times inside the loop for O(n).

The third part is a return operation, returning the answer variable, which is constant at O(1).

The overall number of operations adds up to the number of operations in each part:

4 + 1 + 2n + n + 1 = 3n + 6

In Big-O notation from the equation above, the time complexity is O(n).