# Lab 6: Running time Analysis

I. Requirements: Please ANALYZE the worst-case running time of the following methods, WRITE down your analysis in

DETAIL, and denote their time complexity in Big-O.

Hint: You need to define n first, before showing whether the method is O(n), O(logn), O(n2), etc.

1) (25 pts) The add method in IntArrayBag:

```
(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)}
```

**Solution:**

(1pts) Let n=manyItems. We measure the complexity of this function as a function of n

(2pts) Line (3): if condition, it needs 2 operations (data.length and compare manyItems with data.length).

(2pts) Line (5): 1 operation

(2pts) Line (6): 2n+1 operations for allocating (2n+1) space.

(9pts) Line (7)-(9): In worst case, this loop iterates n times. Each iteration, there are 3 operations: update I, compare i with manyItems, check biggerArray[i] equals to data[i] => total 3n.

(2pts) Line (10): 1 operation

(2pts) Line (12): 1 operation

(2pts) Line (13): 1 operation

(3pts) The total number of operations: 2+1+2n+1+3n+3 = 5n+4. In Big-O, its time complexity is O(n)

-1pts: if not declare n

-2pts: do not show the correct operations for line (6)

-9pts: do not show the detailed analysis for the loop (-3 if the total is incorrect).

-12.5pts: do not show the detailed analysis

(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)}

**Solution:**

(1pts) Let n=manyItems.

(9pts/3pts each each)Line (3)-(5): 3 operations

(9pts)Line (6)-(8): For loop, in the worst case, it iterates n times. In each iteration it needs approximately 4 operations: update i, compare I with manyItems, check data[index] equals target, increase answer. Worst case: 4n

(3pts)Line (9): 1 operation.

(3pts)Total: 4n+4 operations. In Big-O, its time complexity is O(n).

-1pts: if not declare n

-9pts: do not show the detailed analysis for the loop (-3 if the total is incorrect).

-2pts: do not show the correct Big-O.

-12.5pts: do not show the detailed analysis

Q3

```
1. public static IntNode listPosition(IntNode head, int position) {
2. IntNode cursor;
3. int i;
4. if (position <= 0)
5. throw new IllegalArgumentException("position is not positive");
6. cursor = head;
7. for (i = 1; (i < position) && (cursor != null); i++)
8.      cursor = cursor.link;
9. return cursor;
10.  }
```

Here, n is the number of nodes in the the list.
1. line 1 is only declaration
2. line 2 is only declaration
3. line 3 is only declaration
4. line 4 has one operation
5. line 5 has one operation
6. line 6 has one operation
7. in the for loop, i=1 will be initialize only once. in worst case, i< position, cursor != null and i++ will run n times.  So  time complexity will be 3n+1
8. line 8 is only one operation.
9. it is only one operation

Worst case time complexity= 0(n)

rubric
5 pts if define the n properly
10 pts if define the operation cost properly line by line
10 pts if find the total worst time complexity correctly

(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 }
```

Solution:
(1 pts) Let n is the size of the linked list
(4 pts) Line 3 and 4 are both assignment operations, they take 2 operations.
(15 pts) Line 5-6, this is a loop through all elements of the linked list, hence it takes n times to iterate through the linked list. In line 5, initialization takes 1 operation (2 pts), comparison and increasing cursor take n operations each (6 pts). In line 6, answer increasing by 1, this takes n operations (7 pts),. Therefore, these two lines take 3n+1 operations.
(3 pts) Line 7, returning the value of answer variable takes 1 operation.
(2 pts) Conclusion, the whole function takes 2 + (3n + 1)  + 1 = 3n + 4 operations, its time complexity is O(n).